Avizo® 9

Avizo User's Guide

Contents

Chapter 1

Introduction

Avizo is a 3D data visualization, analysis and modeling system. It allows you to explore and analyze data sets from various areas, including:

- Scientific visualization, physics, chemistry, astrophysics, archaeology, and others
- Material sciences, non-destructive testing, tomographic imaging and microscopy
- Computer-aided Engineering, computational fluid dynamics, numerical simulation
- Oil and gas, mining, Earth science
- Climate, oceanography, environment

Examples from these disciplines are illustrated by several *demo scripts* contained in the online version of the user's guide.

3D data can be quickly explored, analyzed, compared, and quantified. 3D objects can be represented as image volumes or geometrical surfaces and grids suitable for numerical simulations, notably as triangular surface and volumetric tetrahedral grids. Avizo provides methods to generate such grids from voxel data representing an image volume, and it includes a general-purpose interactive 3D viewer.

Section 1.1 (Overview) provides a short overview of the fundamentals of Avizo, i.e., its object-oriented design and the concept of data objects and modules.

Section 1.2 (Features) summarizes key features of Avizo, for example direct volume rendering, image processing, and surface simplification.

Section 1.3 (Editions and Extensions) briefly describes optional extensions and editions available for Avizo and for what they can be used for.

Section 1.4 (System Requirements) provides system specific information.

Section 1.5 (Avizo License Manager) details for entering and managing Avizo license passwords.

Section (First steps) provides hints about tutorials included in this guide.

Section (Contact and Support) provides contact information for your technical support, license administrator, or sales representative.

1.1   Overview

Avizo is a modular and object-oriented software system. Its basic system components are modules and data objects. Modules are used to visualize data objects or to perform some computational operations on them. The components are represented by little icons in the *Project View*. Icons are connected by lines indicating processing dependencies between the components, i.e., which modules are to be applied to which data objects. Alternatively, modules and data objects can be displayed in a *Project Tree View*. Modules from data objects of specific types are created automatically from file input data when reading or as output of module computations. Modules matching an existing data object are created as instances of particular module types via a context-sensitive popup menu. Projects can be created with a minimal amount of user interaction. Parameters of data objects and modules can be modified in Avizo's interaction area.

For some data objects such as surfaces or colormaps, there exist special-purpose interactive editors that allow the user to modify the objects. All Avizo components can be controlled via a Tcl command interface. Commands can be read from a script file or issued manually in a separate console window.

The biggest part of the screen is occupied by a 3D graphics window. Additional 3D views can be created if necessary. Avizo is based on the latest release of Open Inventor by FEI, SAS. In addition, several modules apply direct OpenGL rendering to achieve special rendering effects or to maximize performance. In total, there are more than 270 data object and module types. They allow the system to be used for a broad range of applications. Scripting can be used for customization and automation. User-defined extensions are facilitated by the Avizo developer version.

1.2   Features overview

Avizo provides a large number of data types and modules allowing you to visualize, analyze and model various kinds of 3D data. The Avizo framework is ideal to integrate the data from multiple sources into a single environment.

This section summarizes the main features of the Avizo software suite. For more complete information, you may browse indexes for data types, file formats and modules in the Reference Guide. This is accessible from the home page of the online help browser.

Section describes the Avizo optional editions and extensions.

1.2.1 Data import

Avizo can load directly different types of data, including:

- 2D and 3D image and volume data
- Geometric models such as point sets, line sets, surfaces, grids
- Numerical simulation data
- Time series and animations

A large number of file formats are supported in Avizo or through specific optional readers. For an introduction to data import, see section 3.1. For more details, see section 2.6.

1.2.2 Viewing, navigation, interactivity

All visualization techniques can be arbitrarily combined to produce a single scene. Moreover, multiple data sets can be visualized simultaneously, either in several viewer windows or in a common one. Thus you can display single or multiple data sets in a single or multiple viewer windows, and navigate freely around or through those objects. Views can be synchronized to facilitate comparisons.

A built-in spatial *transform editor* makes it easy to register data sets with respect to each other or to deal with different coordinate systems. Automatic alignment and registration of image or geometric data is also possible.

Direct interaction with the 3D scene allows you to quickly control regions of interest, slices, probes and more.

Combinations of data sets, representation and processing features can be defined with minimal user interaction for simple or complex tasks. See section 1.1.

1.2.3 Visualization of 3D Image Data

1.2.3.1 Slicing and Clipping

You can quickly explore 3D images looking at single or multiple orthographic or oblique sections. Data sets can be superimposed on slices, displayed as height fields, or with isolines contouring. You can cut away parts of your data to uncover hidden regions. Curved or cylinder slices are also available.

1.2.3.2 Volume Rendering

One of the most intuitive and most powerful techniques for visualizing 3D image data is *direct volume rendering*. Light emission and light absorption parameters are assigned to each point of the volume. Simulating the transmission of light through the volume makes it possible to display your data from any view direction without constructing intermediate polygonal models. By exploiting modern graphics hardware, Avizo is able to perform direct volume rendering in real time, even on very large data when

using the Avizo XLVolume Extension. Thus volume rendering can instantly highlight relevant features of your data. Volume rendered images can be combined with any type of polygonal display. This improves the usefulness of this technique significantly. Moreover, multiple data sets can be volume rendered simultaneously – a unique feature of Avizo. Transfer functions with different characteristics required for direct volume rendering can be either generated automatically or edited interactively using an intuitive colormap editor. Avizo volume rendering can use the latest techniques for high quality visualization effects such as lighting or shadows.

1.2.3.3 Isosurfaces

Isosurfaces are most commonly used for analyzing arbitrary scalar fields sampled on discrete grids. Applied to 3D images, the method provides a very quick, yet sometimes sufficient method for reconstructing polygonal surface models. Beside standard algorithms, Avizo provides an improved method, which generates significantly fewer triangles with very little computational overhead. In this way, large 3D data sets can be displayed interactively even on smaller desktop graphics computers.

1.2.3.4 Large Volume Data

With the Avizo XLVolume Extension, even very large data sets that cannot be fully loaded in memory can be manipulated at interactive speed. Multi-resolution techniques can manage and visualize extremely large amounts of volume data of up to hundreds of gigabytes. You can then, for instance, quickly select a region of interest and extract down-sampled or partial data for further processing.

1.2.4 Image processing

1.2.4.1 Alignment of image slices

The image *Align Slices* enables you to build a consistent stack of images with manual or automatic tools, if, for instance, physical cross sections have been shifted during image acquisition.

1.2.4.2 Image filters

Image features can be enhanced by applying a wide range of filters for controlling contrast, smoothing, noise reduction and feature enhancement. See Section 4.7 More about Image Filtering.

1.2.4.3 Image segmentation

Segmentation means assigning labels to image voxels that identify and separate objects in a 3D image. Avizo offers a large set of segmentation tools, ranging from purely manual to fully automatic: brush (painting), lasso (contouring), magic wand (region growing), thresholding, intelligent scissors, contour fitting (snakes), contour interpolation and extrapolation, wrapping, smoothing and de-noising filters,

morphological filters for erosion, dilation, opening and closing operations, connected component analysis, images correlation, objects separation and filtering, etc. See section 3.5 for a tutorial about image segmentation with Avizo.

1.2.4.4 Image quantification and analysis

Avizo provides tools for probing image data, extracting profiles, value or correlation histogram. It can extract information from segmented images such as area, volumes, intensity statistics. Avizo provides an extensive set of tools for image quantification and analysis. See *Advanced Image Processing, Segmentation and Analysis* examples.

1.2.5 Model reconstruction

1.2.5.1 Surface generation

Once the interesting features in a 3D image volume have been segmented, Avizo is able to create a corresponding polygonal surface model. The surface may have non-manifold topology if there are locations where three or more regions join. Even In this case, the polygonal surface model is guaranteed to be topologically correct, i.e., free of self-intersections. Fractional weights that are automatically generated during segmentation allow the system to produce optionally smooth boundary interfaces. This way realistic high-quality models can be obtained, even if the underlying image data are of low resolution or contain severe noise artifacts. Making use of innovative acceleration techniques, surface reconstruction can be performed very quickly. Moreover, the algorithm is robust and fail-safe.

1.2.5.2 Surface Simplification, Editing, Remeshing

Surface simplification is another prominent feature of Avizo. It can be used to reduce the number of triangles in an arbitrary surface model according to a user-defined value. Thus, models of finite-element grids, suitable for being processed on low-end machines, can be generated. The underlying simplification algorithm is one of the most elaborate available. It is able to preserve topological correctness, i.e., self-intersections commonly produced by other methods are avoided. In addition, the quality of the resulting mesh, according to measures common in finite element analysis, can be controlled. For example, triangles with long edges or triangles with bad aspect ratio can be suppressed.

A surface editor is also available for smoothing or refining surface in whole or part, cutting and copying parts of surfaces, defining boundary conditions for further numerical simulation, checking and modifying surface triangles.

Surface Path Set can be created interactively - for instance as geodesic contours, edited and used to cut surface patches. Surface path can also be obtained by intersecting surfaces,

Avizo and *Avizo XWind Extension* also provide a powerful *Remesh Surface* module that can be used for producing high quality surfaces.

1.2.5.3 Generation of Tetrahedral Grids

Avizo allows you not only to generate surface models from your data but also to create true volumetric tetrahedral grids suitable for advanced 3D finite-element simulations. These grids are constructed using a flexible advancing-front algorithm. Again, special care is taken to obtain meshes of high quality, i.e., tetrahedra with bad aspect ratio are avoided. Several different file formats are supported, so that the grid can be exported to many standard simulation packages.

1.2.5.4 Point Clouds/Scattered Data

Avizo can also reconstruct surfaces from scattered points (see *Delaunay Triangulation* and *Point Wrap Triangulation* modules).

1.2.5.5 Skeletonization

A set of tools is included for reconstructing and analyzing a dendritic, porous or fracture network from 3D image data.

1.2.6 Visualization of 3D models and numerical data

1.2.6.1 Point sets, line sets

Avizo can visualize arbitrary functional data given on 3D *Point Cloud* sets or *Line Sets*.

1.2.6.2 Polygonal models

A number of drawing styles and coloring schemes help to yield meaningful and informative visualizations of polygonal models, whether generated from image data or imported from CAD or simulation package. Surface and 3D grid meshes can be colored or textured in order to visualize a second independent data set.

Another Avizo feature comprises the realistic view-dependent way of rendering semi-transparent surfaces. By correlating transparency with local orientation of the surface relative to the viewing direction, complex spatial structures can be understood much more easily.

1.2.6.3 Numerical data post-processing

Avizo allows you to analyze numerical data coming from measurements or simulations. Avizo supports polygonal surfaces such as triangular meshes, 3D lattices with uniform, rectilinear of curvilinear coordinates, and 3D tetrahedral or hexahedral grids. Most general purpose image visualization techniques and analysis tools can be applied, such as: slice extraction, computation of isolines or isosurfaces, data probing and histograms. In addition, scalar quantities can be visualized with pseudo-colors on the grid itself.

Beside visualization, data representations such as isosurfaces, grid cuts or contour lines can be extracted as first class data objects.

Displacement vectors can be visualized on grids or applied as grid deformation that can be animated. *Avizo XWind Extension* provides extensive additional support for numerical data import/export, visualization and analysis.

1.2.6.4 Flow Visualization

Avizo provides many advanced tools for vector fields and flow visualization. Vector arrows can be drawn on a slice, within a volume, or upon a surface. The flow structure may be better revealed by representations such as fast Line Integral Convolution on slices or arbitrary surfaces, illuminated and animated streamlines, stream ribbons, stream surfaces, particle animations, synthetic *Vector Probe...* All of these stream visualization techniques are highly interactive. While seed point distributions can be automatically calculated, you can also select and interactively manipulate seed points and structures, thus supporting the investigation of the flow field and highlighting of different features. Avizo can also support six-component complex vector fields and phase visualization, e.g., electromagnetic fields.

Avizo XWind Extension provides extensive support for flow and CFD post-processing.

1.2.6.5 Tensor Data

Avizo has support for iconic visualization of tensor field, extraction of eigenvalues, computation of rate of strain tensor, and gradient tensor.

Avizo XWind Extension can compute many secondary variables from simulation results including various tensor.

1.2.7 General Data Processing and Data Analysis

1.2.7.1 3D registration of multiple data sets

Multiple data sets can be combined to compare images of different objects, or images of an object recorded at different times or with different imaging modalities such as X-ray CT and MRI. In addition, fusion of multi-modal data by arbitrary arithmetic operations can be performed to increase the amount of information and accuracy in the models. Avizo allows manual registration through interactive manipulators, automatic rigid or non-rigid registration through landmarks, and automatic registration using iterative optimization algorithms (see *Register Images* module).

Surfaces can also be registered using rigid or non-rigid transformations, based on landmarks sets warping, alignment of centers or principal axes, or distance minimization algorithms.

1.2.7.2 Operating on 3D data

Many utilities are available for data processing. Here are some important ones. Resampling can reduce or enlarge the resolution of a 3D image or data sets defined on regular grids, and different sampling kernels are supported. Data can be cropped or regions of interest can be defined. Data can be converted to any supported primitive type, from byte to 64-bits floating point numbers. Multi-component data such as multi-channel images or vector data can be composed or decomposed. Standard 3D field operators such as scalar field gradient or vector field curl are available. Surface curvatures and distances between surfaces can also be computed, as scalar or vector information. The powerful *Arithmetic* module allows the user to perform calculations on data sets with user-defined expression, and can be used to interpolate data between regular grids and polyhedral grids. Data sets can also be created from arithmetic expressions.

1.2.7.3 Measurements, quantification

You can query the exact values of your data sets at arbitrary locations specified interactively by a mouse click, or along user-defined lines and spline curves. Probe points can serve to set interactively isosurfaces. You can plot or export the data for further processing with spreadsheet or plotting applications, with probing, measuring, counting, and other statistical modules quantify densities, distances, areas, volumes, mean value and standard deviation.

Histograms of values can be computed and plotted, possibly restricted to a region of interest.

The Avizo provides an extensive set of intensity and geometrical measurements on image or label data, either for individual labeled particles or as statistics. See *Advanced Image Processing, Segmentation and Analysis* examples.

1.2.8 Matlab integration

You can integrate complex calculus using Matlab software from The Mathworks, Inc. by means of the *Calculus MATLAB* module. This module provides connection to your Matlab server from your Avizo session, and executes Matlab computations directly on your Avizo data. It is also possible to import and export Matlab matrices to and from Avizo, and export Avizo surfaces to Matlab surfaces. See section 9.6.1.

1.2.9 High Performance Visualization

Avizo makes extensive use of graphics hardware for optimal performance and rendering quality on your system. Moreover, the *Avizo XScreen Extension* allows combining multiple graphics engines for advanced displays and high-performance requirements.

1.2.10 Automation, Customization, Extensibility

Tcl scripting

All Avizo components can be controlled via a Tcl command language interface. Tcl scripts are used for saving your work session. Tcl scripts also allow the advanced user to automate or customize tasks with Avizo for routine workflows, without the need for C++ programming. Custom Avizo modules with user interface can even be created as Tcl scripts. Avizo module behaviour and 3D interaction can be customized by using Tcl. Avizo can also be used for batch processing.

See Chapter 9.4, including a short introduction to the Tcl scripting language.

C++ programming

With the Avizo XScreen Extension, Avizo can also be extended by programmers. The Avizo XPand Extension permits creation of new custom components for Avizo such as file readers and writers, computation modules, and even new visualization modules, using the C++ programming language. New modules and new data classes can be defined as subclasses of existing ones. In order to simplify the creation of new custom extensions, a development wizard is included.

See the *Avizo XPand Extension User's Guide* for detailed information.

Template projects

Template projects can be used to ease repetitive tasks on a set of similar data. A template project consists of a backup of an original project that can be replicated on another data of the same type. See Chapter 9.1.1.

1.3   Editions and Extensions

Some features of Avizo are grouped for convenience into so-called *Extensions*. These extensions are used to define specific subsets such as the "Avizo XScreen Extension". Avizo Lite Edition contains default features always available and some extensions available as options requiring a specific license.

1.3.1   Editions

Currently, the following editions are available for Avizo 9.

Avizo Lite Edition. The application for scientific visualization. Avizo is a versatile and flexible application framework providing powerful 3D visualization capabilities helping understand more efficiently the meaning of scientific data. Researchers, scientists, Avizo delivers you advanced visualization techniques which allow you to gain detailed insight into your 3D data. It provides exactly the same level of capabilities as the previous **Avizo Standard** edition, extended by the enhancements and new features introduced by version 9, but excluding **Avizo** advanced image processing, analysis and quantification features, and other **Avizo** extension-specific features. Avizo Lite Edition includes natively the following extensions: Avizo XSkeleton Extension and Avizo XMesh Extension.

Avizo.   The application for Material Sciences.   For industrial tomography, crystallography, material microstructure evolution, modality inspection for nano-structure, non-destructive investigation and surface analysis, Avizo delivers the whole feature-set allowing Material Scientists to see

the invisible, and understand and present their data. It is the equivalent of previous **Avizo Fire** edition.

Avizo Inspect Edition. This is the application for Industrial Inspection and Materials R&D. The Avizo Inspect Edition streamlines the process of industrial inspection and materials design off-line, near-line, and in-line, including dimensional metrology with advanced measurements; an extensive set of inspection workflows, so-called *recipes* for quantification of indications, porosity, inclusions, and defects; fiber characterization of parts and materials; ability to do actual/nominal comparison by integrating CAD models; and reverse engineering workflows for additive manufacturing. The Avizo Inspect Edition includes all Avizo features as well as the following extensions: *Avizo XMetrology Extension*, *Avizo XRecipe Extension*, *Avizo XReporting Extension*, and Avizo XLVolume Extension.

1.3.2   Optional Extensions

Note: See section 1.4 System Requirements for system requirements and hardware platform availability.

Currently, the following extensions are available for Avizo 9:

- **Avizo XWind Extension** includes all features previously available in the *Avizo Wind* edition for advanced post-processing of simulation data (i.e., ranging from flow to thermal) and stress data; an extensive array of advanced visualization and analysis tools to CFD and multi-physics; mechanical and thermal engineering; manufacturing simulation and micro-structural prediction; and non-linear structural and geotechnical problems. It also includes an advanced meshing workroom for creating simulation inputs (tetrahedral mesh with boundary conditions) directly from a label image.

- **Avizo XFiber Extension** provides specific support for analyzing fibers, filaments, tunnels, and other networks or tree-like structures. This extension assists segmentation and analysis with automatic, semi-automatic, and interactive tools. Advanced modules allow for the detection and tracing of tube-like structures in various modality acquisitions such as X-ray microtomography. Various distributions and properties can be calculated and plotted. Segmented fibers can also be filtered based on properties and attributes.

- **Avizo XEarth Extension** includes dedicated models and workflows for visualization and computation with a SEG-Y reader for the exploration and analysis of geosciences data. A *Geophysics profile* has been defined to enable all **XEarth** extension features, such as the dedicated tree view, modules, colormaps, etc.

- **Avizo XGreen Extension** encompasses visualization and computation modules; and a set of geographical projections with a *NetCDF* reader dedicated to the analysis of climate, oceanography, environmental, and earth-mapped data. A *Climatology profile* has been defined to enable all **XGreen** extension features, such as the *NetCDF* import, dedicated modules, colormaps, etc. Please refer to the online help for documentation on this extension.

- **Avizo XPand Extension** allows for the creation of custom components, such as modules for

visualizing or processing data, file readers or file writers for using the C++ programming language. New modules and new data classes can be defined as subclasses of existing ones. To simplify the creation of new custom extensions, XPand includes a development wizard.

- **Avizo XScreen Extension** is designed to enable the use of Avizo's advanced data visualization and analysis features on arbitrary tiled screens or immersive VR configurations. It has built-in support for distributed rendering on a cluster system. Tracking capabilities allows for real immersive experience as well as interaction with the visualization. Please refer to the online help for documentation on this extension.

- **Avizo XLVolume Extension** manages and visualizes large amounts of volume data up to hundreds of gigabytes. The multi-resolution technique used in this extension allows for interactive visualization and navigation through large amounts of data. You will need this extension to visualize data greater than 8 GB.

- **Avizo XMolecular Extension** combines Avizo's strong capabilities for 3D data visualization with specific tools for molecular visualization and data analysis, such as molecular surfaces, molecular interfaces, or configuration density computation. Please refer to the online help for documentation on this extension.

- **Avizo XReadIGES Extension, Avizo XReadSTEP Extension, Avizo XReadCATIA5 Extension**. The Catia 5, IGES, and STEP readers allow for visualization of data coming from the CAD area. Coupled with XScreen, they provide the means to visualize CAD models in an immersive visualization or collaborative environment.

- **Avizo XLabSuite Extension** is an extension of Avizo providing numerical simulation capabilities to calculate physical properties of materials from the 3D image of a sample (for instance, scanned with CT, FIB/SEM, MRI, etc.). Material properties are directly computed from the segmented 3D image. The calculated materials properties include absolute permeability, molecular diffusivity, formation factor, electrical conductivity, and thermal conductivity.

- **Avizo XMetrology Extension** allows for the creation test plans consisting of set of measures onto the part.

- **Avizo XRecipe Extension** allows for the creation of user-defined *recipes*, for the automation of a complex scenario.

- **Avizo XReporting Extension** allows for the creation of reports, including snapshots and analysis results in an HTML template.

- **Avizo XPoreNetworkModeling Extension** allows for access to different statistics from a labeled and separated pore space, and allows for pore network code reading.

- **Avizo XBioFormats Extension** allows Avizo to read image and metadata from over 140 file formats. To facilitate data exchange between different software packages and organizations, Avizo XBioFormats Extension converts data stored in proprietary file formats into an open standard called the OME data model, in particular the OME-TIFF file format.

- **Avizo XDigitalVolumeCorrelation Extension** allows for access to several tools and tutorials to compute the displacement and strain map from volume images in reference and deformed states, in order to study the deformation of materials.

1.3.3   License keywords

The following table shows the license keyword associated with each of the Avizo extensions:

Avizo Extension	License keyword
Avizo XWind Extension	AvizoWind
Avizo XFiber Extension	AvizoXFiber
Avizo XEarth Extension	AvizoEarth
Avizo XGreen Extension	AvizoGreen
Avizo XPand Extension	AvizoXPand
Avizo XScreen Extension	AvizoXScreen (*)
Avizo XLVolume Extension	AvizoXLVolume
Avizo XReadCATIA5 Extension	AvizoXReadCATIA5
Avizo XReadIGES Extension	AvizoXReadIGES
Avizo XReadSTEP Extension	AvizoXReadSTEP
Avizo XLabSuite Extension	AvizoXLabSuite
Avizo XMetrology Extension	AvizoXMetrology
Avizo XRecipe Extension	AvizoXRecipe
Avizo XReporting Extension	AvizoXReporting
Avizo XPoreNetworkModeling Extension	AvizoXPoreNetworkModel
Avizo XBioFormats Extension	AvizoXBioFormats
Avizo XDigitalVolumeCorrelation Extension	AvizoXVolumeCorrelation

(*) With cluster configurations, Avizo XScreen Extension requires as many licenses as slave rendering nodes in the cluster configuration. Ex: A configuration with a master node driving a cluster of 4 nodes will require 4 Avizo XScreen Extension licenses.

For additional information about Avizo and its extensions, please refer to the Avizo web site, http://www.avizo3d.com.

1.4   System Requirements

Avizo runs on:

- Microsoft Windows 7/8/10 (64-bit).
- Linux x86_64 (64-bit). Supported 64-bit architecture is Intel64/AMD64 architecture. Supported Linux distribution is Red Hat Enterprise Linux 6 and Red Hat Enterprise Linux 7.
- Mac OS X El Capitan (10.11) and macOS Sierra (10.12).

Some of the *Editions and Extensions* are limited to some platforms:

- **Avizo XWind Extension**: support of Abaqus reader (`.odb` format) and STAR-CCM reader are available only on Microsoft Windows and Linux. The meshing workroom is only available on Microsoft Windows platform,
- **Avizo XScreen Extension** is supported only on Microsoft Windows and Linux, not Mac OS X,
- **Avizo XPand Extension** is supported only on Microsoft Windows and Linux, not Mac OS X,
- **Avizo XReadIGES Extension, Avizo XReadSTEP Extension, Avizo XReadCATIA5 Extension** are supported only on Microsoft Windows, not on Linux or Mac OS X,
- **Avizo XLabSuite Extension**: molecular diffusivity, formation factor and thermal conductivity computation are supported only on Microsoft Windows, not on Linux or Mac OS X.
- **Avizo XMetrology Extension** is supported only on Microsoft Windows, not on Linux or Mac OS X.
- **Avizo XRecipe Extension** is supported only on Microsoft Windows, not on Linux or Mac OS X.
- **Avizo XGreen Extension**: This is the last version of Avizo to support Avizo XGreen Extension on Microsoft Windows and Linux. Avizo XGreen Extension is not supported on Mac OS X.
- **Avizo XEarth Extension**: This is the last version of Avizo to support Avizo XEarth Extension on Microsoft Windows and Linux. Avizo XEarth Extension is not supported on Mac OS X.
- **Avizo XDigitalVolumeCorrelation Extension** is supported only on Microsoft Windows and Linux, not Mac OS X.

1.4.1 Prioritizing hardware for Avizo

1.4.1.1 Introduction

This document is intended to give recommendations about choosing a suitable workstation to run Avizo.

The four most important components that need to be considered are the graphics card (GPU), the CPU, the RAM and the hard drive.

The performance of direct volume rendering of large volumetric data or large triangulated surface visualization extracted from the data depends heavily on the GPU capability. The performance of image processing algorithms depends heavily on the performance of the CPU. The ability to quickly load or save large data depends heavily on the hard drive performance. And, of course, the amount of available memory in the system will be the main limitation on the size of the data that can be loaded and processed.

Because the hardware requirements will widely vary according to the size of your data and your workflow, we strongly suggest that you take advantage of our supported evaluation version to try working with one of your typical data sets.

In this document, the term Avizo refers to all Avizo editions and all Avizo extensions.

1.4.1.2 Graphics Cards

The single most important determinant of Avizo performance for visualization is the graphics card.

Avizo should run on any graphics system (this includes GPU and its driver) that provides a complete implementation of OpenGL 2.1 or higher (certain features may not be available depending on the OpenGL version and extensions supported). However, graphics board and driver bugs are not unusual.

The amount of GPU memory needed depends on the size of the data. We recommend a minimum of 1 GB on the card. Some visualization modules may require having graphics memory large enough to hold the actual data.

High-end graphics cards have 12 to 16 GB of memory. Optimal performance volumetric visualization at full resolution requires that data fit in graphics memory (some volume rendering modules of Avizo are able to go around this limitation).

Avizo will not benefit from multiple graphics boards for the purpose of visualization on a single monitor. However, some of the image processing algorithms rely on CUDA for computation, and while the computation can run on the single CUDA-enabled graphics board, this computation can also run on a second CUDA-enabled graphics card installed on the system. A multiple graphics board configuration can be useful to drive many screens or in immersive environments.

When comparing graphics boards, there are many different criteria and performance numbers to consider. Some are more important than others, and some are more important for certain kinds of rendering. Thus, it's important to consider your specific visualization requirements. Integrated graphics boards are not recommended for graphics-intensive applications such as Avizo except for basic visualization.

Wikipedia articles on NVIDIA GeForce/Quadro and AMD Radeon/FirePro cards will detail specific performance metrics:

- Memory size: This is very important for volume visualization (both volume rendering and slices) to maximize image quality and performance because volume data is stored in the GPU's texture memory for rendering. It is also important for geometry rendering if the geometry is very large (large number of triangles).
- Memory interface / Bandwidth: This is important for volume rendering because large amounts of texture data need to be moved from the system to the GPU during rendering. The PCI Express 3 buses are the fastest interfaces available today.
- Number of cores (also known as stream processors): This is very important for volume rendering because every high-quality rendering feature you enable requires additional code to be executed on the GPU during rendering.
- Triangles per second: This is very important for geometry rendering (surfaces, meshes).
- Texels per second / Fill rate: This is very important for volume visualization (especially for volume rendering), because a large number of textures will be rendered and pixels will be "filled" multiple times to blend the final image.

	Vendor	Family	Series
Professional graphics boards	NVIDIA	Quadro	Maxwell, Kepler
	AMD	FirePro	W, V

Note: this is the last version of Avizo to support Quadro Fermi series.

All driver bugs are submitted to the vendors. A fix may be expected in a future driver release.

	Vendor	Family	Series
Standard graphics boards	NVIDIA	GeForce	Maxwell, Kepler
	AMD	Radeon	since Evergreen
	Intel	HD Graphics	Ivy Bridge, Haswell

Note: this is the last version of Avizo to support GeForce Fermi series.

Due to vendor support policies, on standard graphics boards we are not able to commit to providing a fix for bugs caused by the driver.

- A **professional graphics boards** will benefit from the professional support offered by the vendors (driver bug fixes).
- Always use a recent driver version for your graphics board.
- With an NVIDIA Quadro board we recommend to use the driver profile "3D App - Visual Simulation". In case of rendering or performance issues you may want to experiment with different "3D App" profiles.
- Turning off the Vertical sync feature improves frame rate.
- Visit http://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units for a complete list of NVIDIA boards and comparisons.
- Visit http://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units for a complete list of AMD boards and comparisons.
- Some visualization modules like *Volume Rendering* may not support Intel graphic cards.
- An issue has been detected on Ivy Bridge GPU (HD graphics, HD2500, HD4000) that prevents the shader cache mechanism from working as expected. In order to be able to run Avizo correctly on these plateforms user must set **OIV_SHADER_DISK_CACHE** envvar to 0. Newer Intel platforms such as HD 4600 are not concerned by this issue.

1.4.1.3 System Memory

System memory is the second most important determinant for Avizo users who need to process large data.

You may need much more memory than the actual size of the data you want to load within Avizo. Some processing may require several times the memory required by the original data set. If you want to load, for instance, a 4 GB data set in memory and apply a non-local means filter to the original data and then compute a distance map, you may need up to 16 or 20 GB of additional memory for the

intermediate results of your processing. Commonly you will need 2 or 3 times the memory footprint of the data being processed for basic operations. For more complex workflows you may need up to 6 or 8 times amount of memory, so 32 GB may be required for a 4 GB dataset. Due to the potentially high memory requirement we highly recommend running on 64-bit computer and operating systems.

Also notice that size of the data on disk may be much smaller than memory needed to load the data as the file format may have compressed the data (for instance, loading a stack of JPEG files).

Avizo's Large Data Access (LDA) technology will enable you to work with data sizes exceeding your system's physical memory. LDA is an excellent way to stretch the performance, but it is not a direct substitute for having more physical memory. The best performance and optimal resolution will be achieved by using Avizo's LDA technology in combination with a large amount of system memory. LDA provides a very convenient way to quickly load and browse your whole dataset. Note that LDA data will not work with most compute modules, which require the full resolution data to be loaded in memory.

Avizo provides another loading option to support for 2D and 3D image processing from disk to disk, without requiring loading the entire data into memory; modules then operate per data slab. This enables processing and quantification of large image data even with limited hardware memory. Since processing of each slab requires loading data and saving results from/to the hard drive, it dramatically increases processing time. Thus, processing data fully loaded in memory is always preferred for best performance.

1.4.1.4 Hard Drives

When working with large files, reading data from the disk can slow down your productivity. A standard hard drive (HDD) (e.g., 7200rpm SATA disk) can only stream data to your application at a sustained rate of about 60 MB/second. That is the theoretical limit; your actual experience is likely to be closer to 40 MB/second. When you want to read a 1 GB file from the disk, you will likely have to wait 25 seconds. For a 10 GB file, the wait is 250 seconds, over 4 minutes. LDA technology will greatly reduce wait time for data visualization, but disk access will still be a limiting factor when you want to read data files at full resolution for data processing. Compared to traditional HDDs, solid state drives (SSD) can improve read and write speeds.

For best performance, the recommended solution is to configure multiple hard drives (3 or more HDD or SSD) in RAID5 mode; note that RAID configurations may require substantially more system administration. For performance only, RAID 0 could be used, but be warned of risk of data loss upon hard-drive failure. If you want performance and data redundancy then RAID 5 is recommended.

Reading data across the network, for example from a file server, will normally be much slower than reading from a local disk. The performance of your network depends on the network technology (100 Mb, 1 Gb, etc.), the amount of other traffic on the network, and number/size of other requests to the file server. Remember, you are (usually) sharing the network and server and will not get the theoretical bandwidth. LDA technology may also facilitate visualization of volume data through the network, but if data loading is a bottleneck for your workflow, we recommend making a local copy of your data.

1.4.1.5 CPU

While Avizo mostly relies on GPU performance for visualization, many modules are computational intensive and their performance will be strongly affected by CPU performance.

More and more modules inside Avizo are multi-threaded and thus can take advantage of multiple CPUs or multiple CPU cores available on your system. This is the case for most of the quantification modules provided with Avizo, a number of modules of the Avizo XLabSuite Extension,and also various computation modules.

Fast CPU clock, number of cores, and memory cache are the three most important factors affecting Avizo performance. While most multi-threaded modules will scale up nicely according to the number of cores, the scaling bottleneck may come from memory access. From experience, up to 8 cores show almost linear scalability while more than 8 cores do not show much gain in performance. A larger memory cache improves performance.

1.4.2 How hardware can help optimizing

Here is a summary of hardware characteristics to consider for optimizing particular tasks.

Visualizing large data (LDA):

- Fast hard drive,
- System memory,
- GPU Memory,
- Memory to GPU/CPU bandwidth.

Basic volume rendering:

- GPU fill rate (texels per second)

Advanced volume rendering (Volume Rendering module):

- Heavy use of pixel shaders,
- GPU clock frequency, number of GPU cores.

Large geometry rendering such as large surfaces from Isosurface or Generate Surface, large point clusters, large numerical simulation meshes,...:

- GPU clock frequency, number of triangles per second.

Image processing and quantification (Avizo):

- Multiple CPU cores (for many modules, including most image processing modules),
- CPU clock frequency.

Anisotropic Diffusion, Non-Local Means Filter (high-performance smoothing and noise reduction image filters) , Avizo XLabSuite Extension (absolute permeability computation):

- GPU speed, number of GPU cores (stream processors), CUDA-compatible (NVIDIA).

Other compute modules, display module data extraction:

- CPU clock frequency,
- Multiple CPU cores (for a number of multi-threaded modules, such as Generate Surface, Register Images, Resample, Arithmetic).

Multi-display systems with Avizo XScreen Extension: tiled displays, immersive displays, virtual reality:

- Multi-GPU systems such as PC clusters, etc.

GPU computing using custom module programmed using Avizo XPand Extension and GPU API:

- GPU clock frequency, number of GPU cores (stream processors),
- Multi-GPU systems such as NVIDIA Tesla,
- CUDA support.

1.4.3   Special considerations

1.4.3.1   Firewall

An internet access is necessary to activate Avizo. Your firewall may prevent the connection to the license server.
For more information, please refer to *licensing troubleshooting (firewall problem)*.

1.4.3.2   Linux

Avizo is only available for Intel64/AMD64 systems.

The official Linux distribution for Avizo is Red Hat Enterprise Linux 6 64-bit. Nevertheless, Avizo is likely to work on some other 64-bit Linux distributions if the required version of system libraries can be found, but technical support of those platforms will be limited. Here is a non-exhaustive list of these 64-bit Linux distributions:

- Red Hat Enterprise Linux 6.x, the official Linux distribution on which Avizo has been fully tested.
- Red Hat Enterprise Linux 7.x, using some of the additional libraries from the $AVIZO_ROOT/lib/compat-LinuxAMD64 directory.
- CentOS 6 and Scientific Linux 6.

- Ubuntu 12.04 and 13.04, using some of the additional libraries from the $AVIZO_ROOT/lib/compat-LinuxAMD64 directory.
- OpenSUSE 12.3, using some of the additional libraries from the $AVIZO_ROOT/lib/compat-LinuxAMD64 directory.

The folder $AVIZO_ROOT/lib/compat-LinuxAMD64 contains the most commonly missing dependencies (`libjpeg.so.62`, `libgfortran.so.1`, `libXm.so.4`, `libstdc++.so.6`). If your Linux distribution misses one of those files or has different versions installed, copy that file into the $AVIZO_ROOT/lib/arch-LinuxAMD64-Optimize directory. You can also install them using the software manager/tool of your linux distribution. For instance, you can install some of them on Ubuntu distribution using the following command:

```
sudo apt-get install libjpeg62 libstdc++6 libmotif4
```

Notes:

- After a standard installation of Linux, hardware acceleration is not necessarily activated, although X-Windows and Avizo may work fine. To enable OpenGL hardware, acceleration specific drivers may have to be installed. This can drastically increase rendering performance. Sometimes it is necessary to disable the stencil buffers (by starting Avizo with the option -no_stencils) to get acceleration.

- On some distributions, some parts of the user interface, the segmentation editor for example, may not display correctly. This is a known Qt issue. You can work around this by disabling the composite option in the extension section of your Xorg.conf configuration file:

```
Section "Extensions"
     Option       "Composite"      "disable"
EndSection
```

- To work properly on Linux systems where SELinux is enabled, Avizo requires the modification of the security context of some Avizo shared object files so they can be relocated in memory. The user (maybe root) that installs Avizo has to run the following command from a shell console in order to set the right security context :

```
chcon -v -t texrel_shlib_t "${AVIZO_ROOT}"/lib/arch-Linux*-*/lib*.so
```

- Even if Avizo should work with any desktop (like KDE), it has been validated only with GNOME.

1.4.3.3 Mac OS X

In order to run CUDA enabled modules, Mac OS X requires the most recent CUDA driver for Mac to be installed. At least version 8.x is required. It can be found at the following URL: http://www.nvidia.com/object/mac-driver-archive.html and must be installed manually. If this driver

is not installed, modules of Avizo using GPU compute capabilities will not be able to run properly. Moreover, compilation of Avizo XPand Extension modules using the CUDA API (even the examples) will fail.

1.4.3.4 XPand

To add custom extensions to Avizo with Avizo XPand Extension on Windows, you will need Microsoft Visual Studio 2013 Update 4. The compiler you need depends on the version of Avizo you have. You can obtain the version information by typing `app uname` into the Avizo console. It is important to install Visual Studio prior to run Avizo in debug mode.

To add custom extensions to Avizo with Avizo XPand Extension on Linux, you will need gcc 4.4.x on RHEL 6. Use the following command to determine the version of the GNU compiler:
`gcc --version`

Currently, Avizo XPand Extension support is not available for **Mac OS X** El Capitan (10.11) nor macOS Sierra (10.12). It will become available once Clang support has been completed.

1.4.3.5 Matlab

To use the *Calculus MATLAB* module that establishes a connection to MATLAB (MathWorks, Inc.), follow these installation instructions:

Windows

If you did not register during installation, enter the following command on the Windows command line: `matlab /regserver`.

In addition, add `MATLAB_INSTALLATION_PATH/bin` and
`MATLAB_INSTALLATION_PATH/bin/win32` or
`MATLAB_INSTALLATION_PATH/bin/win64` in your PATH environment variable to allow Avizo to find MATLAB libraries.

Linux

The   LD_LIBRARY_PATH   environment   variable   should   be   set   to
`MATLAB_INSTALLATION_PATH/bin/glnx86`   on   Linux   32-bit   and
`MATLAB_INSTALLATION_PATH/bin/glnxa64` on Linux 64-bit.

The PATH environment variable should be also set to `MATLAB_INSTALLATION_PATH/bin`.
If you still have trouble starting *Calculus MATLAB* after setting the environment variable, it might be because the GNU Standard C++ Library (`libstdc++`) installed on your platform is older than the one required by MATLAB. You can check MATLAB's embeded `libstdc++` version in `MATLAB_INSTALLATION_PATH/sys/os/glnx86` on Linux 32-bit and `MATLAB_INSTALLATION_PATH/sys/os/glnxa64` on Linux 64-bit.

If needed, add this path to LD_LIBRARY_PATH.

Mac OS X

The LD_LIBRARY_PATH environment variable should be set and the PATH variable must be updated with the MATLAB binaries directory. For example, for MATLAB2017a:

```
export LD_LIBRARY_PATH=/Applications/MATLAB_R2017a.app/bin/maci64:
$LD_LIBRARY_PATH
export PATH=/Applications/MATLAB_R2017a.app/bin:$PATH
```

This will set the environment variables for the current terminal session.
Avizo must be run from this terminal session to enable access to MATLAB librairies.
To set these variables for a global environment, please refer to the Mac developer zone.

Matlab can be used on OSX10.12 only after deactivation of System Integrity Protection:
Deactivation of System Integrity Protection (http://www.imore.com/el-capitan-system-integrity-protection-helps-keep-malware-away):

1. Click the Apple menu
2. Select Restart...
3. Hold down command-R to boot into the Recovery System
4. Click the Utilities menu and select Terminal
5. Type "csrutil disable" and press return
6. Close the Terminal app
7. Click the Apple menu and select Restart...

1.4.3.6 Dell Backup and Recovery Application

We have detected some incompatibility issues with former versions (¡ 1.9) of *Dell Backup and Recovery Application* which can make Avizo crash when opening files with the file dialog. Please update your Dell Backup and Recovery Application to 1.9.2.8 or higher if you encounter this issue.

1.5 Avizo License Manager

1.5.1 Contents

- *About Avizo licensing management*
    - *Node-locked versus floating licenses*
    - *Time-limited versus perpetual licenses*
- *License manager actions*
    - *Online local activation mode*
    - *Offline local activation mode*
    - *FNP license server mode*
    - *Actions independent of activation mode (local or server)*
- *Licensing troubleshooting*
    - *Deactivating Avizo*
    - *Firewall problem*
    - *Licensing events log*
    - *License lookup log*
    - *Return or Upgrade impossible with node-locked licenses*
- *Contacting the license administrator*

1.5.2 About Avizo Licensing Management

The FEI software license you have acquired defines your rights to use Avizo and some of its modules (extensions) with a specific level of functionality, for a certain version, on designated equipment, and for a specific number of simultaneous users. The license may additionally be restricted to a specific period of time or to specific use cases.

Your FEI software license is protected by a license key. Each time the Avizo application is launched, it checks for a valid license key. If a valid license key is found, the application runs. Otherwise, you are requested to get a valid license key.

Avizo has separate license keys for all Avizo editions and extensions.

1.5.2.1 Node-locked Versus Floating Licenses

There are two main types of licenses: *node-locked licenses* and *floating licenses*.

Node-locked licensing allows the software to run on a single, identified computer only.

If you have purchased Avizo*floating licenses*, one or several concurrent users can run Avizo sessions

at the same time from their computers located on your local network. The number of concurrent users is given by the number of Avizo tokens purchased. In this case, you must install FlexNet Publisher license server manager on a local server. Then, on each end user's computer, you will need to specify the location of this local licensing server (see *FNP license server mode*).

1.5.2.2 Time-limited Versus Perpetual Licenses

Licenses can be *time-limited* or *perpetual*. Time-limited licenses are valid until a given date, which is shown in the product splash screen at start up or in the About dialog (accessible via the *Help > About* menu). After this date, Avizo will stop working. This is usually the case for trial licenses, Beta, or for renewable rented licenses (i.e., yearly subscriptions).

1.5.3 License Manager Actions

When you launch Avizo for the first time, a dialog appears, to indicate that no valid license has been found on your computer. You have the choice of activating or evaluating the product (see Figure 1.1). Select the *Activate* option, to open the Activation Wizard. Three modes of configuration (see Figure 1.2) are available:

- *Use online local activation codes*: to activate one or several node-locked licenses on a computer with an Internet connection
- *Use offline local activation codes*: to activate one or several node-locked licenses on a computer with no Internet connection
- *Use FNP license server*: to specify a license server to activate one or several Avizo (editions and/or extensions) licenses
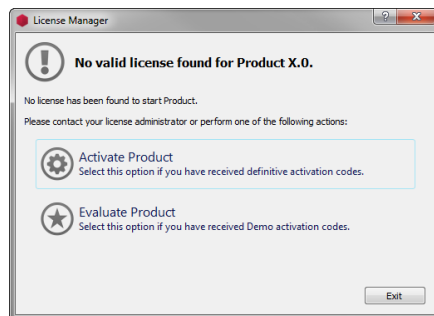
Figure 1.1: License Manager - No Valid License Found

Note: Other actions may be available depending on the configuration mode.
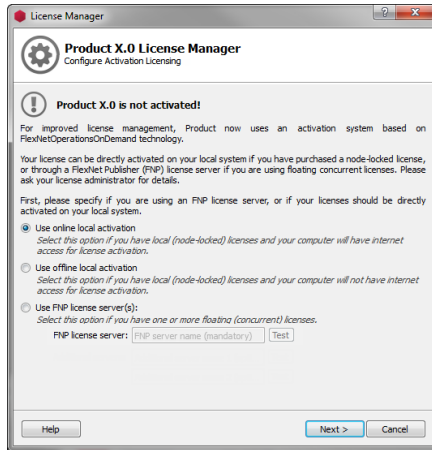
Figure 1.2: License Manager - Configuration Wizard

1.5.3.1 Online Local Activation Mode

When you work in local activation mode, Avizo allows you to manage your licenses. Different actions are available on depending on the status of your licenses.

Note: An Internet connection is required for all following actions. If you do not have any Internet connection, please refer to the 1.5.3.2 documentation.

Activate a Node-Locked License

When you purchase one or more Avizo (editions and/or extensions) licenses, you should receive an email from *fei@flexnetoperations.com* containing a set of activation codes for each purchased product or option.

- Launch Avizo and select *Activate*.
- On the first page of the Activation Wizard, select *Use local activation codes* and click on *Next*.
- In the *Activate Licenses* page, simply copy and paste your activation codes into the provided text field and press *Activate*.

The *Activate* button is enabled when the activation codes into the provided text field respect this format:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
```

HostId is optional.

A connection to the online activation server will start immediately.

When activation is complete, the product is ready to be used and the license manager displays information related to the activation.

An activation code is a string similar to the following:

#License: Avizo (Avizo- 1 year time-limited node-locked application license (including maintenance service) - Multiple platforms support) - Expiration date: 12/13/2014:

BL5F-7773-A9F1-F79B Avizo _9.4
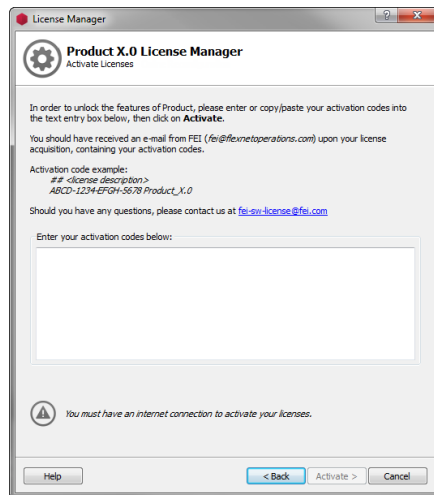
Figure 1.3: License Manager - Node-Locked Activation Wizard

Add New Node-Locked License after Activation

Once Avizo is activated, you can add other node-locked license to activate new extensions or editions. To do so:

- Select *Help > License Manager*
- Click on the *Activate Additional Extensions or Editions* button (see Figure 1.4)
- The license manager wizard is opened and displays the page *Activate Licenses* (see Figure 1.3). Simply copy and paste your activation codes into the provided text field and press *Activate*.

The *Activate* button is enabled when the activation codes into the provided text field respect this format:

    <EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
    HostId is optional.
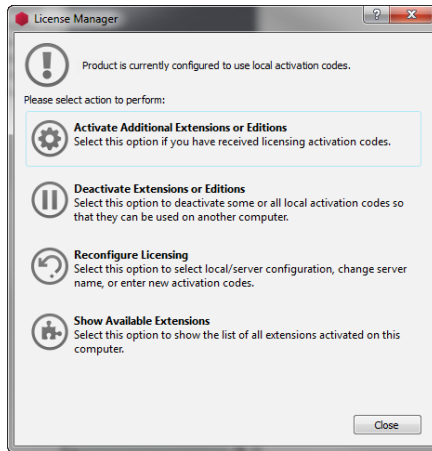
Figure 1.4: License Manager Dialog

Transferring all Node-Locked Licenses to a New Computer

After activating a node-locked license on a computer, you may wish to move it to a new one. To do so, you first need to de-activate the first license by returning the associated activation code and then activate it again on the new computer.

To do so:

- Open the *Help > License Manager* (see Figure 1.4)
- Click on the *Deactivate Extensions or Editions* button. A dialog will open (see Figure 1.16), allowing you to select the licenses you want to deactivate.
- When you click on the *Deactivate* button (see Figure 1.5, the selected licenses are returned to the server and can be activated on any other system.

Upgrading to a New Avizo Version

When a new version of Avizo is released, a dialog is displayed if the activation process was used for a previous version after installing and launching the new Avizo version. It offers several choices, including an **Upgrade Local Licenses** option (see Figure 1.6).

If the licensed modules are under maintenance, they will be immediately available for use after the upgrade process.
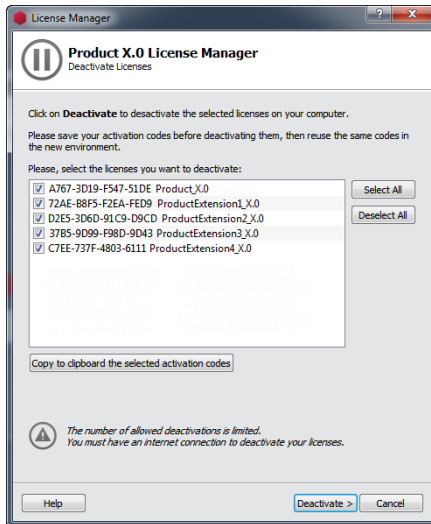
Figure 1.5: License Manager - Deactivation Dialog
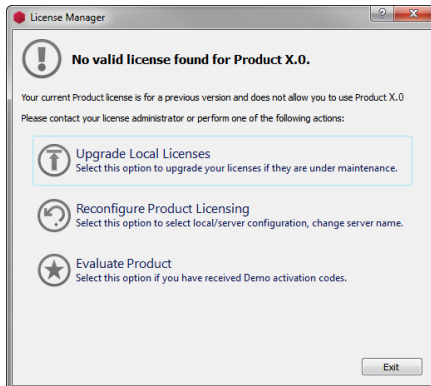
Figure 1.6: License Manager - Upgrade Dialog

Note: The previous version can still be used on the same computer even after upgrading to the new version.

Reactivating a License

When you use a renewable license (i.e., through a subscription agreement), Avizo will stop working when the license expiration date is reached.
When you launch a new session of Avizo, an error dialog is displayed with several choices, including a *Reactivate* option.
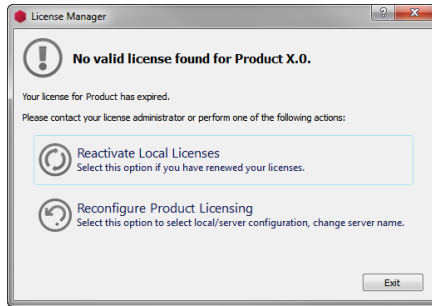
Figure 1.7: License Manager - Reactivation Dialog

As soon as you have renewed your subscription, you can re-activate your node-locked license to allow Avizo to run again for a new period of time.

Activate Demo Avizo licenses

The Demo licenses are node-locked and time-limited.

- Launch Avizo and select the *Evaluate* option.
- On the first page of the Activation Wizard, select *Use local activation codes* and click on *Next*.
- On the *Activate Demo Licenses* page, simply copy and paste your activation codes into the provided text field and press *Activate*.

The *Activate* button is enabled when the format of activation code into the provided text field respect this format:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
```
HostId is optional.

A connection to the online activation server will start immediately. When activation is complete, the product is ready to be used.

Activate Beta Avizo Licenses
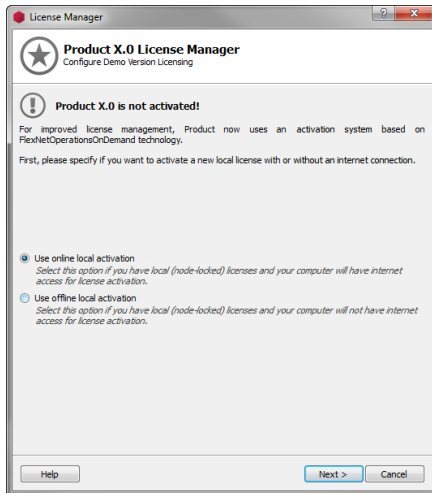
Figure 1.8: License Manager - Demo Activation Wizard - Configure Demo Version Licensing
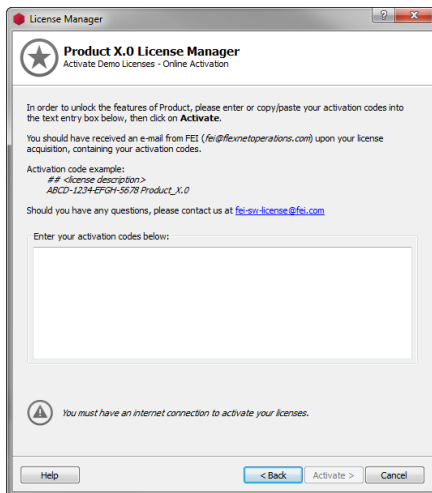
Figure 1.9: License Manager - Demo Activation Wizard - Activate Demo Licenses

The Beta licenses are only node-locked and time-limited.

When you launch a Beta version for the first time, or if your Beta licenses are expired, the Activation Wizard is opened at the *Configure Beta Version Licensing* page.

- Select *Use local activation codes* and click on *Next*
- On the *Activate Beta Licenses* page, simply copy and paste your activation codes into the provided text field and press *Activate*

The *Activate* button is enabled when the format of activation code into the provided text field respect this format:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
```
HostId is optional.

A connection to the online activation server will start immediately. When finished, the product is ready to be used.
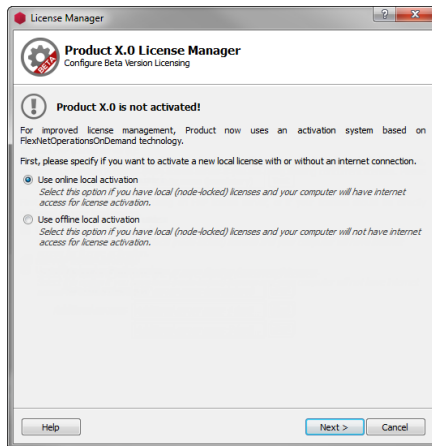
Figure 1.10: License Manager - Beta Activation Wizard - Configure Beta Version Licensing

1.5.3.2 Offline Local Activation Mode

In offline activation mode, you will be able to perform the same actions described in 1.5.3.1, but without any Internet connection.

Activate a Node-Locked License

When you purchase one or more Avizo licenses (i.e., editions and/or extensions), you should receive an email from *fei@flexnetoperations.com* containing a set of activation codes for each purchased product or option.
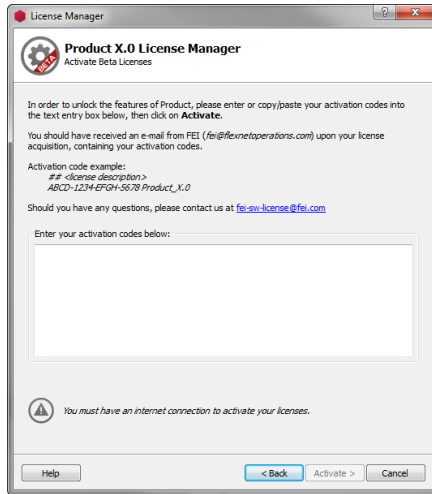
Figure 1.11: License Manager - Beta Activation Wizard - Activate Beta Licenses

- Launch Avizo and select **Activate**.
- On the first page of the Activation Wizard, select **Use offline activation codes** and click **Next**.
- In the *Activate Licenses* page, copy and paste your activation codes into the provided text field and specify a path to an XML file (i.e., a file with extension *.xml*) that will contain your offline activation request.
- Then, click **Next** (see Figure 1.12).

The **Next** button is enabled when the path of XML request file is specified. The format of activation code uses the following template:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
    HostId is optional.
```

The next page (see Figure 1.13) explains the remaining steps to perform before completing the offline activation request:

1. Transfer your XML request file to a computer with an Internet connection.
2. Open vsg3d.flexnetoperations.com/control/vsgs/offlineActivation in a web browser.
3. Specify the path to your XML request file in the provided file field.
4. Click **Process**. An *activation.xml* XML response file will be downloaded.
5. Transfer *activation.xml* to the computer without Internet access.
6. Click **Next** to import the *activation.xml* file and resume your activation request (see Figure 1.14).

When activation is complete, the product is ready to be used and the license manager displays
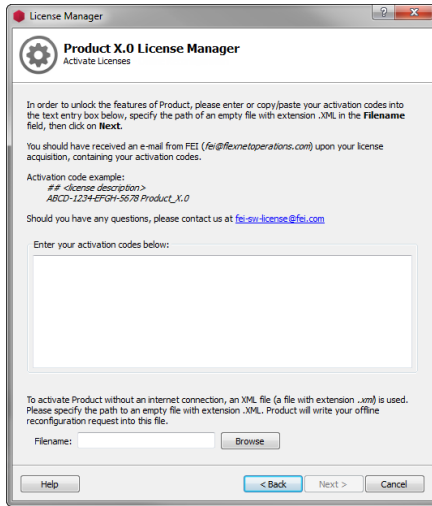
Figure 1.12: License Manager - Offline Activation Wizard
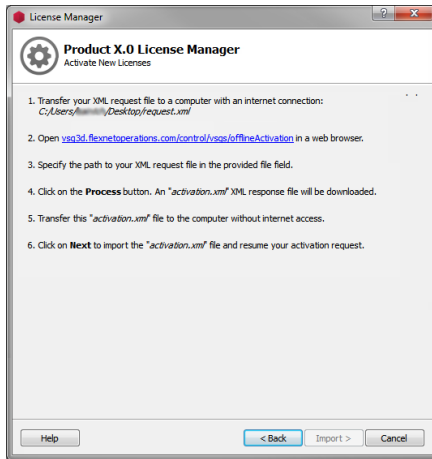
Figure 1.13: License Manager - Offline Instructions

information related to the activation.

Add New Node-Locked License After Activation
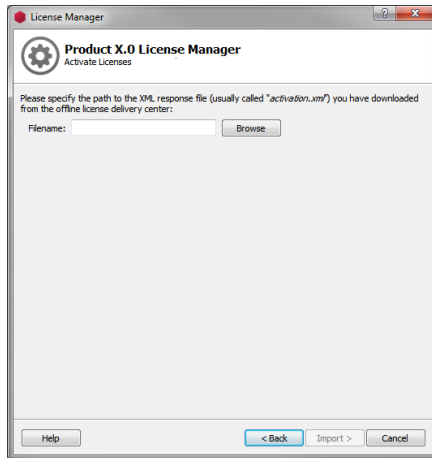
Figure 1.14: License Manager - Offline Import

Once Avizo is activated, you can add other node-locked license to activate new extensions or editions. To do so:

- Select *Help* > *License Manager* and click **Activate Additional Extensions or Editions** (see Figure 1.15).
- The license manager wizard is opened and displays the page *Activate Licenses* (see Figure 1.12). Copy and paste your activation codes into the provided text field and specify a path to an XML file that will contain your offline activation request.
- Then click **Next** and follow the displayed instructions (listed also in the 1.5.3.2 section).

The **Next** button is enabled when the path of XML request file is specified. The format of activation code uses the following template:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
HostId is optional.
```

Transferring all Node-Locked Licenses to a New Computer

After activating a node-locked license on a computer, you may wish to move it to a new one. To do so, you first need to deactivate the first license by returning the associated activation code and then activate it again on the new computer.

To do so:
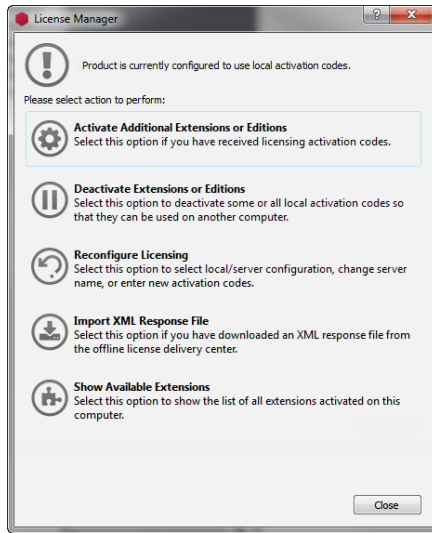
Figure 1.15: License Manager Dialog Offline

- Open the *Help > License Manager* (see Figure 1.15)
- Click *Deactivate Extensions or Editions*.
- A dialog will open (see Figure 1.16): select the licenses you want to deactivate and the path to the XML file that will contain your offline deactivation request.
- After clicking **Next**, the dialog displaying the offline instructions (listed also in the 1.5.3.2 section) will be displayed.
- Once the import of the deactivation request has been performed, the selected licenses are returned to the server and can be activated on any other system.

Upgrading to a New Avizo Version

When a new version of Avizo is released, a dialog is displayed if the activation process was used for a previous version after installing and launching the new Avizo version. It offers several choices, including an **Upgrade Local Licenses** option.

After clicking **Upgrade Local Licenses**, a dialog will open (see Figure 1.18), allowing you to select the licenses you want to upgrade and the path to the XML file that will contain your offline upgrade request. Click **Next** to display the offline instructions dialog (also listed in the 1.5.3.2 section). Once the import of the upgrade request has been performed (and if the licensed modules are under maintenance), the selected licenses will be upgraded and the new version of Avizo can be used immediately.

Note: The previous version can still be used on the same computer even after upgrading to the new
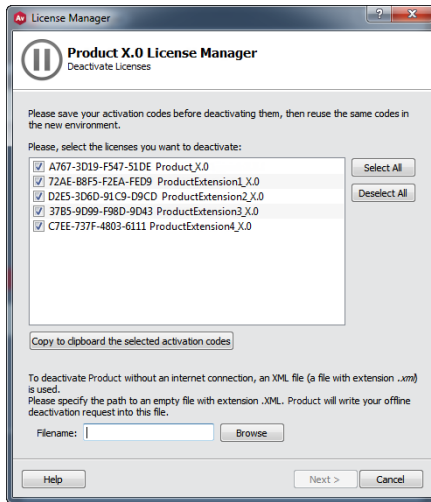
Figure 1.16: License Manager - Deactivation Dialog Offline
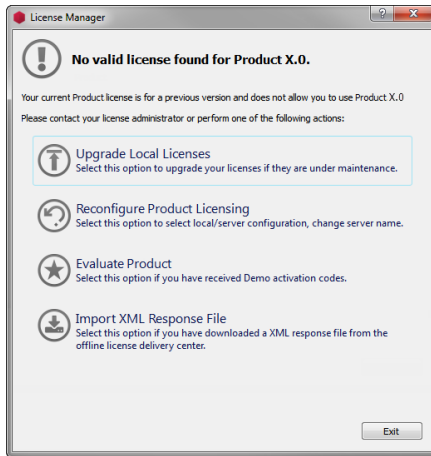
Figure 1.17: License Manager - Upgrade Proposal Dialog Offline

version.

Reactivating a License

When you use a renewable license (i.e., through a subscription agreement), Avizo will stop working
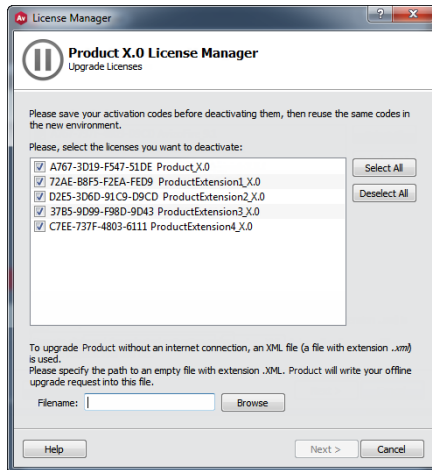
Figure 1.18: License Manager - Upgrade Licenses Dialog Offline

when the license expiration date is reached. When you launch a new session of Avizo, an error dialog will appear with several choices including a **Reactivate Local Licenses** option.
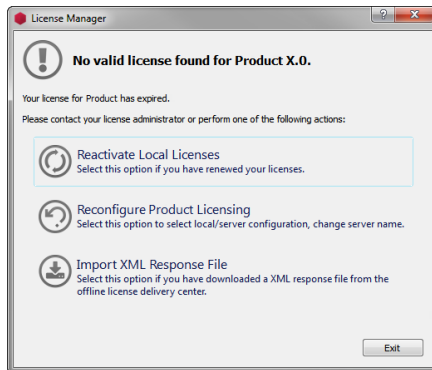
Figure 1.19: License Manager - Reactivation Proposal Dialog Offline

Click **Reactivate Local Licenses** to open the License Manager dialog box (see Figure 1.19) and select the licenses you want to activate plus the path to the XML file that will contain your offline reactivation request (see Figure 1.20). Click **Next** to display the offline instructions dialog (also listed in the 1.5.3.2 section). Once the import of the reactivation request has been performed (and if you have renewed your subscription), the selected licenses will be reactivated and the new version of Avizo can be used immediately for a new period of time.
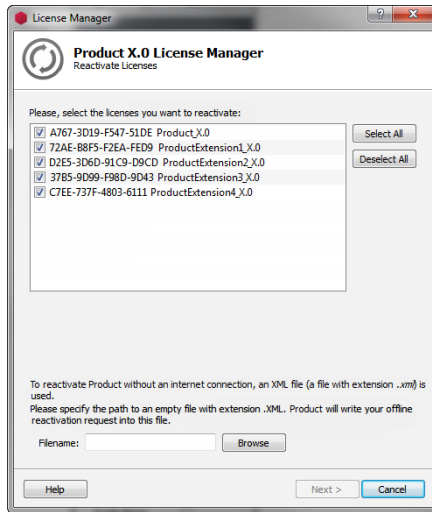
Figure 1.20: License Manager - Reactivate Licenses Dialog Offline

Activate Demo Avizo Licenses

The *Demo Licenses* are node-locked and time-limited.

- Launch Avizo and select the **Evaluate** option.
- On the first page of the Activation Wizard, select **Use offline activation codes** and click **Next** (see Figure 1.21).
- On the *Activate Demo Licenses* page, copy and paste your activation codes into the provided text field and specify a path to an XML file that will contain your offline activation request (see Figure 1.5.3.2).

The **Next** button is enabled when the path of XML request file is specified. The format of activation code uses the following template:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
HostId is optional.
```

- Click **Next** to display the offline instructions dialog (also listed in the 1.5.3.2 section).

Once the import of the activation request has been performed, the product is ready to be used.
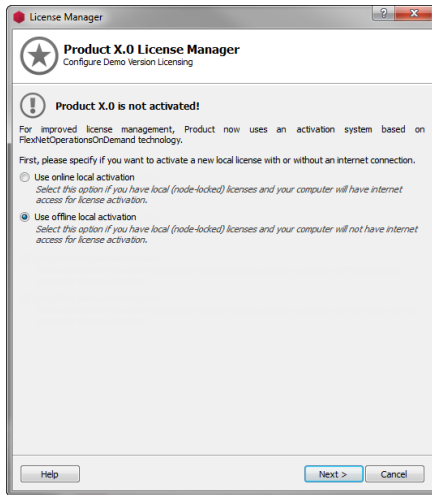
Figure 1.21: License Manager - Demo Activation Wizard - Configure Demo Version Licensing Offline
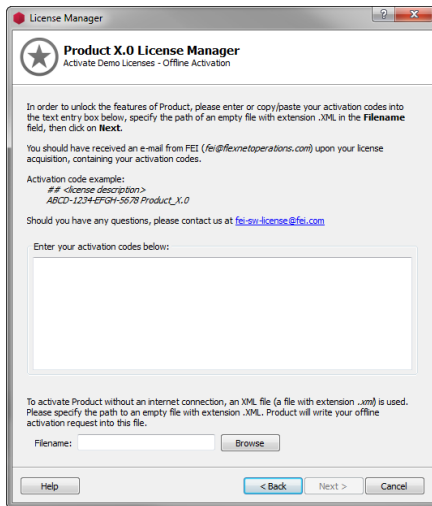
Figure 1.22: License Manager - Demo Activation Wizard - Activate Demo Licenses Offline

Activate Beta Avizo Licenses

The Beta licenses are only node-locked and time-limited.

When you launch a Beta version for the first time or if your Beta licenses are expired, the Activation Wizard will open on the *Configure Beta Version Licensing* page.

- You must select **Use offline activation codes** and click **Next** (see Figure 1.23).
- On the *Activate Beta Licenses* page, simply copy and paste your activation codes into the provided text field and specify a path to an XML file that will contain your offline activation request (see Figure 1.5.3.2)

The **Next** button is enabled when the path of XML request file is specified. The format of activation code uses the following template:

```
<EntitlementId> [<HostId>] <ProductName>_<ProductVersion>.
HostId is optional.
```

- Click **Next** to display the offline instructions dialog (also listed in the 1.5.3.2 section).

Once the import of the activation request has been performed, the product is ready to be used.
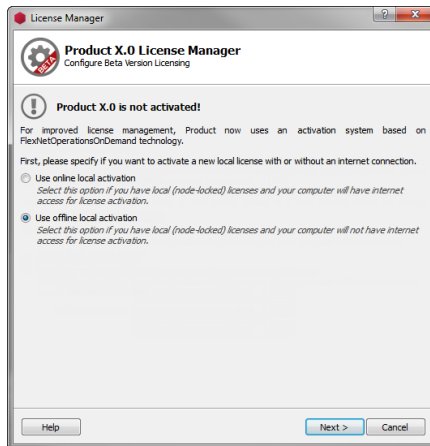
Figure 1.23: License Manager - Beta Activation Wizard - Configure Beta Version Licensing Offline

1.5.3.3   FNP License Server Mode

Avizo floating or concurrent licenses are handled by FlexNet Publisher tool. Do the following to manage Avizo floating licenses:
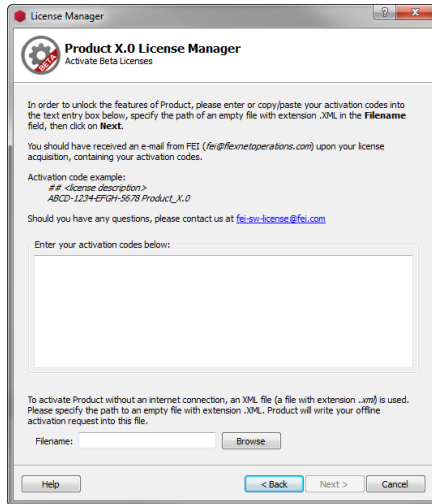
Figure 1.24: License Manager - Beta Activation Wizard - Activate Beta Licenses Offline

1. *Install* FlexNet Publisher license server manager on a local server of your LAN
2. *Activate* your Avizo licenses on this license server
3. *Configure* each Avizo instance on local computers to use this license server

FNP License Server Manager Installation and Management

Refer to the following website to install a FlexNet license server and manage floating licenses, including activation, upgrade, reactivation, and return:
www.fei-software-center.com/flexnet-server-doc/

Configure Avizo to Use Licenses from FNP License Server

At Avizo startup, select the *Activate* option and then *Use FNP license server* on the first page of the Activation Wizard.
You will need to specify the name of the FNP license server. Optionally, you can specify a port number with a separator ":" (<ServerName>:<PortNumber>). Using a specific port number depends on the way the server is configured. In such a case, please contact your server license administrator for more information about the FNP license server configuration and the potential need to indicate a specific port number.

A connection to the FNP license server will start immediately. When done, the product is ready to be used.
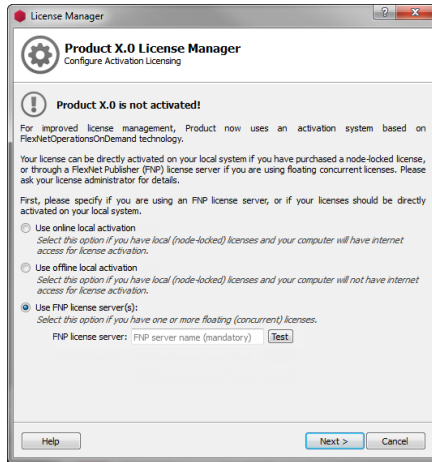
Figure 1.25: License Manager - FNP License Server Configuration

Note: It is possible to test the connection to the specified server using the *Test* button.

1.5.3.4   Actions Independent of Activation Mode (Local, Server or Offline)

Reconfigure

You can change your configuration mode at any time, but this action requires that you restart the application.
To do so (in *local activation mode*), select *Help > License Manager* and click on the *Reconfigure Licensing* button.

Show Available Extensions

When Avizo is running, you can see the activated licenses on your computer.
Select *Help > Show Available Extensions*.

Note: One activation code can activate several licenses.
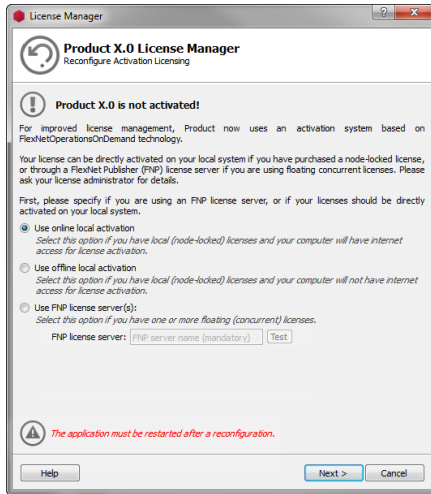
Figure 1.26: License Manager - Reconfiguration Wizard
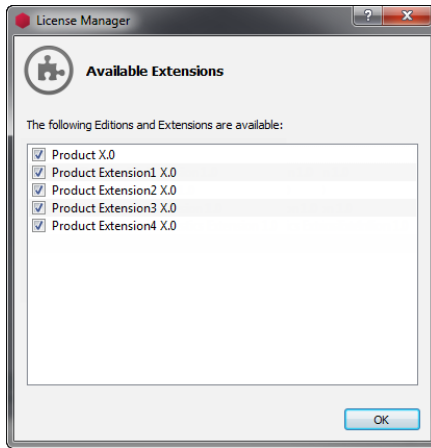
Figure 1.27: License Manager - Available Extensions

1.5.4   Licensing Troubleshooting

1.5.4.1   Deactivating Avizo

When using Avizo node-locked activation codes, please deactivate them before any of the following operations:

- Uninstalling Avizo
- Reinstalling or replacing the operating system
- Replacing the hard drive
- Disposing of the system

Otherwise, you will not be able to reactivate them on another system since they will still be active on the original system, and it will be difficult or impossible for you to deactivate them for a transfer.
To deactivate Avizo, please read the following instructions according to your licensing mode: *online* or *offline*.

1.5.4.2 Firewall Problem

If you have an Internet connection and encounter the error "Unable to connect to server https" when trying to activate a license, it originates from your firewall configuration.
You need to configure your firewall to allow the connection to the online activation server:

- Server: vsg3d.flexnetoperations.com
- IP: 64.14.29.85
- Port: 443 (https)

1.5.4.3 Licensing Events Log

If an error occurs during Avizo activation, you can find more details in the licensing events log.
However, most likely you will need to send it to the *technical support team* for analysis (*Help > Online Support menu*).

The log file is saved in the user application data directory:

- Windows path: *C:\Users\<your_login_name>\AppData\Roaming\FEI \licensingAvizo.log*
- Linux and Mac OS X path: */home/<your_login_name>/.config/FEI/licensingAvizo.log*

This log file details all activation actions and licensing information only when an error occurs.

1.5.4.4 License Lookup Log

If the *VSG_LICENSE_DEBUG* environment variable is set to a file name, licensing debug output is written to the specified file when Avizo is launched. You can open and read this debug file yourself. However, most likely you will need to send it to the *technical support team* for analysis (*Help > Online Support menu*).

Below are instructions for setting this environment variable on Windows and on Linux and Mac OS X systems.

Note: For using a license debug filename as indicated below, you need sufficient privileges to write into the Avizo installation directory. Otherwise, set *VSG_LICENSE_DEBUG* to a path where you can write files. For example, */home/<your_login_name>/debug.txt*, or *C:\Temp\debug.txt*.

You can set the environment variable via the Control Panel for Windows or you can set it in a command prompt on all platforms.

Windows - Control Panel

1. Go to the Control Panel via the Windows Start menu.
2. In the Control Panel, select the System application.
3. Click the Advanced tab.
4. Click the Environment variables button.
5. In either the User or System variables, click *New*.
6. For the Variable name, enter *VSG_LICENSE_DEBUG*.
7. For the Variable value, enter *debug.txt* (without the quotes).
8. Click *OK* to close the dialog boxes
9. Run Avizo from the Start menu. When the License Manager dialog is displayed, dismiss it.
10. Then look for a file named *debug.txt* in the top level of the Avizo installation directory. Send the file to *tech support* for analysis.

Windows - Command Prompt

1. On Windows you can get to a command prompt via the Windows Start menu as follows:
   Start > Programs > Accessories > Command Prompt
2. In the command prompt window, type:
   `set VSG_LICENSE_DEBUG=debug.txt`
3. Change the current directory to where your Avizo executable is. For example:
   `cd C:\Program Files\Avizo9\bin\arch-Win64VC12-Optimize`
4. Run Avizo as follows:
   `Avizo.exe`
5. When the License Manager dialog displays, dismiss it.
6. Look for the file *debug.txt* in the current directory. Send the file to *tech support* for analysis.

Linux and Mac OS X - Terminal

1. On Linux and Mac OS X systems, the exact command to use for setting environment variables depends on the kind of shell you are using. Here are examples for a few commonly used shells. Bash shell:
   ```
   export VSG_LICENSE_DEBUG=debug.txt
   ```
   C shell:
   ```
   setenv VSG_LICENSE_DEBUG debug.txt
   ```
2. In the window in which you set the environment variable, change to the Avizo installation directory. For example:
   ```
   cd /opt/Avizo9
   ```
3. Run Avizo as follows:
   ```
   bin/start
   ```

4. When the License Manager dialog is displayed, dismiss it.
5. Look for the file debug.txt in the current directory. Send the file to *tech support* for analysis.

1.5.4.5 Return or Upgrade Impossible with Node-Locked Licenses

The License Manager uses the Unique Machine Number (UMN) to validate the identity of the machine that initiated the request. The Unique Machine Number value is retrieved from your system hardware information, or from hypervisor-controlled information for virtual machines. If your system hardware is modified since the original activation, the UMN of your machine can be changed and the License Manager will consider your machine to be a different machine. Consequently, the following error message may appear:

>"Trying to return a fulfillment issued to a different machine."

If this error message occurs, contact Technical Support for assistance. Section 1.7 (Contact and Support).

1.5.5 Contacting the License Administrator

You can contact the license administrator using this address: fei-sw-license@fei.com
You can find more information here: Section 1.7 (Contact and Support).

1.6 First steps in Avizo

For a quick introduction to Avizo, you can visit our youtube channel, and watch introductory videos such as *Avizo Getting Started*, *Introduction to Avizo*, and *Avizo Getting Started* in addition to the tutorials below.

The step-by-step tutorials in this user's guide are largely independent of each other, so after reading the *Getting Started* section it is possible to skip around and just follow those tutorials which interest

you. If you go through all the tutorials you will get a good survey of Avizo's features. A minimum set of recommended tutorials are shown in bold in the list below.

In all tutorials the steps to be performed by the user are marked by a dot. If you only want to get a quick idea how to work with Avizo you may skip the explanations between successive steps and just follow the instructions. But in order to get a deeper understanding, you should refer to the text.

- **Avizo All Editions**

    - *Getting started* - **the basics of Avizo, useful for all editions**
    - *Reading images* - **how to read images**
    - *Visualizing 3D images* - **slices, isosurfaces, volume rendering**
    - *Image segmentation* - **segmentation of 3D image data**
    - *Surface reconstruction* - **surface reconstruction from 3D images**
    - *Grid generation* - creating a tetrahedral grid from a triangular surface
    - *Vector fields* - streamlines and other techniques
    - *The Animation Director* - creating animations using the Animation Director
    - *Large data* - how to work with out-of-core data files (LDA)
    - *Creating movie files* - using the Movie Maker module
    - *Using MATLAB* - using the Calculus MATLAB module
    - *Python Tutorial* - Using Tools from the Python Eco-System within Avizo
    - *Skeletonization* - how to analyse the network or tree-like structures in 3D image data
    - *Molecular Visualization* - how to visualize and analyse molecular data

    Note: Section 2.6 contains some general hints on how to import data sets into Avizo.

- **Avizo**

    - *Getting Started with Image Processing and Analysis* - **the basics for 3D image processing**
    - *Distribution of distance between 2 phases* - measuring a catalyst
    - *Separation, Measures and Reconstruction* - advanced pore reconstruction
    - *Granulometry* - distribution of pore diameters
    - *Pore Thickness Computation* - average thickness of pores
    - *Advanced segmentation* - using watershed with segmentation editor or wizard
    - *Image filtering* - choosing, tuning, and combining image filters
    - *Registration, Alignment and Data Fusion* - registration of surfaces and images
    - *More about label measures* - selecting the measures to perform on labels and creating your own measures
    - *Advanced surface and grid generation* - meshing and numerical simulation

First steps in Avizo **46**

- **Avizo XWind Extension**

  - *Getting Started* - the basics to visualize and analyze numerical simulations data
  - *Visualizing a CFD Model* - model information and display on a YF-17 Cobra aircraft
  - *Scalar Field Visualization* - scalar field display on a YF-17 Cobra aircraft
  - *Vector Field Visualization* - vector field display on a wing airplane
  - *Statistical and Arithmetic* - statistical and arithmetic computations on a YF-17 Cobra aircraft
  - *Vorticity Study* - vorticity identification on a wing airplane
  - *Measurement Tools* - measurements on a YF-17 Cobra aircraft
  - *Meshing Workroom Tutorial* - meshing workroom for numerical simulation

- **Avizo XGreen Extension**

  - *Getting Started***- the basics to visualize earth-mapped data**
  - *2D Scalar Data* - visualization of 2D scalar data on climatology data sets
  - *2D Vector Data* - visualization of 2D vector data on climatology data sets
  - *3D Scalar Data* - visualization of 3D scalar data on climatology data sets

- **Avizo XFiber Extension**

  - *Filament Editor* - filament tracing for neurons and vessels images
  - *Fibers detection* - detection workflow for the example of steel fibers in reinforced concrete
  - *Advanced statistics computation* - fibrous materials quantification and analysis

- **Avizo XMetrology Extension**

  - *Metrology* - test plan creation, geometry fitting, view alignment, and part dimensioning

- **Avizo XRecipe Extension**

  - *Recipes* - recipes creation, customizable workflow automation

- **Avizo XReporting Extension**

  - *Reporting* - report generation, export of analysis to HTML and PDF

- **Avizo XPoreNetworkModeling Extension**

  - *Pore Space Analysis* - analyze and characterize a pore space

- **Avizo XBioFormats Extension**

  - *Bio-Formats* - Bio-Formats Image Data Import Extension

- **Avizo XDigitalVolumeCorrelation Extension**

  - *Digital Volume Correlation Analysis* - perform a DVC analysis and visualize the corresponding displacement and strain fields

Avizo technical support can be contacted using the *Online Support* item in Avizo Help menu or using the contact information in Section 1.7 (Contact and Support). FEI technical support is dedicated to customers having purchased or renewed a maintenance contract. To speed up the management of your questions, please don't forget to provide the license number or license file of your FEI product, the platform where you use it, and depending on your issue, a log file (VSG_LICENSE_DEBUG export, Help/SystemInformation for Avizo), images and a small sample project.

If you are evaluating Avizo, do not hesitate to contact your support application engineer or sales representative to assist you in learning how Avizo can help you.

FEI also provides trainings and consulting services.

1.7   Contact and Support

For purchasing an Avizo license and for support, visit our web sites or contact one of the following addresses.

Online:

For technical support:
fei-sw-support@fei.com

For requesting license keys, please contact:
fei-sw-license@fei.com

For contacting your sales representative:
http://www.fei.com/contact-fei-sales/

FEI web site:
http://www.avizo3d.com

Phone numbers and addresses:

License key requests:

Phone: +33 556 13 37 78
Fax: +33 556 13 02 10

Email: fei-sw-license@fei.com

FEI SAS
3, Impasse Rudolf Diesel, Bat A - BP 50227
F-33708 Merignac Cedex
France
Phone: +33 (0) 556 13 37 77
Fax: +33 (0) 556 13 02 10
Email: fei-sw-info@fei.com

Hotline requests
Phone: +33 556 13 37 71
Fax: +33 556 13 02 10
Email: fei-sw-support@fei.com

Chapter 2

Getting Started

In this section, you will learn how to

1. start the program.
2. load a demo data set into the system
3. invoke editors for editing the data
4. connect visualization modules to the data
5. interact with the 3D viewer.

The following text has the form of a short step-by-step tutorial. Each step builds on the steps described before. We recommend that you read the text online and carry out the instructions directly on the computer. Instructions are indicated by a dot so you can execute them quickly without reading the explanations between the instructions.

2.1   Start the program

Two editions of Avizo can be installed:

- Run *Avizo Lite Edition* from the start menu for the Avizo Lite Edition for Scientific Visualization
- Run *Avizo* from the start menu for the Avizo for Materials Science

- On a Windows system, select the Avizo or AvizoLite icon from the start menu.
- On a Unix system, start Avizo or AvizoLite by launching the script *bin/start*.
- On a Mac OS X system, select the Avizo or AvizoLite icon from the Application menu.
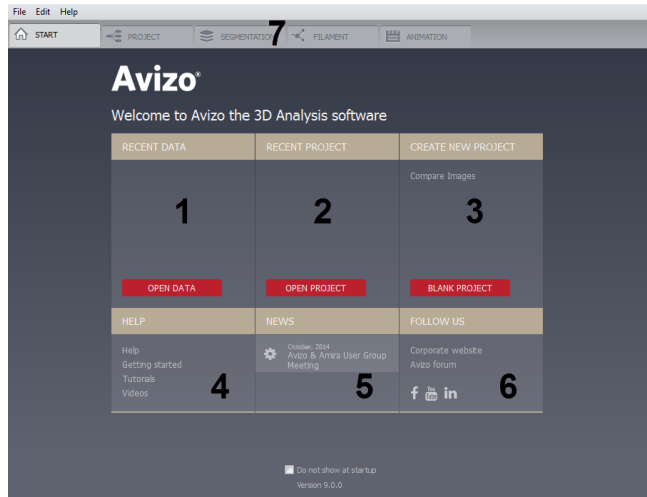
Figure 2.1: The Avizo*Start* page

When Avizo is running, a window like the one shown in Figure 2.1 appears on the screen. This *Start* page is divided into several major regions:

1. **Recent Data**: allows opening data using `Open Data` button and displays all data recently opened (at first start, this section is empty).
2. **Recent Project**: allows opening project using `Open Project` button and displays all project recently opened (at first start, this section is empty).
3. **Create New Project**: allows creating a new project using `Blank Project` button.
4. **Help**: contains some links to help of Avizo.
5. **News**: contains some news on Avizo.
6. **Follow us**: contains some links to follow Avizo community.
7. **Workrooms toolbar**: provides quick access to some important workrooms. See *specific documentation of workroom toolbar.*

When switching to *Project* workroom, a window like the one shown in Figure 2.2 appears on the screen. The user interface is divided into three major regions.

1. The Project View will contain small icons representing data objects and modules.
2. The Properties Area displays interface elements (*ports*) associated with Avizo objects.
3. The 3D viewer window displays visualization results, e.g., slices or isosurfaces.
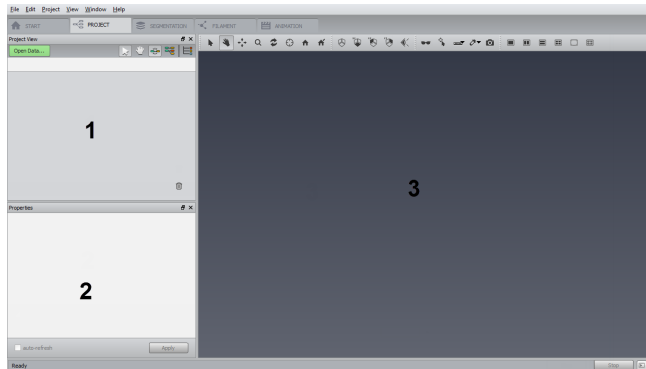
Figure 2.2: The *Project* workroom

You can also activate the help browser by pressing F1, selecting *User's Guide* from the *Help* menu, or by typing help in the console window.

2.2 Loading Data

Usually, the first thing you will do after starting Avizo is to load a data set. Let's see how this can be done:

- Choose *Open Data...* from the *File* menu.

After selecting this menu item, the file dialog appears (see Figure 2.3). By default, the dialog displays the contents of the directory defined in the environment variable AVIZO_DATADIR

If no such variable exists, the contents of Avizo's demo data directory are displayed. You can quickly switch to other directories, e.g., to the current working directory, using the directory list located in the upper part of the dialog window.

At the top of the Project View is an Open Data button which is a shortcut to the File/Open Data dialog. You may use it for opening data files in the tutorials that follow. However, the tutorials will instruct you to use the File/Open Data command.

Avizo is able to determine many file formats automatically, either by analyzing the file header or the file name suffix. The format of a particular file will be printed in the file dialog right beside the file name.

Now, we would like to load a scalar field from one of the demo data directories contained in the Avizo distribution.

- Change to the directory data/tutorials, select the file chocolate-bar.am and press
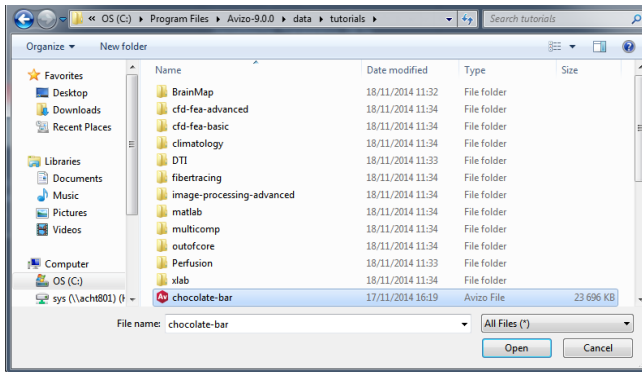
Figure 2.3: Data sets can be loaded into Avizo using the file browser. In most cases, the file format can be determined automatically. This is done by either analyzing the file header or the file name suffix.

      OK.

The data will be loaded into the system. Depending on its size this may take a few seconds. The file is stored in Avizo's native *Avizo* format. The file `chocolate-bar.am` contains 3D image data of a chocolate bar. The data represents a series of parallel 2D image slices across a 3D volume. Once it has been loaded, the data set appears as a green icon in the Project View. In the following, we call this data set "chocolate-bar data set".

- Click on the green data icon with the left mouse button to select it.

This causes some information about the data record to be displayed in the Properties Area (Figure 2.4). In our case, we can read off the dimensions of the data set, the primitive data type, the coordinate type, as well as the voxel size. To deselect the icon, click on an empty area in the Project View window. You may also pick the icon with the left mouse button and drag it around in the Project View.

Clicking on an object typically causes additional buttons to be displayed in the button area at the top of the Project View. These buttons are convenience buttons allowing easy one-click access to the modules most frequently used by the selected object. The tutorials, however, will have you access modules via the menu interface to help familiarize you with the organization of modules within Avizo.

2.3   Invoking Editors

After selecting an object, in addition to the textual information, some buttons appear in the Properties Area, to the far right of the data object's name. These buttons represent *editors* which can be used to interactively manipulate the data object in some way. For example, all data objects provide a *data parameter editor*. This editor can be used to edit arbitrary attributes associated with the data set, e.g.,
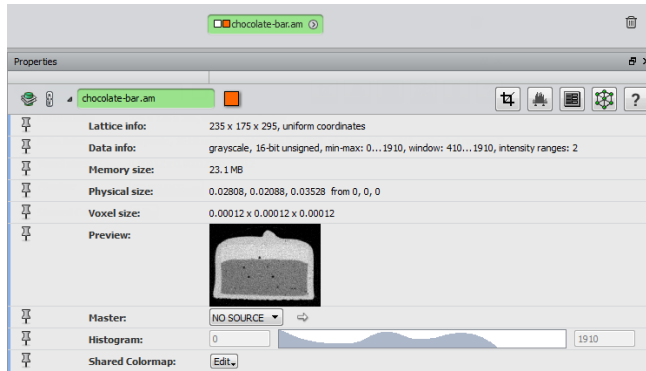
Figure 2.4: Data objects are represented by green icons in the Project View. Once an icon has been selected information about the data set such as its size or its coordinate type is displayed in the Properties Area.

filename, original size, or bounding box. Another example is the *Transform Editor* which can be used to translate or rotate the data in world coordinates. However, at this point we don't want to go into details. We just want to learn how to create and delete an editor:

- Invoke one of the editors by clicking on an editor icon.
- Close the editor by clicking again on the editor icon.

Further information about particular editors is provided in the user's reference manual.

2.4 Visualizing Data

Data objects like the chocolate bar data can be visualized by attaching *display modules* to them. Each icon in the Project View provides a popup menu from which matching modules, i.e., modules that can operate on this specific kind of data, can be selected. To activate the popup menu

- Right-click on the green data icon. Choose the entry called *Bounding Box*.

After you release the mouse button, a new *Bounding Box* module is created and is automatically connected to the data object. The *Bounding Box* object is represented by a yellow icon in the Project View and the connection is indicated by a gray line connecting the icons. At the same time, the graphics output generated by the *BoundingBox* module becomes visible in the 3D viewer. Since the output is not very interesting, in this case we will connect a second display module to the data set:
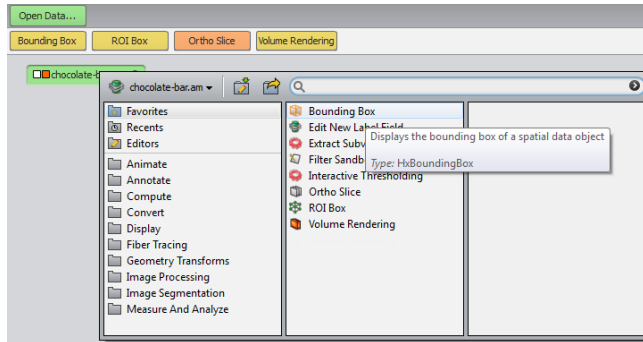
Figure 2.5: In order to attach a module to a data set, click on the green icon using the right mouse button. A popup menu appears containing all modules which can be used to process this particular type of data.

- Choose the entry called *Ortho Slice* from the popup menu of the chocolate bar data set.

Now a 2D slice through the chocolate bar is shown in the viewer window. Initially, a slice oriented perpendicular to the z-direction and centered inside the image volume is displayed. Slices are numbered 0, 1, 2, and so on. The slice number as well as the orientation are parameters of the *Ortho Slice* module. In order to change these parameters, you must select the module. As for the green data icon, this is done by clicking on the *Ortho Slice* icon with the left mouse button. By the way, in contrast to the *Bounding Box*, the *Ortho Slice* icon is orange, indicating that this module can be used for clipping.

- Select the *Ortho Slice* module.

Now you should see various buttons and sliders in the Properties Area, ordered in rows. Each row represents a *port* allowing you to adjust one particular control parameter. Usually, the name of a port is printed at the beginning of a row. For example, the port labeled *Slice Number* allows you to change the slice number via a slider.

- Select different slices using the *Slice Number* port.

By default, *Ortho Slice* displays slices with *xy* orientation, i.e., perpendicular to the z-direction. However, the module can also extract slices from the image volume perpendicular to x- and y-direction.

2.5 Interaction with the Viewer

The 3D viewer lets you look at the model from different positions. If you click on the *Trackball* button in the viewer toolbar, moving the mouse inside the viewer window with the left mouse button pressed lets you rotate the object. If you click on the *Translate* or the *Zoom* buttons, you can translate or zoom the object. (For zoom, move the mouse up and down.)
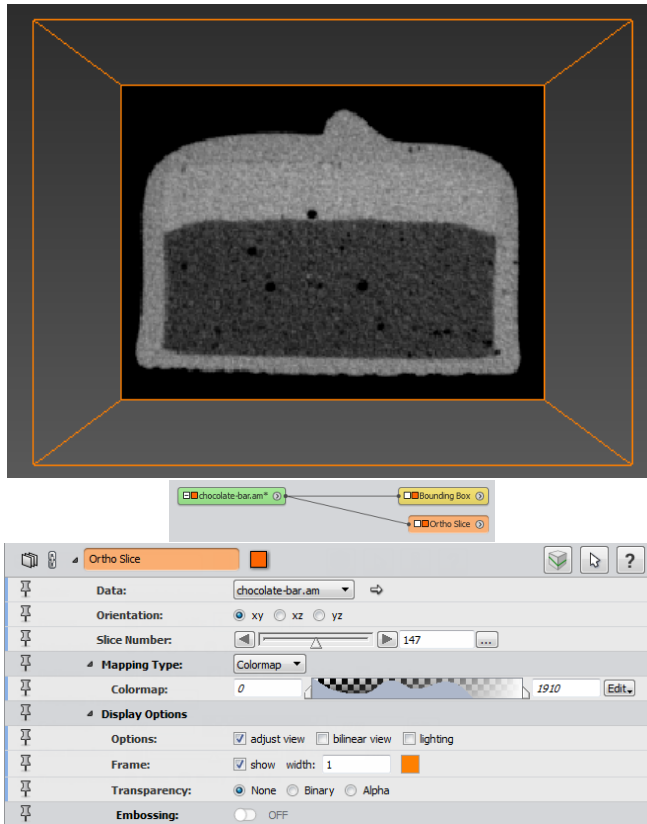
Figure 2.6: Visualization results are displayed in the 3D viewer window. Parameters or ports of a module are displayed in the Properties Area after you select the module.

Alternatively, with the middle mouse button pressed, you can translate the object. For zooming, press both the left and the middle mouse buttons at the same time and move the mouse up or down.

Notice that the mouse cursor has the shape of a little hand inside the viewer window. This indicates that the viewer is in viewing mode. By pressing the ESC key you can switch the viewer into interaction mode. In this mode, interaction with the geometry displayed in the viewer is possible by mouse operations. For example, when using *Ortho Slice* you can change the slice number by clicking on the slice and dragging it.

- Select different buttons of the *Orientation* port of the *Ortho Slice* module.
- Rotate the object in a more general position.
- Disable the *adjust view* toggle in the *Options* port.
- Change the orientation using the *Orientation* port again.
- Choose different slices using the *Slice Number* port or directly in the viewer with the interaction mode described above.
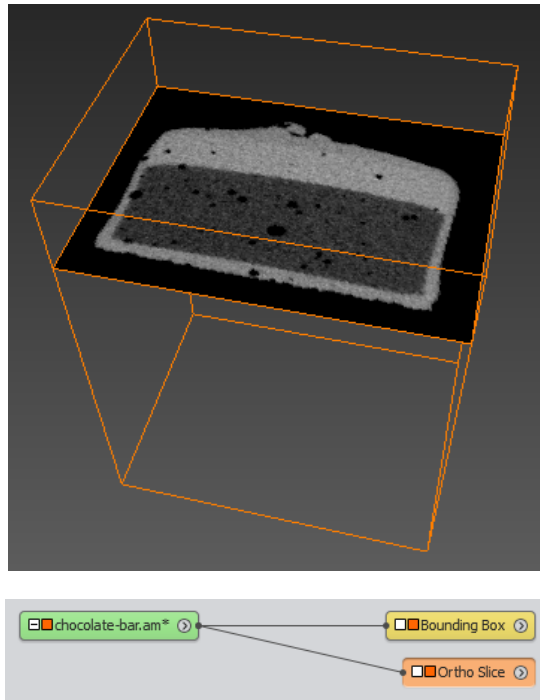
Figure 2.7: The *Ortho Slice* module is able to extract arbitrary orthogonal slices from a regular 3D scalar field or image volume.

Each display module has a *viewer toggle* by which you can switch off the display without removing

the module. This button is just to the right of the colored bar where the module name is shown, as illustrated below.

- Deactivate and activate the display of the *Ortho Slice* or *Bounding Box* module using the *viewer toggle*.
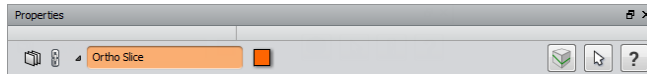
If you want to remove a module permanently, select it and choose *Remove Object* from the *Project View* menu. Choose *Remove All* from the same menu to remove all modules.

- Remove the *Bounding Box* module by selecting its icon and choosing *Remove Object* from the *Project View* menu.
- Remove all remaining modules by choosing *Remove All Objects* from the same menu.

Now the Project View should be empty again. You may continue with the next tutorial.

2.6   Data Import

Usually, one of the first things Avizo users want to know is how to import their own data into the system. This section contains some advice intended to ease this task.

In the simplest case, your data is already present in a standard file format supported by Avizo. To import such files, simply use *Open Data* in the *File* menu or use the *"Open Data..."* green button in the project view. A *list of all supported formats* can be found in the index section of the user's guide. Usually, the system recognizes the format of a file automatically by analyzing the file header or the filename suffix. If a supported format is detected, the file browser indicates the format name.

Often, *3D image volumes* are stored slice by slice using standard 2D image formats such as TIFF or JPEG. In case of medical images, slices are commonly stored in ACR-NEMA or DICOM format. If you select multiple 2D slices simultaneously in the file browser, all slices will automatically be combined into a single 3D data set. Simultaneous selection is most easily achieved by first clicking the first slice and then shift-clicking the last one.

If your data is not already present in a standard file format supported by Avizo, you will have to write your own converter or export filter. For many data objects such as 3D images, regular fields, or tetrahedral grids, Avizo's native *Avizo* format is most appropriate. Using this format you can even represent point sets or line segments for which there is hardly any other standard format. The *Avizo format documentation* explains the file syntax in detail and contains examples of how to encode different data objects. One important Avizo data type, triangular non-manifold surfaces, cannot be represented in a *Avizo* file but has its own file format called *HxSurface* format.

Alternatively, with Avizo XPand Extension, a C++ programmer can extend Avizo in order to integrate a custom reader or writer.

Finally, in case of images or regular fields with uniform coordinates, you may also read *binary raw data.* Note that for raw data the dimensions and the bounding box of the data volume must be entered manually in a dialog box which pops up after you have selected the file in the file browser.

Chapter 3

Tutorials: Visualizing and Processing 2D and 3D Images

Avizo provides extensive support for importing, visualizing, and processing 2D and 3D images. For an overview of related features, please refer to the section 1.2. The tutorials in this chapter introduce the following topics:

- *Reading images* - how to read images
- *Visualizing 3D images* - slices, isosurfaces, volume rendering
- *Combined 3D and 2D views with Ortho Views* - how to efficiently visualize and examine 3D images
- *Intensity Range Partitioning* - segmentation of 3D image data
- *Image segmentation* - segmentation of 3D image data
- *Deconvolution for light microscopy* - powerful algorithms for improving the quality of microscopic images

3.1   How to load image data

Loading image data is one of the most basic operations in Avizo. Other than with 2D images, there are not many standardized file formats containing 3D images. This tutorial guides you by means of examples on how to load the different kinds of 3D images into Avizo. In particular this tutorial covers the following topics:

1. Using the *File/Open Data...* browser and setting the file format.
2. Reading 3D image data from multiple 2D slices.

3. Setting the bounding box or voxel size of 3D images.
4. The *Stacked Slices* file format.
5. Working with Large Disk Data.

3.1.1 The Avizo File Browser

Image data is loaded in Avizo with the *File/Open Data...* dialog. All *file formats* supported by Avizo are recognized automatically either by a data header or by the file name suffix. What follows is only of concern in these cases:

- The automatic file format detection fails.
- 3D image data is stored in several 2D files.
- The data is larger than the available main memory.

Setting the file format
In most cases the format of a file is determined automatically, either by checking the file header or by comparing the file name suffix with a list of known suffixes. In the load dialog, the file format is displayed in a separate column in detail view.

Example:

- Files containing the string `Avizo` in the first line are considered *Avizo* files.
- Files with the suffix .stl are considered STL files.

If automatic file format detection fails, e.g., because some non-standard suffix has been used, the format may be set manually using the *File/Open Data As...* dialog.

3.1.2 Reading 3D Image Data from Multiple 2D Slices

A common way to store 3D image data is to write a separate 2D image file for each slice. The 2D images may be written in TIFF, BMP, JPEG, or any other supported image *file format*. In order to load such data in Avizo, all 2D slices must be selected simultaneously in the file browser. This can be done by clicking the first file and shift-clicking the last one.

- Open the *File/Open Data...* dialog.
- Browse to the `$AVIZO_ROOT/data/teddybear/` directory.
- Select the first file *teddybear000.jpg*
- `Shift`-click the last file (`teddybear061.jpg`).
- Click *Open*.
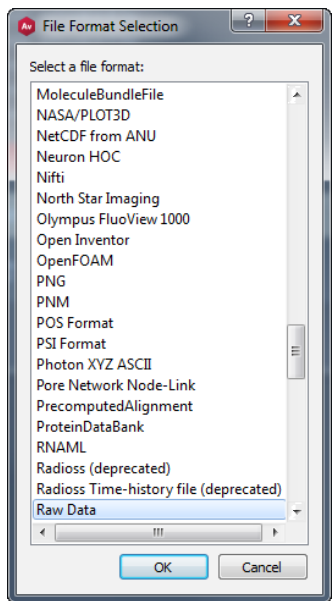
Figure 3.1: The *Format* option of the file browser
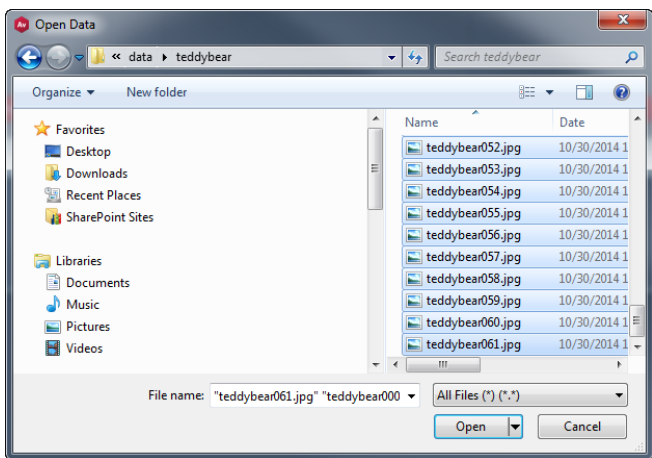
Figure 3.2: Loading multiple 2D images
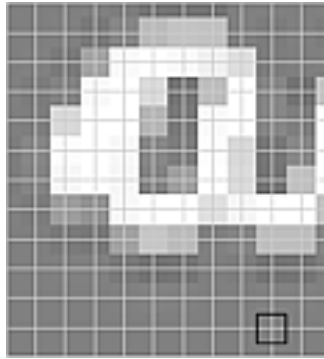
Figure 3.3: The definition of the bounding box in Avizo. Different gray shades depict the intensity values defined on the regular grid (white lines). The black square depicts the extent of one voxel. The outer frame depicts the extent of the bounding box.

3.1.3   Setting the Bounding Box

When loading a series of bitmap images, usually the physical dimensions of the images are not known to Avizo. Therefore, an `Image Read Parameters` dialog appears that prompts you for entering the physical extent of the *Bounding Box*. Alternatively, the size of a single voxel can be set. In Avizo, the bounding box of an object is the smallest rectangular, axis-aligned volume in 3D space that encompasses the object. *Note that in Avizo, the bounding box of a uniform data set extends from the center of the first voxel to the center of the last one. For example, if you have 256 voxels and you know the voxel size to be 1 mm, the bounding box should be set to 0 - 255 (or to some shifted range).*

- Enter 1 in the first, second, and third text field of the **Voxel Size** port.
- Click *OK*.

This method will always create a data set with uniform coordinates, i.e., uniform slice distance. In case of variable slice distances, the *StackedSlices* format should be used.

3.1.4   The Stacked Slices file format

Especially with histological serial sections, it often happens that slices are lost during preparation. To handle such cases, Avizo provides a special data type corresponding to a file format, called *Stacked Slices*. This file format allows a stack of individual image files to be read with optional z- values for each slice. The slice distance is not required to be constant. The images must be one-channel or RGBA images in an image format supported by Avizo (e.g., TIFF). The reader operates on an ASCII description file, which can be written with any editor. Here is an example of a description file:

```
# Avizo Stacked Slices
```

How to load image data   **63**

```
# Directory where image files reside
pathname C:/data/pictures
# Pixel size in x- and y-direction
pixelsize 0.1 0.1
# Image list with z-positions
picture1.tif 10.0
picture7.tif 30.0
picture13.tif 60.0
colstars.jpg 330.0
end
```

Some remarks on the syntax:

- `# Avizo Stacked Slices` is an optional header that allows Avizo to automatically determine the file format.
- `pathname` is optional and can be included if the pictures are not in the same directory as the description file. A space separates the tag "pathname" from the actual pathname.
- On Windows systems, pathname should still be specified with / and not \.
- `pixelsize` is optional, too. The statement specifies the pixel size in x- and y-directions. The bounding box of the resulting 3D image is set from 0 to pixelsize*(number_of_pixels-1).
- `picture1.tif 10.0` is the name of the slice and its z-value, separated by a space character.
- `end` indicates the end of the description file.
- Comments are indicated by a hash-mark character (#).

3.1.5 Working with Large Disk Data

Sometimes image data are so large that they do not fit into the main memory of the computer. Since the Avizo visualization modules rely on the fact that data are in physical memory, this would mean that such data cannot be displayed in Avizo. To overcome this, a special purpose module is provided that leaves most of the data on disk and retrieves only a user-specified subvolume. This subvolume can then be visualized with the standard visualization modules in Avizo.

- Use the *File/Open Data As...* dialog and go to `$AVIZO_ROOT/data/grain/`
- Select the `grain.am` and press the *Open* button.
- Select *Avizo as Large Disk Data* as format and confirm your choice with *OK*.

The data will be displayed in the Project View as a regular green data icon. The info line indicates that it belongs to the data class *HxRawAsExternalData*.

- Right-mouse click, attach an *Extract Subvolume* module.

- Select the *Extract Subvolume* module in the Project View and press the *Max width*, *Max height* and *Max depth* buttons.
- Check *Subsample* and enter 2 2 2 into the *Subsample* fields and press the *Apply* button.

This retrieves a down-sampled version of the data. Disconnect the grain.view data icon from the *Extract Subvolume* module by selecting NO SOURCE in the Master port of grain.view, and use it as an overview (e.g., with *Ortho Slice*). Selecting NO SOURCE in the DATA port of *Extract Subvolume* doesn't disconnect grain.view from *Extract Subvolume*

- Selecting the *Extract Subvolume* module in the Project View and deselect the *subsample* check box.
- Use the dragger box in the viewer to resize the subvolume.
- Press the *Apply* button.
- Attach an **Isosurface** module to the grain2.view (set threshold set to 100).
- Press Apply.

Otherwise, the Isosurface is not displayed.

Tip: To browse the data, check the *auto-refresh* check box for the *Extract Subvolume* and Isosurface modules. Now each time the blue subvolume dragger is repositioned, the visualization is updated automatically.

Loading *Avizo*, *StackedSlices*, and *Raw "as Large Disk Data"* is a convenient and fast way of exploring data that exceed the size of system memory. However, especially with *StackedSlices*, it is not always the most efficient way of doing this. Avizo can store the image data in a special format that facilitates the random retrieval of data from disk.

- Choose from the *File* menu *Convert To Large Data Format.*
- Click *Browse* from the **Input** port.
- Click *Add*, go to $AVIZO_ROOT/data/grain/ and select grain.am, then click *OK*.
- Click *Browse* from the **Output** port.
- Go to a temporary folder and enter a file name of your choice.
- Press the *Apply* button.

Although you will most likely not notice any difference with the small image data used in this tutorial, this method for retrieving large data significantly accelerates the *Apply* operation.

3.1.6 Working with out-of-core data files (LDA)

The out-of-core management tools allow you load and visualize data sets larger than the amount of RAM installed on your system, as well as convert these data sets into LDA (Large Data Access) files. These LDA files can be used to visualize very large data (hundreds of gigabytes), such as seismic
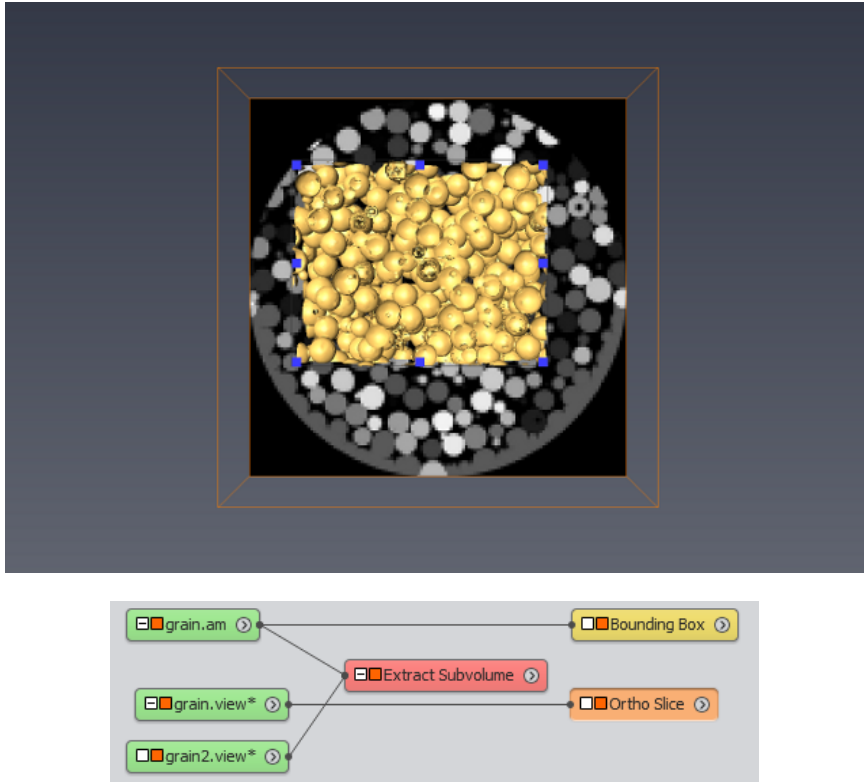
Figure 3.4: The usage of *Avizo* as *Large Disk Data*. For instantaneous update, the *auto-refresh* check boxes of the *Extract Subvolume* and *Isosurface* modules have been checked

or microscopy data, using a limited amount of memory. It is possible to convert original data of the following types: Avizo, RawData, and StackedSlices (stacks of SGI, TIFF, GIF, JPEG, BMP, PNG, JPEG2000, PGX, PNM, and RAS raster files). LDA data allows subvolume extraction to display parts of the volume, or multi-resolution access to have a full subsampled view or accurate refined local views.

Note: in standard, Avizo can support up to 8GB size LDA data. Above 8GB, you will need the Avizo XLVolume Extension.

In particular, the following topics will be discussed in this tutorial:

1. Adjusting the size threshold to allow conversion
2. Loading the out-of-core data
3. Raw data parameters
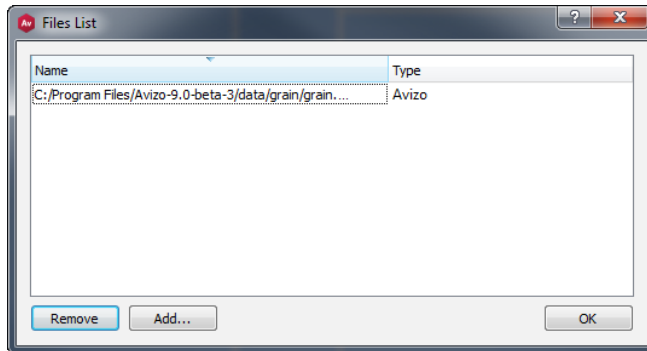
Figure 3.5: The *Input* dialog of the *Convert To Large Data Format* module.

4. Out-of-core conversion
5. Displaying an ortho slice, a slice, and a 3D volume

Please follow the instructions below. Each step builds on the step before.

3.1.6.1 Tune the Size Threshold to Enable Conversion

In the *Edit* menu, select the *Preferences* item. This opens the Avizo*Preferences* dialog. Please select the *LDA* tab (see Figure 3.6). Using the slider or text field, set the threshold to 10MB. When you load a file of file size greater than this threshold, the out-of-core data dialog will be displayed.

Note: To see the images as laid out in this tutorial, you should also use the *Layout* tab of the *Edit/Preferences* menu, and toggle on *show viewer in top-level window*.

3.1.6.2 Load the Out-of-core Data

Please open the file grain.raw using the *File/Open Data...* menu (see Figure 3.7). The file is located in `$AVIZO_ROOT/data/tutorials/outofcore/` in the Avizo install directory. Its size is 16MB, above the defined threshold.

The Out-Of-Core data dialog is displayed. Three loading options are displayed (see Figure 3.8):

- *Convert to Large Data Access (LDA) format*: convert the file to an LDA file, and then load it.
  - *PRO*: This builds a multiresolution file allowing full interactive view or local full resolution viewing.
  - *CON*: This can be time consuming, with an initial pass and then the true conversion pass.
- *Read as External Disk Data*: read data blocks from disk, allowing almost continuous disk access.
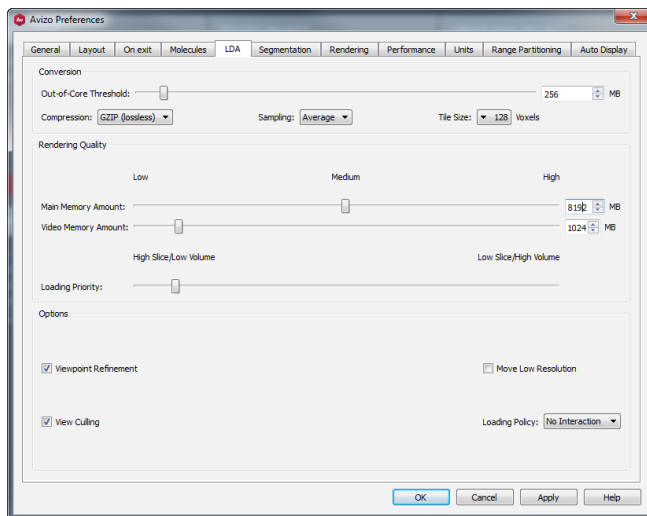
Figure 3.6: Avizo Preferences, LDA settings
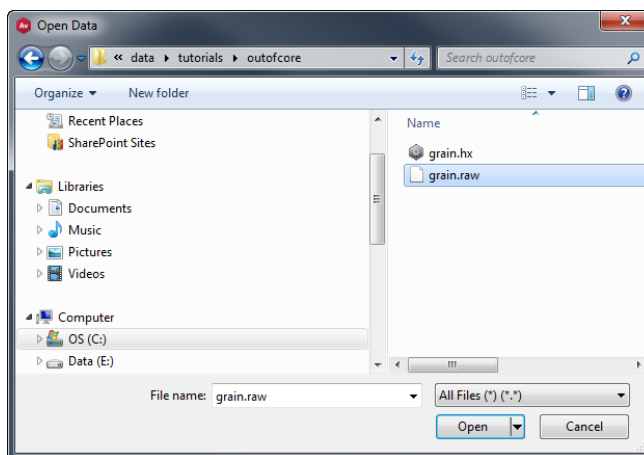
Figure 3.7: Open the out-of-core data file
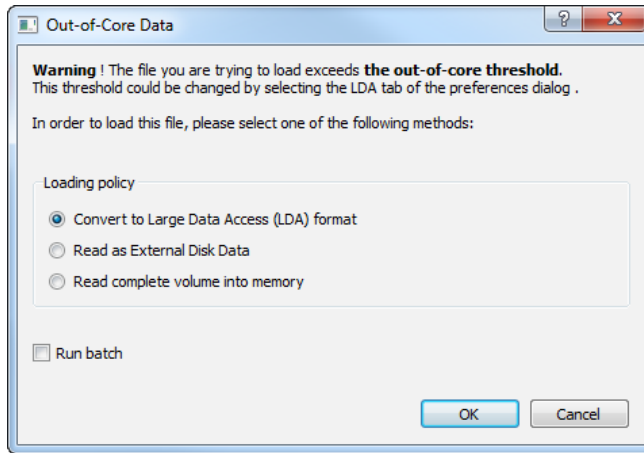
Figure 3.8: Out-of-Core data dialog

- - *PRO*: No need to generate an extra file.
  - *CON*: Continuous access to disk. Slow with data sets larger than 4GB.
- *Read complete volume into memory*: load full data into memory and then access to memory only.
  - - *PRO*: Adapted for average sized data.
    - *CON*: Requires as much RAM as your data set size. Usually not applicable for data sets greater than 500MB or 1GB.

Please select "Convert to Large Data Access (LDA) format". Then, on the next dialog (Destination file), specify the LDA destination file. grain.lda in a temporary folder for instance (see Figure 3.9).

Note: A `.lda` file can be loaded then, without any conversion required.

Another option allows you to perform conversion in batch mode so you can run other processes while the conversion is done in the background.

3.1.6.3 Raw Data Parameters

As the input data is raw, please fill in the raw data parameters dialog with information as on the following figure (see Figure 3.10):

Data type is "8-bit unsigned", dimension 256*256*256.
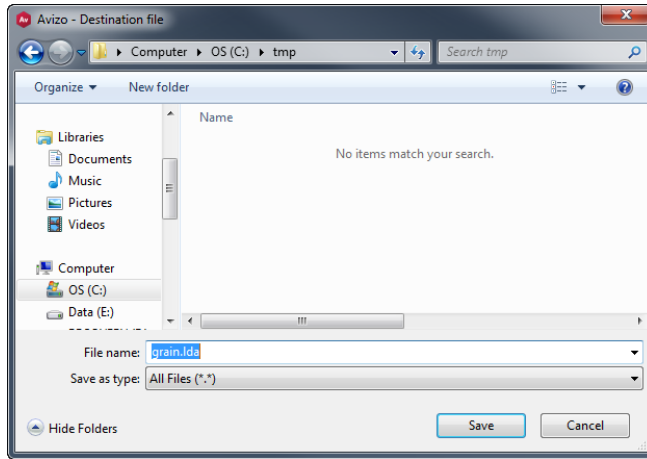
Figure 3.9: Choose LDA destination file
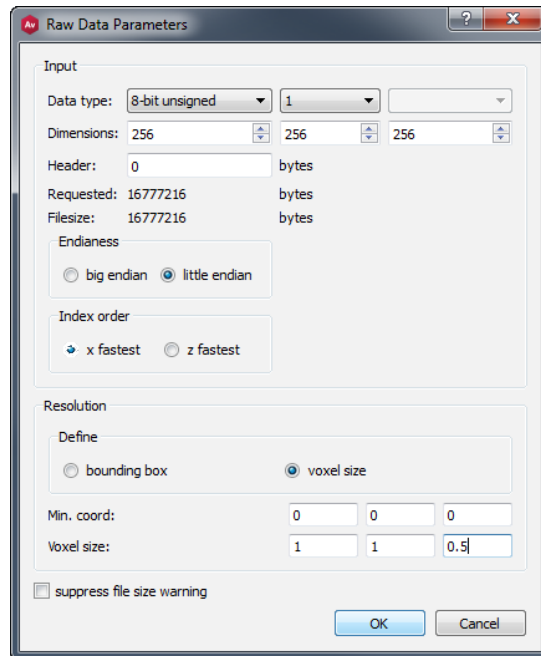
Figure 3.10: Raw data parameters panel

3.1.6.4 Out-of-core Conversion

During conversion, the out-of-core conversion progress is showed in the progress bar (see Figure 3.11). This process is done in two steps. First of all, an initial step, and then the conversion step at about 4MB/s (on a P4 2.6GHz, no SATA disk). You can cancel the process if you wish.

Figure 3.11: Out-of-core conversion progress dialog

The converted file is now in the Project View ready to be used and connected to other modules.

3.1.6.5 Display an Ortho Slice, a Slice, and a 3D Volume

Right-click on the data icon in the Project View. In the Display submenu, choose the *Ortho Slice* module. Repeat these steps for a *Slice* and a *Volume Rendering*. You can also display the bounding box of the full volume.

In order to view this scene with the same settings, after converting grain.raw into grain.lda (lda file required, with the right name) please load the project `grain.hx` (in the same directory `$AVIZO_ROOT/data/tutorials/outofcore`).

3.2 Visualizing 3D Images

This section provides a step-by-step introduction to the visualization of regular scalar fields, e.g., 3D image data. Avizo is able to visualize more complex data sets, such as scalar fields defined on curvilinear or tetrahedral grids. Nevertheless, in this section we consider the simplest case, namely scalar fields with regular structure. Each step builds on the step before. In particular, the following topics will be discussed:

1. orthogonal slices
2. simple threshold segmentation
3. resampling the data
4. displaying an isosurface
5. cropping the data
6. volume rendering

We start by loading the data you already know from Section 2 (Getting Started): a 3D image data set of a part of a 235 x 175 x 295 CT scan of a chocolate bar.
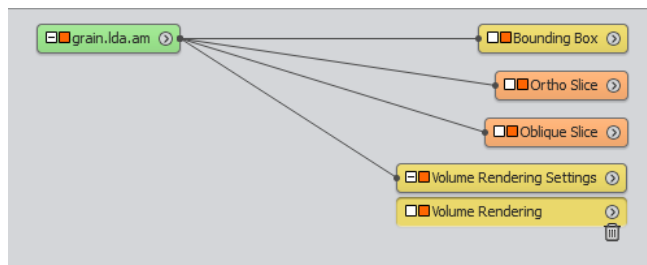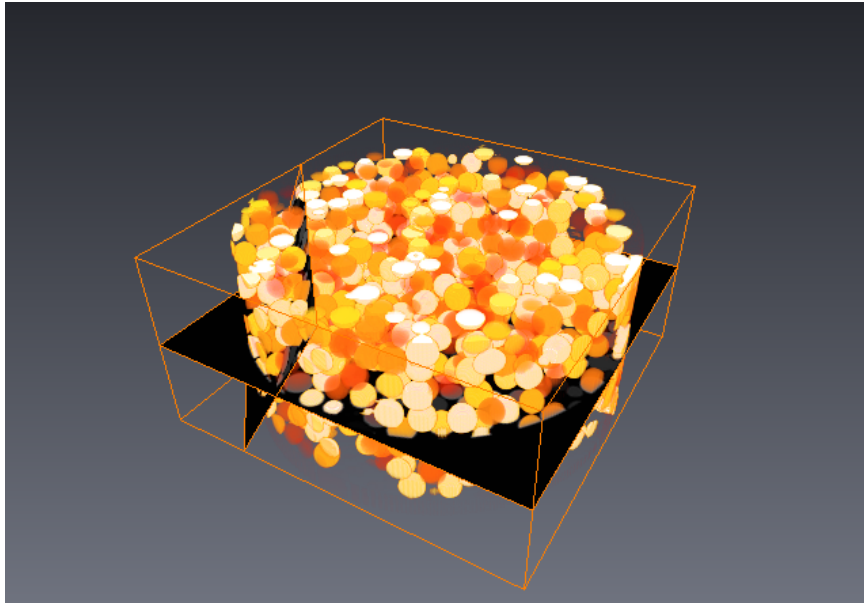
Figure 3.12: Project display (viewer and Project View)

- Load the file `chocolate-bar.am` located in subdirectory `data/tutorials`.

3.2.1 Orthogonal Slices

The fastest and in many cases most "standard" way of visualizing 3D image data is by extracting orthogonal slices from the 3D data set. Avizo allows you to display multiple slices with different orientations simultaneously within a single viewer.

- Connect a *Bounding Box* module to the data (use right mouse on `chocolate-bar.am`).
- Connect an *Ortho Slice* module to the data.

- Connect a second and third *Ortho Slice* module to the data.
- Select *Ortho Slice 2* and press *xz* in the *Orientation* port.
- Similarly, for *Ortho Slice 3*, choose *yz* orientation.
- Rotate the object in the viewer to a more general position.
- Change the slice numbers of the three *Ortho Slice* modules in the respective ports or directly in the viewer as described in the section Getting Started.
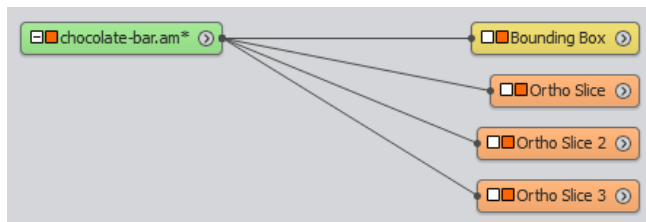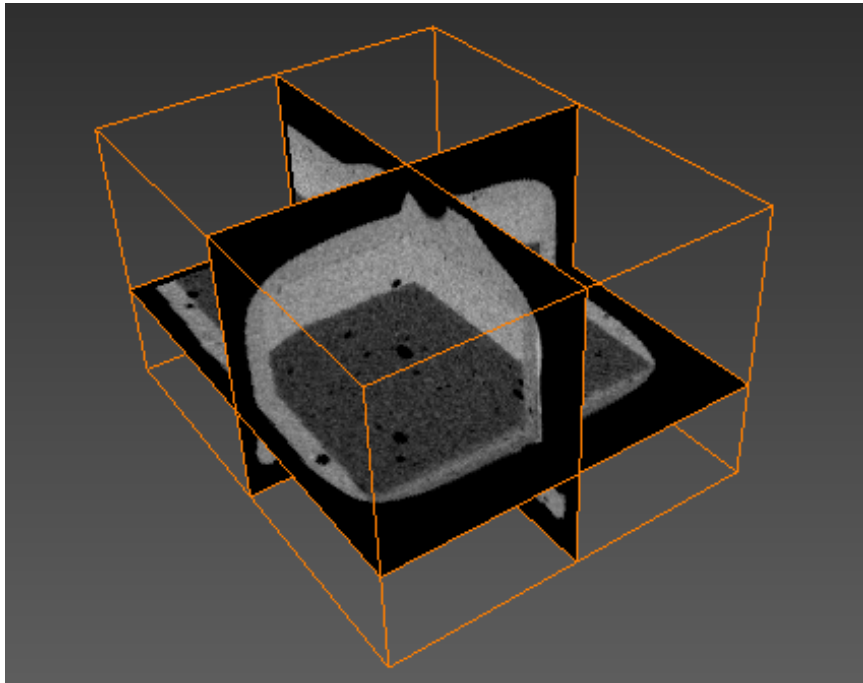
Figure 3.13: Chocolate bar data set visualized using three orthogonal slices.

In addition to the *Ortho Slice* module, which allows you to extract slices orthogonal to the coordinate axes, Avizo also provides a module for slicing in arbitrary orientations. This more general module is

called *Slice*. You might want to try it by selecting it from the Display submenu of the chocolate bar data popup menu.

3.2.2 Simple Data Analysis

The values of the *Colormap* port of the *Ortho Slice* module determine which scalar values are mapped to black or white, respectively. If you choose a range of e.g., 30...100, any value smaller or equal to 30 will become black, and all pixels with an associated value of more then 100 will become white. Try modifying the range. This port provides a simple way of determining a threshold, which later can be used for segmentation, e.g., to separate background pixels from other structures. This can be most easily done by making the minimum and maximum values coincide.

- Remove two of the *Ortho Slice* modules.
- Select the remaining *Ortho Slice* module.
- Make sure that the *Mapping Type* is set to *Colormap*.
- Change the minimum and maximum values of the Colormap port until these values are the same and a suitable segmentation result is obtained. For this data, a value of 410 for the min and 411 for the max should be a good threshold value.

A more powerful way of quantitatively examining intensity values of a data set is to use a data probing module *Point Probe* or *Line Probe*. However, we will not discuss these modules in this introductory tutorial.

3.2.3 Resampling the Data

Now we are going to compute and display an *Isosurface*. Before doing so, we will resample the data. The resampling process will produce a data set with a coarser resolution. Although this is not necessary for the isosurface tool to work, it decreases computation time and improves rendering performance. In addition, you will get acquainted with another type of module. The *Resample* module is a computational module. Computational modules are represented by red icons. Typically you must press the green *Apply* button at the bottom of the Properties Area to start the computation. After you press this button, they produce a new data object containing the result.

- Connect a *Resample* module to the data and select it.
- Enter values for a coarser resolution, e.g., x=64, y=64, z=64.
- Press the *Apply* button.

A new green data icon representing the output of the resample computation named *chocolate-bar.Resampled* is created. You can treat this new data set like the original chocolate bar data. In the popup menu of the resampled chocolate bar you will find exactly the same attachable modules and you can save, export and load it like the original data.
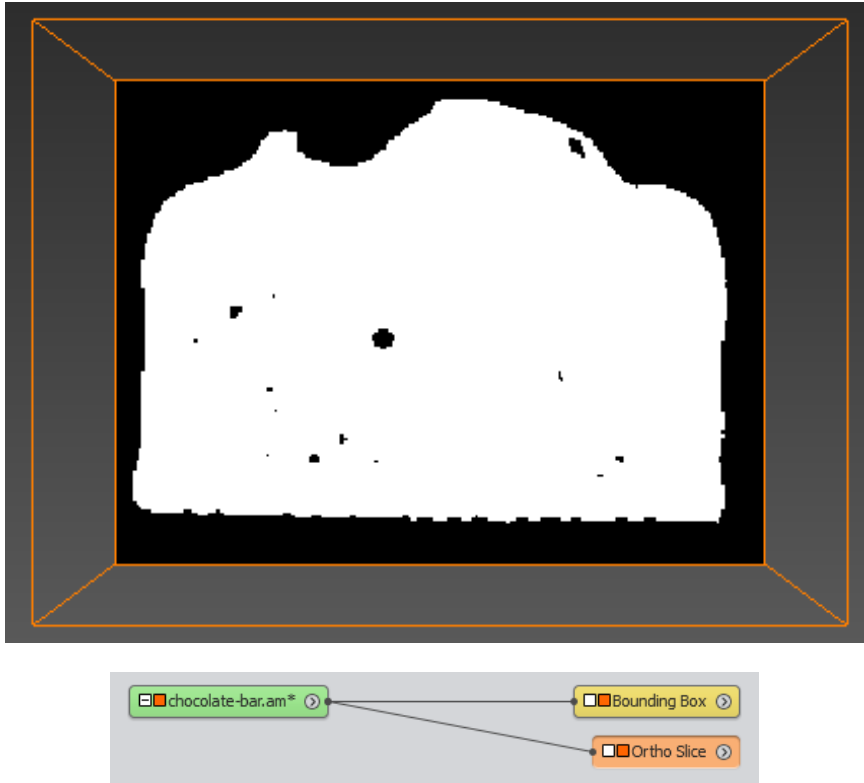
Figure 3.14: By adjusting the data window of the *Ortho Slice* module a suitable value for threshold segmentation can be found. Intensity values smaller than the min value will be mapped to black, intensity values bigger than the max value will be mapped to white.

You may want to compare the resampled data set with the original one using the *Ortho Slice* module. You can simply pick the black line indicating the data connection and drag it to a different data source. Whenever the mouse pointer is over a valid source, the connection line appears highlighted in gray.

3.2.4 Displaying an Isosurface

For 3D image data sets, isosurfaces are useful for providing an impression of the 3D shape of an object. An isosurface encloses all parts of a volume that are brighter than some user-defined threshold.

- Turn off the viewer toggle of the *Ortho Slice* module.
- Connect an *Isosurface* module to the resampled data record and select it.
- Adjust the threshold port to 450 or a similar value.

- Press the *Apply* button.
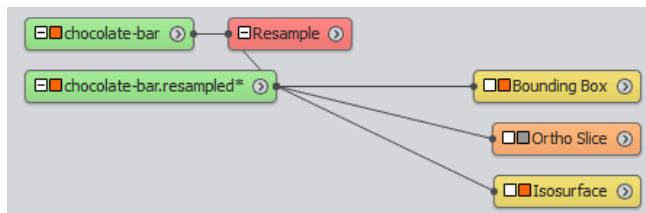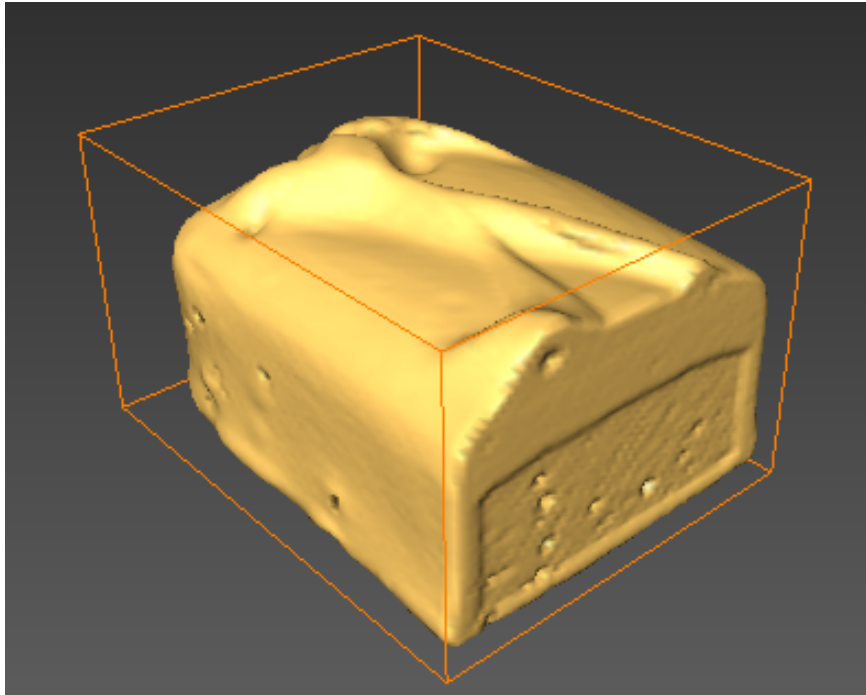
Figure 3.15: Chocolate bar data set visualized in 3D using an isosurface.

3.2.5 Cropping the Data

Cropping the data is useful if you are interested in only a part of the field. A crop editor is provided for this purpose. Its use is described below:

- Remove the resampled data *chocolate-bar.Resampled*.
- Activate the display of the *Ortho Slice* module.
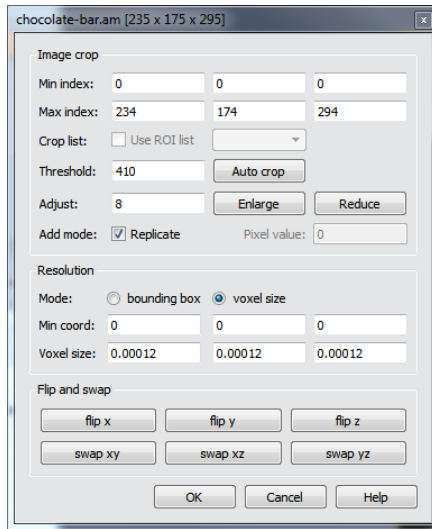
Figure 3.16: The crop editor works on uniform scalar fields. It allows you to crop a data set, to enlarge it by replicating boundary voxels, or to modify its coordinates, i.e., to scale or shift its bounding box.

- Select the *chocolate-bar.am* data icon.
- Click on the Crop Editor button in the Properties Area.

A new window pops up. There are two ways to crop the data set. You can either type the desired ranges of x, y, and z coordinates into the crop editor's window or put the viewer into interaction mode and adjust the crop box using the green handles directly in the viewer window.

- Put the viewer into interaction mode.
- With the left mouse button, pick one of the green handles attached to the crop volume. Drag and transform the volume until the part of the data you are interested in is included.
- Press *OK* in the crop editor's dialog window.

The new dimensions of the data set are given in the Properties Area. If you want to work with this cropped data record in a later sessions, you should save it by choosing *Save Data As ...* or *Export Data As ...* from the *File* menu.

As you already might have noticed, the crop editor also allows you to rescale the bounding box of the data set. By changing the bounding box alone, no voxels will be cropped. You may also use the crop editor to enlarge the data set, e.g., by entering negative values for the *Min index* fields. In this case, the first slice of the data set will be duplicated as many times as necessary. Also, the bounding box will be updated automatically.

Visualizing 3D Images   **77**

3.2.6   Volume Rendering

Volume Rendering is a visualization technique that gives a 3D impression of the whole data set without
segmentation. The underlying model is based on the emission and absorption of light that pertains to
every voxel of the view volume. The algorithm simulates the casting of light rays through the volume
from pre-set sources. It determines how much light reaches each voxel on the ray and is emitted or
absorbed by the voxel. Then it computes what can be seen from the current viewing point as implied
by the current placement of the volume relative to the viewing plane, simulating the casting of sight
rays through the volume from the viewing point.

3.2.6.1   Using Volume Rendering module

- Remove all objects in the Project View other than the *chocolate-bar.am* data record.
- Select the data icon and read off the range of data values printed in the "Data info" line (min-
  max: 0...1910).
- Connect a *Volume Rendering* module to the data.
- Load the chocolate colormap located in
  AVIZO_ROOT/data/colormaps/volrenChocolate.am.
- Change the colormap of Volume Rendering module: in the *Colormap* port, click on "Edit" and
  select "volrenChocolate.am". (if you don't have loaded the colormap, you can still select it by
  clicking on "Edit > Options > Load colormap...").
- Set the range in the *Colormap* port of the *Volume Rendering* to 410..1910.
- Select the *Volume Rendering Settings* and change the lighting to *none*.

By default, emission-absorption volume rendering is shown. The amount of light being emitted and
absorbed by a voxel is taken from the color and alpha values of the colormap connected to the *Volume
Rendering* module. In our example the colormap is less opaque for smaller values. You may try to set
the lower bound of the colormap to 0 or 900 in order to get a better feeling for the influence of the
transfer function.

- Make sure *Volume Rendering Settings* is selected.
- Set *lighting* to *Specular*.

3.2.6.2   Using Volren module

The module *Volren* is the latest addition to volume rendering in Avizo and takes full advantage of the
capabilities of modern graphics boards.

The *Volren* module provides several visualization techniques: shaded and classical texture-based vol-
ume rendering (VRT), maximum intensity projection (MIP), and digitally reconstructed radiograph
(DRR). The standard VRT and its shaded version enable a direct 3D visualization with flexible color
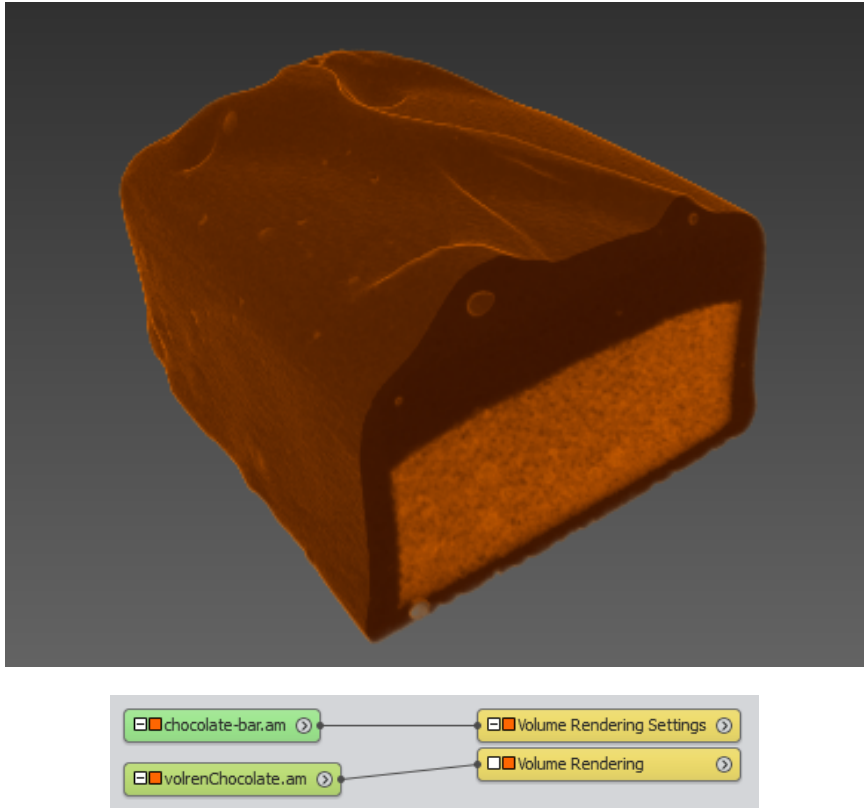
Figure 3.17: The *Volume Rendering* module can be used to generate volume renderings based on an emission-absorption model.

and transparency colormaps with virtual lighting effects for better rendering of complex spatial structures and enhancing fine detail. The MIP rendering allows the visualization of the highest or lowest intensity in a data volume along the current line of sight. The DRR simulates a radiograph display from arbitrary views using the loaded volume data. Furthermore, the *Volren* module enables you to render segmented regions at the same time with different colormaps. A *LabelField* can be attached to the *Volren* and each material of the label list can be rendered separately. In order to optimize the performance, the image volumes are represented at lower resolution when the rendering is done interactively. For relatively large image volumes, you can also switch to a constant "Low Resolution" mode (see below). *Volren* is built for use with popular graphics accelerator technology such as the NVIDIA Quadro FX graphics boards.

- Remove all objects in the Project View other than the *chocolate-bar.am* data record.
- Connect a *Volren* module to the data.
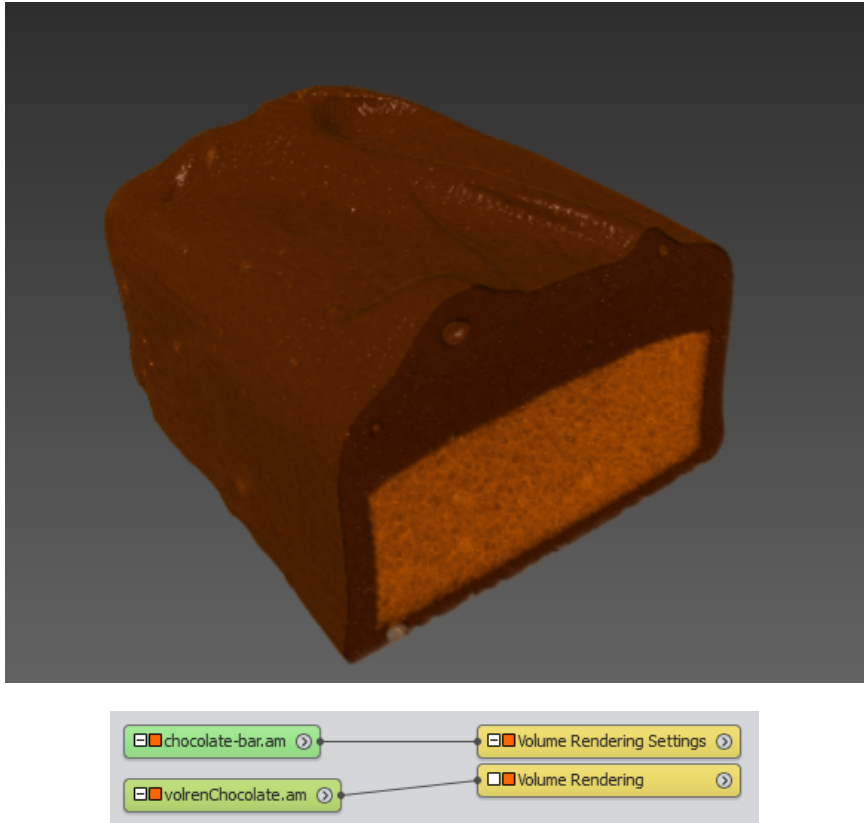
Figure 3.18: Lighting computation are applied to the volume rendering, resulting in an easier to understand representation.

- Select the data icon and read off the range of data values printed on the first info line (410...1910).
- Select the *Volren* module and enter the range in the *Colormap* port.

Notes and Limitations

- On some systems, a significant slowdown can occur if the data set is larger than the available texture memory (which is typically 4 - 16 MB). In this case, select the option LowRes only (see below).
- When two or more *Volren* modules are used to render intersecting volumes (multivolume rendering), some rendering inaccuracies may occur. In case of multivolume rendering, the supported combinations of modes are one MIP together with one MIP or one VRT with one VRT (see below Mode). Other modes and combinations may lead to incorrect visual results.
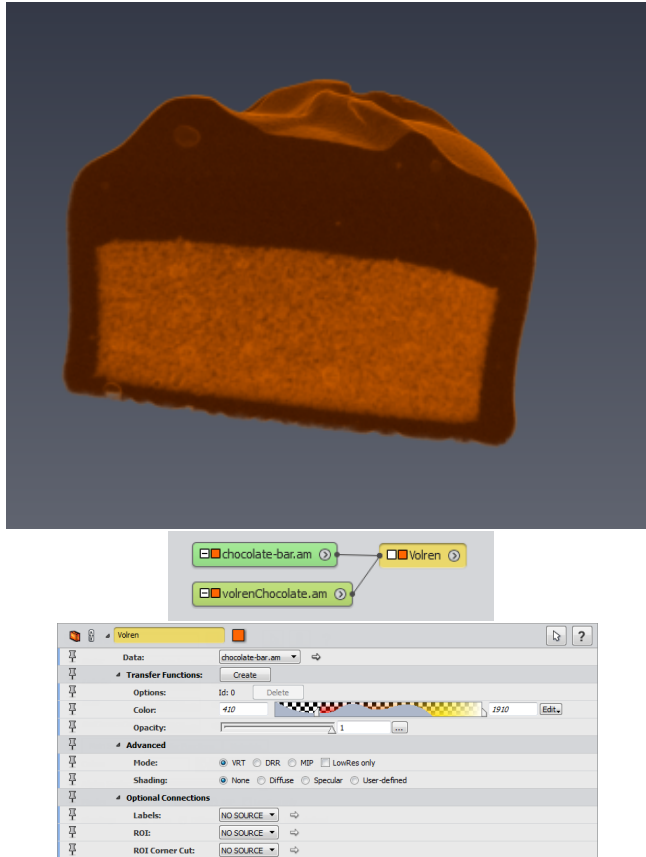
Figure 3.19: The *Volren* module can be used to generate maximum intensity projections as well as volume renderings based on an emission-absorption model.

- The Apply button is enabled only when the rendering must be recomputed.

3.2.6.3 Comparisons between Volren, Voltex and Volume Rendering modules

The *Volume Rendering* module has been developed to take advantage of modern graphics hardware. Rendering may fail on old generation graphics hardware. In some cases this can be solved by updating the graphics driver. Otherwise the *Volren* or the *Voltex* module can be tried alternatively. Comparisons between the three modules are given into Figure 3.20.

Features	*Voltex* **module** (deprecated)	*Volren* **module**	*Volume Rendering* **module**
Mouse Picking	No	No	Yes
Isosurface **Rendering**	No	No	Yes (separate module)
Voxelized **Rendering**	No	No	Yes (separate module)
DDR Rendering simulated Digitally Reconstructed Radiograph	No	Yes	No
MIP Rendering Maximum Intensity Projection	Yes	Yes	Yes
Lighting and Shade effects	No	Yes Diffuse, Specular	Yes Diffuse, Specular, Enhanced, Edge, Boundary, Ambient, Deferred, High Quality
User-defined light coefficients	No	Yes	No
Light angle control	No	Yes Port *Shading: User-defined*	Yes Menu *View / Light* to create custom light and deactivate headlight
Cast shadows	No	No	Yes
Multi-Channel Field input	Yes	No	Yes
Color Field input (RGBA)	Yes	Yes	Yes
Label field secondary input and per-label colormap	No	Yes	No
Label colormap support (cycling, no interpolation)	No	No	Yes
Multi-Volume support	Limited	Yes	Yes
Support for data larger than graphics memory	Yes Very limited bricking with uniform resolution	Yes Full resolution when still, Low resolution during interaction	Yes Progressive adaptive resolution, limited by memory threshold
Support for data larger than the main memory (RAM)	No	No	Yes converting data into *LDA format*
LDA format **support** Large Data Access for progressive loading, instant preview, quick extract from out-of-core data	No	No	Yes
ROI Box: **Region of Interest**	Yes	Yes	Yes
Custom ROI	No	Yes *Corner Cut*: exclusion corner	Yes *ROI Box for Volume Rendering*: exclusion box, fence, cross, sub-volume
Hardware Support	Old graphics boards	Modern graphics boards with up-dated driver (e.g. NVIDIA Quadro)	Modern graphics boards with up-dated driver (e.g. NVIDIA Quadro). Requires support for advanced shaders Requires high-level OpenGL features that may be unavailable or buggy on specific graphic cards, consider using Voltex as an alternative To create a Voltex from the Console, type command: `create HxVoltex`

Figure 3.20: **Comparisons between** *Volren***,** *Volume Rendering* **and** *Voltex* **modules**

3.3 Combined 3D and 2D views with Ortho Views (Avizo)

This tutorial requires an Avizo license (Avizo Lite Edition license is not sufficient).

In this tutorial, you will learn the basics to efficiently visualize and examine 3D images with Avizo:

- *Quick start for viewing 3D images.*
- *Manage combined 3D and 2D slice views with Ortho Views.*

You may use this tutorial as a quick start for visualizing and manipulating 3D images with Avizo. However, you should be familiar with the basic concepts of Avizo to take full advantage of this tutorial. In particular, you should be able to load files, to interact with the 3D viewer, and to connect modules to data modules. All these issues are discussed in Avizo chapter 2 - Getting started. To complement this tutorial, you may also review section 3.2 (Visualizing 3D images) of Avizo Lite Edition about basic 3D image visualization techniques.

3.3.1 Quick start

Here is how to simply examine 3D image data with one 3D view and 3 orthogonal slices in 2D views.

We start by loading the data that was used in the tutorial Section 2 (Getting Started): a 3D image data set of a part of a *CT scan* of chocolate bar (see Figure 3.22).

- Load the file `chocolate-bar.am` in subdirectory `data/tutorials` of the Avizo installation directory.
- Right-click on the green data icon that appeared in the Project View. In the object popup, choose the category entry called *Templates*, then double-click on the entry *Ortho Views Template* (or click on *Create*), see Figure 3.21. You can also use the search field and type the first letters of *Ortho Views Template*, then selecting the module in the list will automatically create it in the Project View.
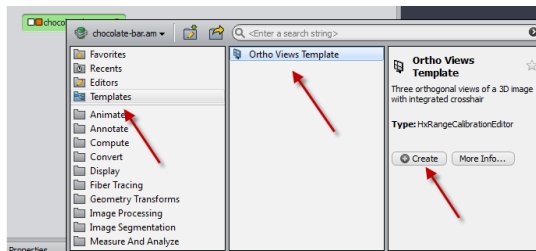
Figure 3.21: Ortho Views Template menu
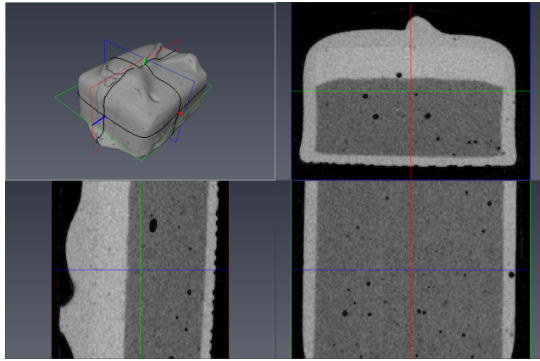
Figure 3.22: CT scan 3D image displayed with Ortho Views Template

You can now try some interaction. You can refer to Section 2 (Getting Started) in 3D navigation with viewer windows.

- In the 2D viewers, you can zoom by dragging the mouse with the left button pressed, or pan the view with the middle mouse button. For this, the mouse cursor shape should be a magnifying glass icon, otherwise press the [ESC] key or use a toolbar button to switch to viewer navigation mode.
- In the 3D viewer, you can rotate the view. For this, the mouse cursor shape should be a hand icon, otherwise press [ESC] key or use a toolbar button to switch to viewer navigation mode.
- In the *Properties* panel of the *Ortho Views* module, you can drag the sliders controlling the slice number.
- You can alternatively pick crosshair lines in the 2D views to move the corresponding slices. For this, you may need to switch to *pick interaction* mode by pressing the [ESC] key or pressing viewer toolbar button *Interact*.

A few explanations before going to the next tutorial steps.

The so-called *Template* menu you have used creates in the Project View a set of *display modules* (representations) attached to the data:

1. an *Ortho Views* managing the display of three orthogonal slices in a 4-viewers window layout. The next section describes it in more details.
2. a *Intensity Ranges Contours* overlay module displaying over the orthogonal slices a set of contours corresponding to intensity levels defined by *Intensity Range Partitioning*, a concept explained later in this chapter.
3. a *Volume Rendering* module displaying the 3D image in the top-left viewer window.
4. an *Isosurface Rendering* module, initially invisible in the viewer windows.

When a module is selected in the Project View (by clicking on the module icon in the Project View), its control parameters - called *ports* - are shown in the *Properties* panel. To select multiple modules, hold down the shift key while clicking.

Tip: Refer to chapter 9.1.1 (Template Projects) for learning how to create template projects.

3.3.2 Manage combined 3D and 2D views with Ortho Views

The *Ortho Views* module extends slice visualization in Avizo. With the Ortho Views module, three different axial views are simultaneously displayed, each in their own axially locked viewer.

When Ortho Views is created, the viewer panel is switched to a four-viewer layout. All three orthogonal views are displayed in the first, top-left viewer. And the initial behavior is for top-right viewer to display the XY view, bottom-right the XZ view and bottom-left the YZ view.

The displayed slices are synchronized so when a slice index is changed, the change will be reflected in all appropriate viewers.

There are four ways you can try to change the slice index for a displayed slice:

- Drag the slider in the properties port.
- Drag a slice directly in the 3D viewer.
- Drag a slice-indicator line (crosshair) in the 2D viewers. If you drag the green line, the green-outlined slice will change.
- In any of the four viewers, click the middle mouse button while pressing the Ctrl key: this will move the slices to the corresponding picked location. For this, the mouse cursor shape should be a arrow icon, otherwise press [ESC]

A slice can be used to clip the representations in the 3D view:

- In the *Properties* panel, click on the clip button (the little cube split by a blue line, see Figure 3.23) of the *Slice Number Z* ports of the Ortho View (if the slider is not visible in *Properties* panel, select the Ortho Views module in the Project View). You will then see a cut in the 3D volume rendering representations (see Figure 3.24).
- Move the Z slice by any means explained above and see the updated 3D view.
- Rotate the 3D view (see above about navigation). The clipped side is always facing the camera. Note that this is as specific feature of Ortho Views, in contrast with the standard behavior for clipping planes in other slice modules in Avizo.

Visibility of slices in the 2D and 3D viewers can be controlled in the standard way with Avizo by clicking on the orange square buttons in the icon in Project View, or beside the module title in the Properties panel. See chapter 8.1.9 about in Project Views.

- You can hide slices, intensity ranges contours, and crosshair, and show volume rendering, as
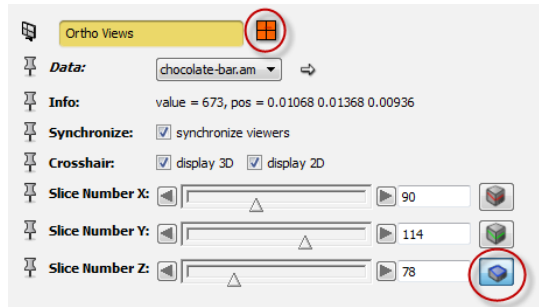
Figure 3.23: Visibility buttons and clip button in Ortho Views module
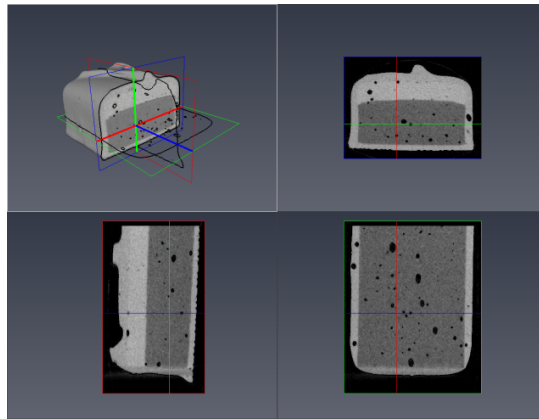
Figure 3.24: Ortho View clipping

shown in Figure 3.25. You will need to use visibility buttons of Ortho Views, Bkg Properties, Intensity Ranges Contours and Volume Rendering modules as shown in Figure 3.26.

Going further: here are some more hints for using *Ortho Views*, see the related reference help for more details.

- To quit Ortho Views mode, you can simply delete the Ortho Views module. You can also click on the viewer toolbar button to set the active viewer (with white frame) as *Single Viewer*. Ortho Views is then still active: click again on Single viewer button to cycle through the different viewers, and press the *Four viewer* button to restore the Ortho Views layout.

- Overlay representations can be combined with images slices, as illustrated by the Intensity Ranges Contours of the Ortho Views Template described earlier in this chapter. You can also control the display ordering of overlays by the *Layer order* port of the Bkg Properties module
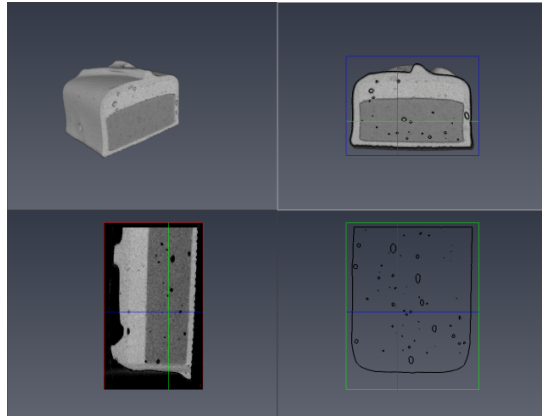
Figure 3.25: Visibility control with Ortho Views
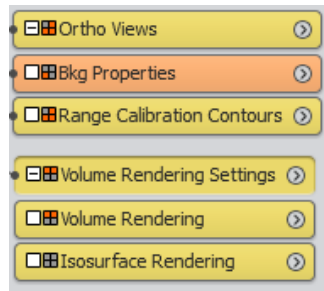
Figure 3.26: Visibility buttons in module icons

associated with Ortho Views.

- Avizo has the capability to combine flexibly multiple data and representations in one or multiple viewers. This is also true with Ortho Views: any 3D representations can be made visible, clipped, or moved to the background of the slice in 2D views (as illustrated below).
- Ortho Views can also be attached to surfaces data, see Figure 3.27.

Note: There can only be one Ortho View in use in the Project, so you can't attach a second Ortho View from a data object menu. It is, however, possible to change data visualized by Ortho View by changing the connection from one image to another. For this, you can either drag the connection line in the Project Graph View, drag the Ortho Views icon to another data item, or change the data connection port in the Properties panel.
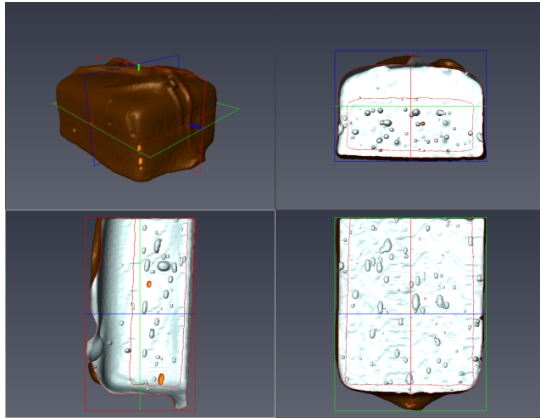
Figure 3.27: Extracted surface displayed with Ortho Views

3.4 Intensity Range Partitioning

This tutorial requires an Avizo license (Avizo Lite Edition license is not sufficient).

The intensity values encoded in grayscale images usually represent some key attribute about the objects. For example, in X-ray CT images the intensity values are linked to mass density. Avizo's Intensity Range Partitioning tools allow you to assign some meaning to the different ranges of intensities in images.

For example, in a CT scan of a rock sample, different intensity ranges may represent:

- surrounding air, pores (void space),
- water,
- clay,
- mineral.

Whereas, in CT scans of industrial parts, intensities may differentiate materials such as:

- surrounding air, pores (void space),
- cast aluminum components,
- cast steel components,
- inclusions.

Avizo's Intensity Range Partitioning tools let you define ranges so that visualization and computation modules can take advantage of this pre-defined identification. Intensity Range Partitioning is used at several locations in Avizo. For example, it allows adjusting the range depicted by a port to a particular

range (which most of the time represents a material). It also greatly simplifies visualization setup. It is possible because the Intensity Range Partitioning contains an "exterior" property which can be set to any range. Then, it's easy to define the "usable" range to be everything not in the "exterior" ranges.

In this tutorial, you will learn how to define and use Intensity Range Partitioning and how it affects visualisation using, as an example, the Volume Rendering representation of a CT scan of an engine block part already used in previous tutorials.

- Let's start from scratch and create a new Project (choose New Project in the File menu).
- Select the entry Preferences in the Avizo Edit menu, and make sure that Intensity Range Partitioning options are set according to the defaults shown on Figure 3.28: automatic guess of two material densities in the loaded image data (i.e., void and solid ranges/regions in the intensities scale).
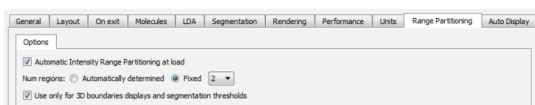
Figure 3.28: Avizo Preferences panel

- Load the file `motor.am` located in subdirectory `data/tutorials` of Avizo installation directory.
- Attach a *Volume Rendering* to the data: right-click on the green data icon appeared in the Project View, click on *Volume Rendering* and press on *Create*, see Figure 3.29
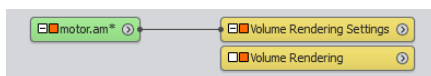
Figure 3.29: Volume Rendering network

As outlined in section 3.2 (Visualizing 3D images), the *Volume Rendering* module displays the intensities of input data voxels with color and opacity depending on the colormap port. With colormaps like the default (volrenWhite.am), the voxels with intensity values below the lower bounds (less than the minimum) of the colormap port are fully transparent, see Figure 3.30. Changing this minimum is an easy way to filter out parts of the 3D volume image that are void or low density materials. Here the colormap bounds have been set according to Intensity Range Partitioning presets on data as explained below.

- Select the *Volume Rendering* module in the Project View and view its colormap port in the Properties panel (see Figure 3.31).
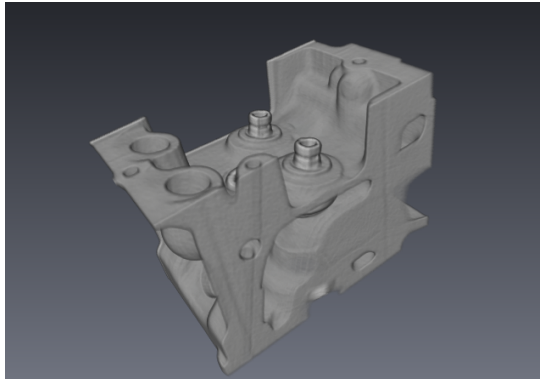
Figure 3.30: Viewing engine block with default Intensity Range Partitioning

- Change the minimum boundary in the colormap port: the *Volume Rendering* is adjusted accordingly. **Tip:** Use the virtual slider to change value continuously: hold down the Shift key while you click and drag the mouse in the numerical text field.
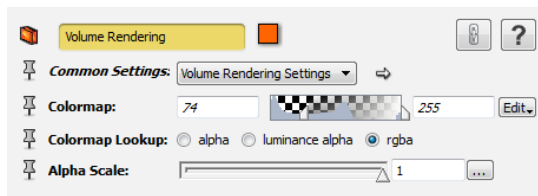
Figure 3.31: Volume Rendering Properties panel

Intensity Range Partitioning intends to provide presets matching to some intensity ranges. This ranges can be easily selected to display a given data set. Avizo can automatically guess thresholds separating different densities of phases or materials by analysing the intensity distribution histogram of the input data. In the example above, Avizo identified an optimal threshold separating void and solid.

- Select the data object motor.am in the Project View and look at the data info in the Properties panel (see Figure 3.32).

You can see: the actual *min-max* of data intensities, the *data window* defining intensity boundaries for exterior to be hidden and used by the colormap port of the *Volume Rendering*, and the number of ranges that have been defined for this data.

- Invoke the *Intensity Range Partitioning editor* by clicking its toggle button (with an histogram ruler symbol) beside data title in the Properties panel as shown below. You can see the defined
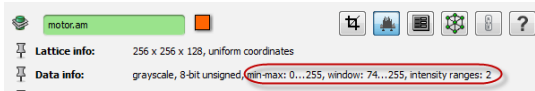
Figure 3.32: Intensity Range Partitioning in data info

ranges for the data set (see Figure 3.33).
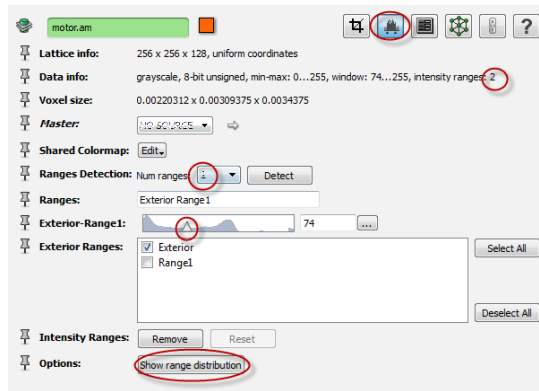
Figure 3.33: Intensity Range Partitioning editor

- Press the *Show range distribution button* to display the data histogram, separated here in two regions/ranges as shown in Figure 3.34.

As you can see, separations between phases/materials are found at local minima of the histogram.

Tip: Other tools are available for automatic threshold determination. See *Auto Thresholding* module.

Note: Intensity Range Partitioning is helpful for making visualizations more easily. For ideal quality images with respect to the features of interest, that can be enough for quantitative analysis. For accurate identification of different phases/materials, in particular with noise, inhomogeous background or artifacts, *image filtering and segmentation* tools may be needed. See the next chapters for more about image segmentation.

The engine block used in this example is actually made of at least two materials of significantly different density and X-ray absorption: aluminum and steel inserts.

Modify the Intensity Range Partitioning as follows:

- Set the Number of ranges to 3, and press Detect. You can see a new slider port and an updated histogram with a new region/range as shown on Figure 3.35.
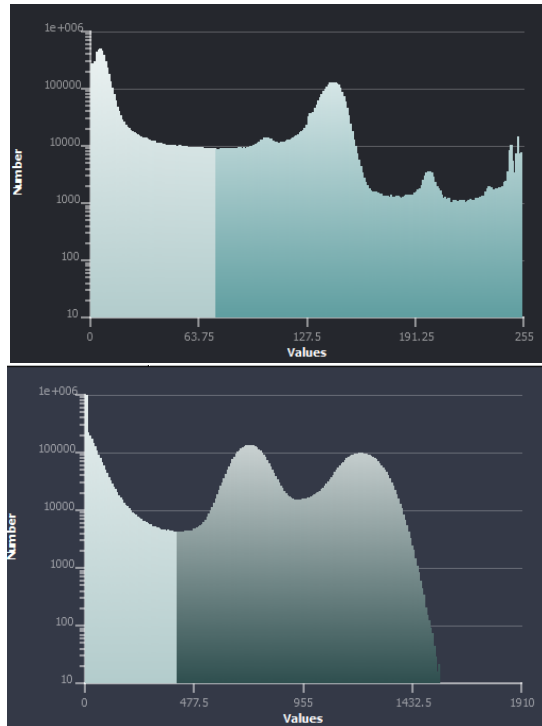
Figure 3.34: Intensity Range Partitioning Histogram

- Change names in Ranges port: Exterior Aluminum Steel
- Set Aluminum as an *Exterior range* : it will be excluded from the *data window* defined above and used as default bounds by display modules. The first range named *Exterior* was set by default as an exterior range.
- Press the *Apply* button at the bottom of the Properties panel. *data info* and editor should be updated as shown in Figure 3.36.
- Now you can close the editor by clicking again on the Intensity Range Partitioning editor toggle button.
- Remove the *Volume Rendering* module from the current project: select the *Volume Rendering Settings* module; press the delete key or drag the icon to the trash can in the Project View.
- Attach a *Volume Rendering* module to the data once again: the aluminum part is hidden now, leaving only higher density steel visible, as shown in Figure 3.37.
- In the colormap port of the *Volume Rendering* module, you can press the Edit button or right-click in a color shaded area to choose a specific range in menu: entire data min-max, data window excluding *exterior ranges*, or specific range defined (see Figure 3.38). For instance,

choosing Aluminum will get the entire block back.
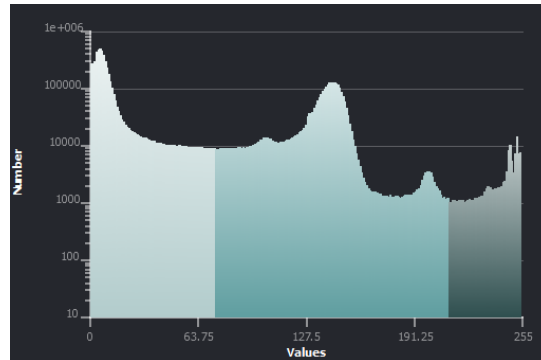
Figure 3.35: Histogram with 3 intensity ranges
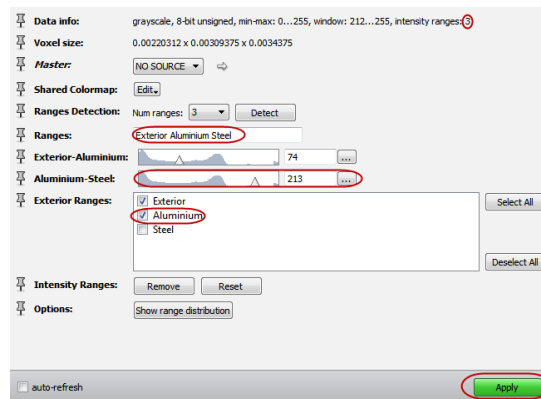
Figure 3.36: Modifying Intensity Range Partitioning

More hints:

- In the Preferences, you can disable Intensity Range Partitioning, if you wish, to get the behavior of the previous Avizo's version and use the default range for data min-max (or Data Window if defined as data parameter).
- You can use the Preferences to set a number of phases/materials to be detected in your data. Keep in mind that detection relies on the shape of the image histogram. Avizo can automatically guess the number of ranges that may be detected, but you may prefer to override this if Avizo fails to find the expected ranges.
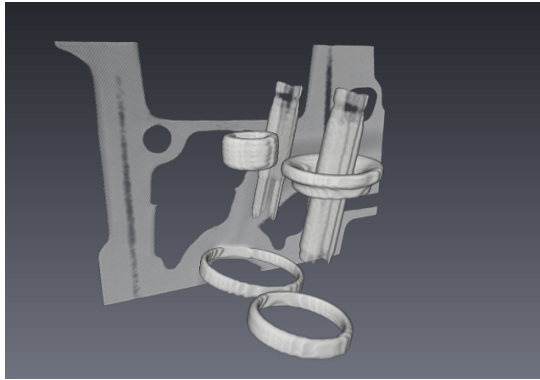
Figure 3.37: Data window now selects high density material
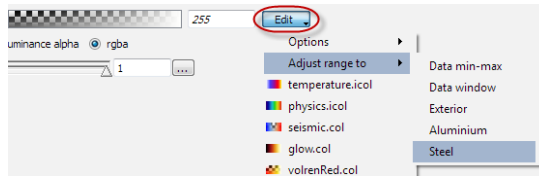
Figure 3.38: Adjusting range in a colormap port

- All modules using colormaps or range slider ports such as Isosurface can take advantage of Intensity Range Partitioning.
- **Tip:** The data window and ranges are defined as persistent data parameters when you save your data to an Avizo format (see Section 8.2.7 Data parameters).

3.5 Segmentation of 3D Images

By following this step-by-step tutorial, you will learn how to interactively create a segmentation of a 3D image. A segmentation assigns a label to each pixel of the image, which describes the region or material associated with the pixel (e.g., material 1 or material 2). The segmentation is stored in a separate data object called a *Label Field*. A segmentation is the prerequisite for surface model generation and accurate quantification such as volume measurement.

The *Segmentation Editor* is a workroom that provides a dedicated user interface for interactive segmentation tasks. While segmenting your data, you can switch between segmentation and other workrooms at any time by clicking the appropriate tab in the *Workroom Toolbar*.

The segmentation process can be automatic, semi-automatic, or interactive depending on the input

images and desired results.

This tutorial introduces each approach and comprises of the following steps:

- Creation of an empty *Label Field*.
- Interactive editing of the labels in the *Image Segmentation Editor*.
- Measuring the volume of the segmented structures.
- An alternative segmentation method called Threshold segmentation.
- Further hints on segmentation.

3.5.1 Interactive Image Segmentation

- Load the file *motor.am* from the directory *data / tutorials*.
- Right-click the green icon and select **Edit New Label Field** from the *Image Segmentation* section.

This will automatically launch the Segmentation workroom thereby replacing the Project View and Properties panels on the left and the 3D viewer on the right. At the same time and not currently visible, a new data object will be added to the Project View that will hold the segmentation results. This is the data object to be saved. By default, the *Segmentation Editor* operates in four-viewer mode. In this mode, the three-slice viewer with XY, XZ, and YZ orientations are shown together with one 3D viewer. Alternatively, single- or two-viewer layouts are possible. In two-viewer layouts (horizontally stacked or side-by-side) a slice viewer is combined with a 3D viewer while in single viewer mode one (XY) slice of the image is shown. You can change the orientation of the slice by either selecting another orientation from the *Segmentation / Orientation* menu or by clicking **Single-viewer** in the viewer toolbar multiple times to cycle the different 2D views (XY, XZ, and YZ) and a 3D view.

The tools of the *Segmentation Editor* are displayed in the panel where the Project View and Properties Area are normally displayed.

- Switch to the four-viewer layout by clicking **Four viewers** in the viewer toolbar.
- In the XY view, use the slider on the bottom to scroll through the slices. Go to slice 30 and you will see six bright structures.
- If necessary, click **-** associated with the *Zoom Tool* in the upper-right corner of the viewer to get a view of the entire slice.
- Click the brush icon in the *Tools* area of the *Segmentation Editor*'s main panel.
- Mark one of the structures with the mouse. Hold down **CTRL** to deselect incorrect pixels, if necessary.
- When finished, select **Inside** in the *Materials* list. Then click **+** in the *Selection* group.

The pixels selected previously are now assigned to material *Inside*. Right-click *Inside* in the *Materials* list to select a different draw style (e.g., dotted).
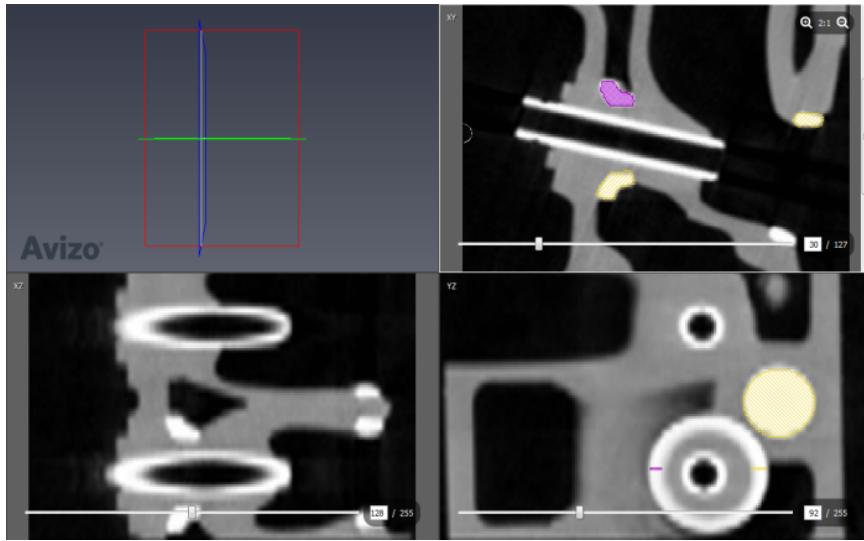
Figure 3.39: Selecting Two Structures by Using One Slice

- Click the *Materials* list and select *New Material* from the right-button menu.
- Mark a second structure using the brush, select the new material in the *Materials* list, and assign the pixels to that structure (see Figure 3.39).
- Go to slice 31 and practice by segmenting the same two structures.

Tip: If you prefer to work with one large view rather than four small views, click **Single Viewer** in the viewer toolbar. To cycle through each of the four views, click **Single Viewer** repeatedly. To return to four-viewer mode, click **Four Viewers**.

If the changes from slice-to-slice are small, use interpolation.

- Go to slice 30 and mark the top-right structure using the brush. Go to slice 35 and mark the same structure.
- Select from the menu bar: *Selection / Interpolate*.
- Scroll through the dataset. You will see that the slices 30 to 35 are now also selected.
- To assign the selected pixels of all slices to the material *Inside*, select it in the *Materials* list, then click **+** in the *Selection* group.
- Repeat the procedure between slice 35 and 45.
- Repeat the procedure for one of the left structures.

Tips:

- While segmenting, it is recommended that you frequently save the segmentation results. To do so, switch to the Project View by clicking **Project** in the *Workroom Toolbar*. Then select the icon of label image and select **Save Data As...** or **Export Data As...** from the Avizo *File* menu. Click **Segmentation** in the Workroom Toolbar to return to the *Segmentation Editor* workroom.
- The *Brush Tool* is probably the most basic segmentation tool. It can be noted that the editor provides additional tools and functions that are described on the *Segmentation Editor help pages*.
- There are useful keyboard short cuts, including **+** (**-**) to add (remove) a selection to (from) a material, **SPACE** and **BACKSPACE** to change the slice number, and **d** to change the material draw style.
- You might want to give materials more meaningful names or different colors. To do so, double-click the material name and enter a new name, or double-click the color button left of the name and select a different color in the *Color Dialog*.

3.5.2   Volume and Statistics Measurement

Once a structure is segmented, you can easily measure its volume:

- In the main menu, select *Segmentation / Material Statistics ...*.

A table appears in a dialog box listing the volume of each material.

The same statistics are available in the **Tables** panel of the **Project** workroom.

- Activate the Project View by clicking **Project** on the workroom toolbar.
- Right-click the icon of the label image and select *Measure and Analyze / Material Statistics ...*.
- Click **Apply** under *Material Statistics* to create a new object *motor.MaterialStatistics* in the Project View.
- Select this object and click **Show**.

A new panel in the application window shows a similar table. The **.MaterialStatistics* object can be saved to text (*.txt), to comma separated values file (*.csv), or to Micrsosoft XML Spreadsheet (*.xml).

Note: If Units Management (see *Edit / Preferences / Units*) is turned off, the physical units of the values in the volume column are implicit and the voxel size scales objects only within this implicit physical unit. No physical units are shown in the column headers. In the case of *motor.am*, the voxel size is in m. Thus, the numbers given in the *Volume* column have to be read as m^3. If Units Management is turned on, the physical units specified as *Display units* are shown in the column header.

3.5.3   Threshold Segmentation

This section describes an alternative way of segmentation that may require less manual interaction, but typically requires the higher quality images. Besides optimizing the image acquisition process,

this can be achieved also by filtering the image before performing the segmentation. Under favorable conditions, a satisfying segmentation can be achieved automatically based solely on the gray values of the image data.

The first step is to separate the object from the background. This is done by classifying the voxels into foreground and background voxels on the basis of their intensities or voxel values.

- Clear the Project View area by selecting *Project / Remove All Objects* from the main menu.
- Load the *motor.am* data record from the directory *data / tutorials*.
- Attach a *Image Segmentation / Multi-Thresholding* module to the data icon and select it.
- Type 85 into the text field of port *Exterior-Range1*. You may also determine some other threshold that separates exterior and interior as described in the tutorial on *Image Data Visualization*.
- Click **Apply**.

With this procedure, each voxel value lower than the threshold is assigned to *Exterior* and each voxel value greater than or equal to the threshold is assigned to *Inside*. However, this may cause voxels to be labeled that are not part of the object but have voxel values above the threshold. Set the *remove couch* option to suppress it, which assures that only the largest coherent area will be labeled as the foreground (*Inside*) and all other voxels are assigned to the background (*Exterior*).

Clicking **Apply** creates a new data object *motor.labels*. This data object is a *Label Field*, has the same dimensions as *motor.am*, and contains an *Range1* or *Exterior* label for each voxel according to the segmentation result.

3.5.4 Refining Threshold Segmentation Results

You can visualize and interactively edit *Label Field* by using Avizo's *Segmentation Editor*. A detailed description of this tool is given in the *Reference* guide. Use the *Segmentation Editor* to smooth the data to remove noise on the surface of the object.

- Select the *motor.labels* icon and click **Segmentation Editor** in the Properties Area.

This automatically activates the *Segmentation Editor* workroom.

- In the YZ view, use the slider on the bottom to select slice 206 (see Figure 3.40).
- Select a suitable magnification by clicking the **+** or **-** buttons of the Zoom Tool (last tool in the Tools area).

The image segmentation editor shows the image data to be segmented (*motor.am*) as well as greyish contours representing the borders between *Range1* and *Exterior* regions as contained in the *motor.labels* data object. As you can see, the borders are not smooth and there are small islands bordered by cyan contours. This is what we want to improve now.
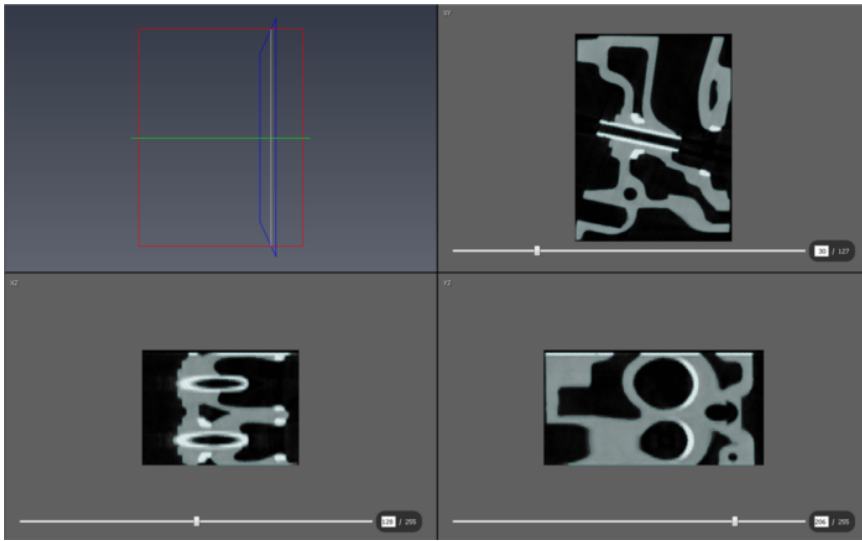
Figure 3.40: Segmented CT Scan Data

- Select **Remove Islands...** from the *Segmentation* menu. In response, a dialog box will appear.
- In the dialog box, select the *All slices* mode. Then click *Highlight all islands* to highlight all islands smaller or equal to 15 voxels.
- Click **Apply** to merge all islands with the material they are embedded in.
- Then click **Close** in order to close the *Islands Filter* window.
- To further clean up the image, select **Smooth Labels...** from the *Segmentation* menu. Another dialog box will appear.
- Select the **3D volume** mode and click **Apply** to execute the smoothing operation.
- Click the **Close** button to close the *Smooth Labels* window.
- To examine the results of the filter operations, browse through the label image slice-by-slice. In addition to the slice slider, you may also use the **cursor-up** and **cursor-down** keys for this.
- Return to the Project View using the workroom toolbar.

3.5.5 More Hints about Segmentation

Avizo includes different tools that can create a label image.

- *Image Segmentation / Multi-Thresholding.* This module automatically creates a label image by thresholding. See also *Selection/Threshold* menu in *Segmentation Editor*.

- *Image Segmentation / Edit New Label Field.* The *Segmentation Editor* provides automatic, semi-automatic, and interactive tools.
- *Compute / Connected Components.* This module searches for connected regions.
- *Image Segmentation / Correlation Histogram.* This module can create a label image from correlated regions in two images sets, typically after registration.
- *Image Segmentation / Interactive Thresholding* and other tools from Avizo. This extension offers advanced tools for quantification, including object separation and labelling. See *Advanced Image Processing, Segmentation, and Analysis* tutorials.

Prior to segmentation, think about improving image quality and reducing noise in the dataset with:

- *Volume Edit* module
- *Visualizing and Processing 2D and 3D Images* tutorials, which provide further filters and tools.

Here are some hints about the *Segmentation Editor*.

- Combine different tools. Remember to toggle 2D / 3D as appropriate.
- Adjust the *Display Control* settings.
- Use Segmentation filters *Smooth labels / Fill holes / Remove islands*.
- Use *Selection / Invert ('I' key)* and combine selections into materials (replace / add / subtract).
- Use interpolate between slices (menu *Selection / Interpolate*).
- Use of four viewers can make the segmentation process easier.
- You can remove erroneous parts from the selection view in all viewers, including the 3D viewer.
- Remember to lock materials to prevent erroneous transfer of previously selected materials.
- Locked material can be used as a mask. This is useful for 3D operations to prevent selection in areas you do not want to add to your material. This process can also be used, so that the magic wand tool only works between specified slices.
- If you want to select a number of materials, hold down **SHIFT** and select the material you want from the *Material* list. If you have the **Show in 3D** check box selected, then you will also get a 3D selection.
- If you want to copy the same selection to a neighboring slice, hold **SHIFT** and press the up / down arrow.
- The state of the current selection is displayed using a label in the *Selection* area. This label can display three different messages: *No selection*, *Active selection*, and *Hidden selection*. The last two states indicate whether an in-plane selection is shown in one of the 2D viewers, or located in a different slice.
- The *Masking* feature allows the user to control the masking by setting its range, and its visibility in 2D/3D viewers, and to enable/disable it in some tools. This *Masking* feature is optional for the *Brush* and *Lasso* tools, and is used in the workflows for the *Magic Wand*, *Threshold*, and *TopHat* tools.

- The *TopHat* tool can be very useful in combination to threshold to segment low contrast or thin features over a non-uniform background.
- The *Segmentation Editor* has other useful *keyboard shortcuts.*

An advanced Segmentation using the *Watershed* tool can be found in the *Watershed Segmentation* tutorial.

3.6   Deconvolution for light microscopy

The deconvolution modules of the Avizo provide powerful algorithms for improving the quality of microscopic images recorded by 3D widefield and confocal microscopes. Two different methods are supported, namely a so-called non-blind and a blind deconvolution method, both based on iterative maximum-likelihood image restoration. In the first case, a measured or computed point spread function (PSF) is required. In the second case, the PSF is estimated along with the data itself.

The deconvolution documentation is organized as follows:

- *General remarks about image deconvolution*
- *Data acquisition and sampling rates*
- *Standard deconvolution tutorial*
- *Blind deconvolution tutorial*
- *Bead extraction tutorial*
- *Performance issues and multi-processing*

The following modules are provided:

- *Extract Point Spread Function* - obtain a PSF from a bead measurement
- *Convolution* - convolve two 3D images
- *Correct Z Drop* - corrects attenuation in z-direction
- *Correct Background And Flat-Field* - background and flatfield correction
- *Deconvolution* - the actual deconvolution front-end
- *Fourier Transform* - computes FFT and power spectrum
- *Generate Point Spread Function* - calculates a theoretical PSF

Examples:

- *Confocal data set*
- *Widefield data set*

3.6.1   General remarks about image deconvolution

Deconvolution is a technique for removing out-of-focus light in a series of images recorded via optical sectioning microscopy. Intended to investigate 3D biological objects, optical sectioning microscopy works by creating multiple images (optical sections) of a fluorescing object, each with a different focus plane. However, besides the in-focus structures, the images usually also contain out-of-focus light from other parts of the object, causing haze and severe axial blur. This is even the case for a confocal laser scanning microscope, where most of the out-of-focus light is removed from the image by a pinhole system. Mathematically, the image produced by any microscopic system can be described as the convolution of the ideal unblurred image of the specimen and the microscope's so-called point spread function (PSF), i.e., the image of an ideal point light source. With the inverse of this process, called deconvolution, a deblurred image of the specimen can be obtained, provided the point spread function is known, or at least can be estimated.

The Avizo deconvolution modules mainly provide two variants of a powerful iterative maximum-likelihood image restoration algorithm, namely a non-blind one and a blind one. The difference between them is that in the first case a measured or computed point spread function is used, while in the second case, the PSF is estimated along with the data itself. Maximum-likelihood image restoration can be considered as the de-facto standard for deconvolution of 3D optical sections. Although computationally quite expensive, the method is able to significantly enhance image quality. At the same time, it is very robust and insensitive with respect to noise artifacts. However, it should be noted that, although rejecting most of the out-of-focus light, by no means all of it is rejected. Therefore, some noticeable haze remains in the images. Also, the images retain a substantial axial smearing in the z-direction, which cannot be removed by any deconvolution algorithm.

At first sight, one may wonder why both a non-blind and a blind deconvolution algorithm are provided; blind deconvolution seems to be more general because the PSF is calculated automatically. One answer is that blind deconvolution is computationally even more expensive than non-blind iterative maximum-likelihood image restoration. The other answer is that in a blind deconvolution algorithm, a meaningful estimate of the PSF can only be computed if severe constraints are imposed. For example, a trivial solution of the blind deconvolution problem would be an image that is identical to the input image and a PSF with the shape of an ideal delta peak. Obviously, this solution isn't useful at all. Therefore, if for example confocal data is to be deconvolved, the algorithm fits the actual PSF in such a way that it looks like a possible measured PSF of a confocal microscope. More precisely, the fit is constrained to be in agreement with the experimental parameters (the refractive index of the medium, the numerical aperture of the objective, and the voxel sizes). Sometimes this can lead to wrong results, for example when the confocal pinhole aperture of the microscope wasn't stopped down sufficiently during confocal image acquisition, in which case the microscope actually didn't behave like a true confocal microscope. As a matter of fact, you should try the approach that provides the best results for your own image data, blind deconvolution or non-blind deconvolution with either a measured or an automatically computed PSF.

3.6.2 Data acquisition and sampling rates

In order to obtain best quality when deconvolving microscopic images, some fundamental guidelines should be obeyed during image aquisition. Good results may be obtained even if some of these guidelines are not followed exactly, but in general the chances to get satisfactory results improve if they are. Below we discuss the most important recommendations.

Adjusting the Scanned Image Volume

The region of interest should be centered in the middle of the image volume, as the optics of the microscope usually has the least aberrations in this region and it helps to avoid possible boundary artifacts, which can arise during the deconvolution procedure. Especially for widefield data, it is important to record a sufficiently large (preferably empty) region below and above the actual sample. Ideally, this region should be as large as the sample itself. For example, if the sample covers 100 micrometers in the z-direction, the scanned image volume should range from 50 micrometers below the sample to 50 micrometers above it.

Choosing the Right Sampling Rate

The sampling rate is determined by the pixel sizes in the x and y directions as well as the distance between two subsequent optical sections, both measured in micrometers. Generally speaking, image deconvolution works best if the data is apparently oversampled, i.e., if the pixel or optical section spacing is smaller than required. The maximal required sampling distance (Nyquist sampling, described in *R. Heintzmann. Band-limit and appropriate sampling in microscopy, chapter 3, Vol. III, in: Cell Biology: A Laboratory Handbook, Julio E. Celis (ed.), Elsevier Academic Press, 29-36, 2006*) to avoid ambiguities in the data can be obtained from considerations in Fourier-space yielding

$$d_{xy} = \frac{\lambda}{4\,NA},$$

where λ denotes the wavelength and NA is the numerical aperture of the microscope. Similar considerations yield for the maximal distance between adjacent image planes:

$$d_z = \frac{\lambda}{2\,n\,(1 - \cos(\alpha))},$$

where n denotes the refractive index of the object medium and α the aperture half angle as determined by $NA = n\,\sin(\alpha)$.

For a confocal microscope, both the in-plane sampling distance and the axial sampling distance need to be, in theory, approximately 2 times smaller. However, this requirement is far too strict for most practical cases and even in the widefield case, approximately fullfilling the above requirements is often sufficient.

The total number of optical sections is obtained by dividing the height of the image volume by the sampling distance d_z. It should be mentioned that deconvolution also works if the sampling distances are

not matched rigorously, but matching them improves the chances to get good results. In general, over-sampling the object is less harmful than undersampling it, with one exception: In the case of confocal data, the sampling distance d_{xy} should not be much smaller than indicated, if the blind deconvolution algorithm or the non-blind deconvolution algorithm together with a theoretically computed confocal PSF are used. Otherwise, the unconstrained Maximum Likelihood algorithm and the predominant noise in the data might lead to unsatisfactory results.

Black Level and Saturation

Before grabbing images from the microscope's camera, the light level should be adjusted in such a way that saturated pixels, either black or white ones, are avoided. Saturated pixels are pixels that are clamped to either black or white because their actual intensity values are outside the range of representable intensities. In any case, saturation means a loss of information and thus prevents proper post-processing or deconvolution. At the same time, a high background level should be avoided because it decreases the dynamic range of the imaging system and the deconvolution works worse. This means that empty regions not showing any fluorescence should appear almost black. A background level close to zero is especially important when bead measurements are performed in order to extract an experimental point spread function. Details are discussed is a separate tutorial about *bead extraction*.

3.6.3 Standard Deconvolution Tutorial

This tutorial explains how 3D image data sets can be deconvolved in Avizo. It is asumed that the reader is already familiar with the basic concepts of Avizo itself. If this is not the case, it is strongly recommended to work through the *standard Avizo tutorials* first. In this section, the following topics are covered:

1. Prerequisites for deconvolution
2. Resampling a measured PSF
3. Deconvolving an image data set
4. Calculating a theoretical PSF

As an example we are going to use a confocal test data set (*polytrichum.am*) provided with the Avizo deconvolution modules. The data file is located in the directory AVIZO_ROOT/*data/deconv*.

- Load the data set *polytrichum.am*.
- Visualize it, for example, using a *Image Ortho Projections* module.

The data set shows four chloroplasts in a spore of the moss polytrichum commune.

Prerequisites for Deconvolution

Besides the image data itself, for the standard non-blind deconvolution algorithm a so-called *point spread function* (PSF) is also required. The PSF is the image of a single point source, or as a close approximation, the image of a single fluorescing sub-resolution sphere. PSF images can either be computed from theory (see below) or they can be obtained from measurements. In the latter case, tiny so-called *beads* are recorded under the same conditions as the actual object. This means that the same objective lens, the same dye and wavelength, and the same immersion medium are used. Typically, the images of multiple beads are averaged to obtain an estimate of a single PSF. Avizo provides a module called *Extract Point Spread Function* facilitating this process. The use of this module is discussed in a *separate tutorial* about bead extraction. At this point, let us simply load a measured PSF from a file.

- Load the data set *polytrichum-psf.am*.
- Use the *Image Ortho Projections* module to visualize it.

The PSF appears as a bright spot located in the middle of the image volume (Figure 3.41). It is important that the PSF is exactly centered. Otherwise, the deconvolved data set will be shifted with respect to the original image. Also, it is important that the PSF fades out to black at the boundaries. If this is not the case, the black level of the PSF image needs to be adjusted using the *Arithmetic* module. Finally, neither the PSF nor the image to be deconvolved should exhibit *intensity attenuation artifacts*, i.e., image slices with decreased average intensity due to excessive light absorption in other slices. If such artifacts are present, they can be removed using the *Correct Z Drop* module.

Resampling a Measured PSF

Next, select both the PSF and the image data. You'll notice that the voxel sizes of both objects are not the same. It is recommended to adjust different voxel sizes of PSF and image data prior to deconvolution using the *Resample* module. The deconvolution module itself also accounts for different voxel sizes, but it does so by using point sampling with trilinear interpolation. This is OK as long as the voxel size of the PSF is larger than that of the image data. However, in our case the voxel size of the PSF is smaller than that of the image data, i.e., the resolution of the PSF higher. Using the *Resample* module provides slightly more accurate results here, since all samples will be filtered correctly using a Lanczos kernel.

- Connect a *Resample* module to *polytrichum-psf.am*.
- Connect the *Reference* port of the *Resample* module to the image data set *polytrichum.am*
- In the *Mode* port of the *Resample* module, choose *voxel size* (see Figure 3.42).
- Resample the PSF by pressing the *Apply* button.

The *voxel size* option means that the PSF will be resampled on a grid with exactly the same voxel size as the image data set, which is connected to the *Reference* port. While the original PSF had a resolution of 12 x 12 x 30 voxels, the resampled one only has 12 x 12 x 16 voxels. However, the extent of a single voxel in z-direction is bigger now.
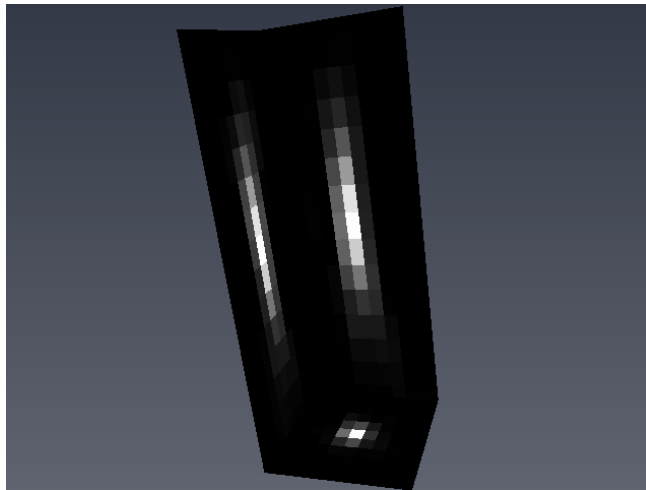
Figure 3.41: Maximum intensity projection of *polytrichum-psf.am*
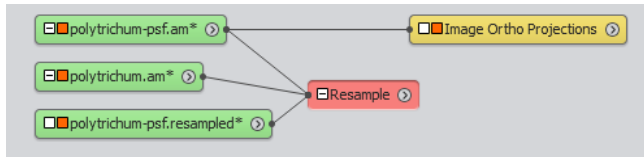
Figure 3.42: Maximum intensity projection of *polytrichum-psf.am*

Deconvolving an Image Data Set

After a suitable PSF has been obtained, we are ready for deconvolving the image data set. This can be done by attaching a *Deconvolution* module to the image data.

- Connect a *Deconvolution* module to *polytrichum.am*.
- Connect the *PSF Kernel* port of the *Deconvolution* module to the resampled PSF *polytrichum-psf.Resampled*. The *PSF Kernel* port is in the *Microscope* section of the module's properties.

Once the deconvolution module is connected to its two input objects, some additional parameters need to be adjusted (for a detailed discussion of these parameters see also the *Deconvolution* reference documentation of the *Deconvolution* module itself). Figure 3.43 shows the project view. The settings for the deconvolution module are :

Border width: For deconvolution the image data has to be enlarged by a guardband region. Otherwise, boundary artifacts can occur, i.e., information from one side of the data can be passed to the other. There is no need to make the border bigger than the size of the PSF. However, if the data set is dark at the boundaries, a smaller border width is sufficient. In our case, let us choose the border values 0, 0, and 8 in the x, y, and z direction.

Iterations: The number of iterations of the deconvolution algorithm. Let us choose a value of 20 here.

Initial estimate: Specifies the initial estimate of the deconvolution algorithm. If *const* is chosen, a constant image is used initially. This is the most robust choice, yielding good results even if the input data is very noisy. We keep this option here.

Overrelaxation: Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. In most cases the best compromise between speed and quality is *fixed* overrelaxation. Therefore, we keep this choice also.

Method: Selects between standard (non-blind) and blind deconvolution. Let us specify the *standard* option here.
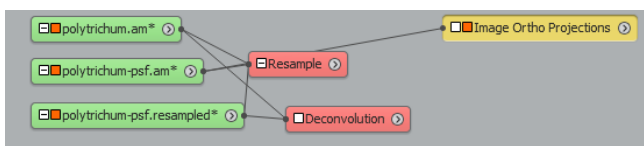
Figure 3.43: Deconvolution module attached to *polytrichum.am*.

The actual deconvolution process is started by pressing the *Apply* button. Please press this button now. The deconvolution should take about 10 seconds on a modern computer. During the deconvolution, the progress bar informs you about the status of the operation. Also, after every iteration, a message is printed in the Avizo console window indicating the amount of change of the data. If the change seems to be small enough, you can terminate the deconvolution procedure by pressing the *Stop* button. However, note that the stop button is evaluated only once between two consecutive iterations.

When deconvolution is finished, a new data set called *polytrichum.deconv* appears in the Project View. You might take a look at the deconvolved data by moving the *Image Ortho Projections* connection line from *polytrichum.am* to *polytrichum.deconv*.

Calculating a Theoretical PSF

Sometimes bead measurements are difficult to perform, so that an experimental PSF cannot easily be obtained. In such cases, a theoretical PSF can be used instead. Avizo provides the module *Generate Point Spread Function*, allowing you to calculate theoretical PSFs. The module can be created by selecting *Images And Fields / Generate Point Spread Function* from the *Project >Create Object...* menu of the Avizo main window.

Once the module is created, some parameters have to be entered again. The *resolution* and the *voxel size* can be most easily specified by connecting the *Data* port of the *PSGGen* module to the image data set to be convolved. In our case, please connect this port to *polytrichum.am*.

In order to generate a PSF, you also need to know the numerical aperture of the microscope objective, the wavelength of the emitted light (to be entered in micrometers!), and the refractive index of the immersion medium. In our test example, these values are NA=1.4, lambda=0.58, and n=1.516 (oil medium). Also, change the microscopic mode from widefield to confocal.

After you press the *Apply* button, the computed PSF appears as an icon labeled *PSF* in the Project View. You can compare the theoretical PSF with the measured one using the *Ortho Slice* module. You'll notice that the measured PSF appears to be slightly wider. This is a common observation in many experiments.
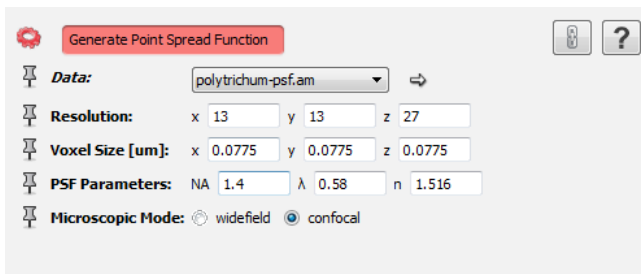
Figure 3.44: The Generate Point Spread Function module calculates theoretical PSFs.

Once you have computed a theoretical PSF, you can perform non-blind deconvolution as described above. However, for convenience the *Deconvolution* module is also able to compute a theoretical PSF by itself. You can check this by disconnecting the *Kernel* port of the *Deconvolution* module. If no input is present at this port, additional input fields are shown, allowing you to enter the same parameters (numerical aperture, wavelength, refractive index, and microscopic mode) as in *PSFGen*. After these parameters have been entered, the deconvolution process again can be started by pressing the *Apply* button.

Note that any previous result connected to the *Deconvolution* module will be overwritten when starting the deconvolution process again. Therefore, be sure to disconnect a previous result if you want to compare deconvolution with different input PSFs.

3.6.4 Blind Deconvolution Tutorial

This tutorial explains how blind deconvolution can be perfomed in Avizo. At the same time it describes how deconvolution jobs can be processed using the Avizo job queue. Like in the previous tutorial, it is assumed that the reader is already familiar with the basic concepts of Avizo itself. If not, we recommend to work through the *standard Avizo tutorials* first.

A Blind Deconvolution Example

Let us start by loading a raw image data set first.

- Load the file *alphalobe.am* from the directory *data/deconv*.
- Visualize the data set by attaching a *Image Ortho Projections* module to it.

The data set has been recorded using a standard fluorescence microscope under so-called *widefield* conditions. It shows a neuron from the alpha-lobe of the honeybee brain. Compared to the confocal data set used in the standard deconvolution tutorial, *alphalobe.am* is much bigger. It has a resolution of 248 x 248 x 256 voxels with a uniform voxel size of 1 micrometer. In the xy-plane of the projection view the structure of the neuron can be clearly identified. However, the contrast of the image is quite poor because there is a significant amount of out-of-focus light or haze present. With Avizo's blind deconvolution algorithm we can enhance the image data without needing to know an explicit PSF in advance.

- Attach a *Deconvolution* module to *alphalobe.am*.
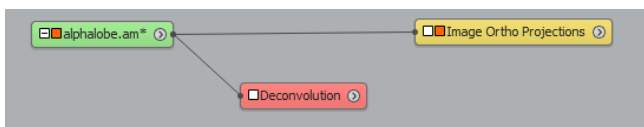
Figure 3.45: Alphalobe Deconvolution

Adjust the Deconvolution parameters like this :

Parameters:

- Border Width: x = 8 y = 8 z = 10
- Iterations: 25
- Initial Estimate: input data
- Method: blind

Microscope:

- PSF: NA = 0.5 λ = 0.58 n = 1

The individual parameters have the following meaning:

Border width: The same as for standard non-blind deconvolution, the image data must be enlarged by a guardband region. Otherwise, boundary artifacts can occur, i.e., information from one side of the data can be passed to the other. For this data set we specify a small guardband region of 8 voxels in the x- and y-direction. In the z-direction we add 10 slices as border region. The resulting size of the data array on which the computation is performed is 256 x 256 x 266.

Note that for dimensions with a power of two (2^8), the Fast Fourier Transforms, the computationally most expensive part of the deconvolution algorithm, can be executed somewhat faster.

Iterations: We choose a value of 25 here. Depending on the data, usually at least 10 iterations are required. With overrelaxation being enabled (see below), results usually don't improve much after 40 iterations.

Initial estimate: Specifies the initial estimate of the deconvolution algorithm. Since there is not much noise present in the original alphalobe images, it is safe to chose *input data* here. This causes the algorithm to converge even faster.

Overrelaxation: Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. We enable overrelaxation by chosing the *fixed* toggle.

Regularization: We chose *none* here in order to do no regularization.

Method: We chose *blind* here in order to select the blind deconvolution algorithm.

PSF Parameters: For *alphalobe.am*, the numerical aperture is 0.5, the wavelength is 0.58 micrometers, and the refractive index is 1.0 (air). These parameters are required in order to apply certain constraints to the estimated point spread function. They are also used in order to compute an initial PSF. The convergence is sensitive to the initial parameters. If a data set would be connected to the *Kernel* port of the deconvolution module, this data set would be used as the inital PSF with the given PSF parameters still acting as constraints. For example, you could provide a measured PSF and let it be fitted to the actual data by the deconvolution algorithm.

Microscopic mode: alphalobe.am is a widefield data set, so select this option here.

Submitting a Deconvolution Job

After all parameters have been entered, the deconvolution process can be started. This computation can take some time. Especially, if you want to deconvolve multiple data sets at once, it is inconvenient to do this in an interactive session. Therefore, multiple deconvolution jobs can be submitted to the Avizo job queue and then, for example, processed overnight. This works as follows:

- Press the *Setup...* button of the *Batch Job* port. A dialog as shown in Figure 3.46 pops up.
- In the dialog choose a file name under which you want to save the deconvolved data set, e.g., `C:/Temp/alphalobe-deconv.am`.
- Modify the text field, so that check point files are written after every 5 iterations.
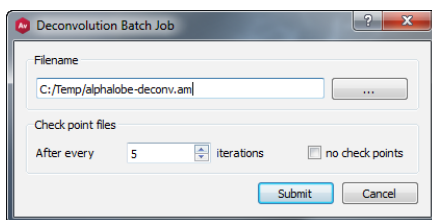
Figure 3.46: Dialog for submitting a deconvolution job.

Check point files are used to store intermediate results. With the above settings, the deconvolved data is written into a file after every 5 iterations. Check point files are named like the final result, but a consecutive number is inserted just before the file name suffix. For example, if the result file name is `C:/Temp/alphalobe-deconv.am`, the check point files are named `C:/Temp/alphalobe-deconv-0005.am`, `C:/Temp/alphalobe-deconv-0010.am` and so on. Now we are ready to actually submit the batch job.

- Press the *Submit* button of the deconvolution dialog. After a few seconds the Avizo batch job dialog appears, compare Figure 3.47.
- Select the deconvolution job and press the *Start* button.

You now have to wait about 20 minutes until the deconvolution job is finished. Once the job queue has been started, you can quit Avizo. The batch jobs will be continued automatically. If Avizo is still running when the deconvolution job exits, then the result will be loaded automatically in Avizo. Otherwise, you have to restart Avizo and load the deconvolved data set manually.

3.6.5 Bead Extraction Tutorial

Non-blind deconvolution is a powerful and robust method for enhancing the quality of 3D microscopic images. However, the method requires that the image of the point spread function (PSF) responsible
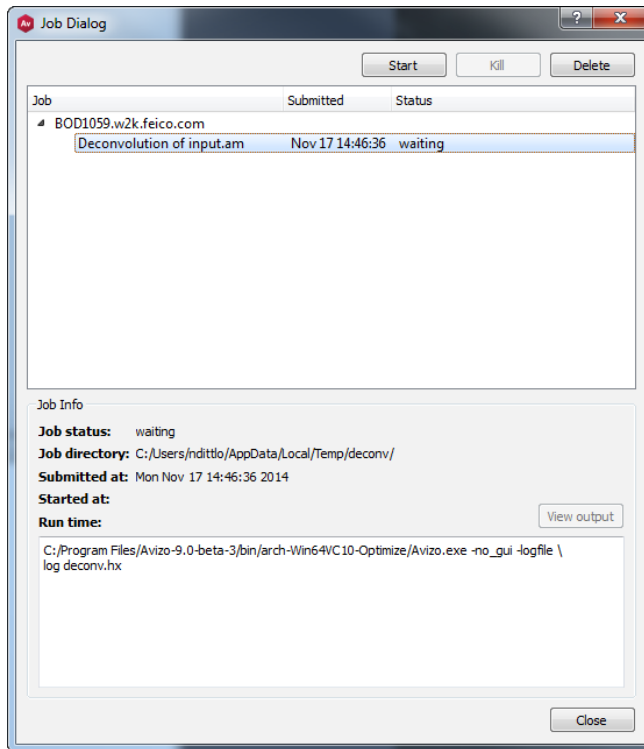
Figure 3.47: The Avizo job dialog showing a pending deconvolution job.

for image blurring is provided. As stated in the *standard deconvolution tutorial*, the PSF can either be calculated theoretically or it can be obtained from a bead measurement. Avizo provides a special-purpose module called *Extract Point Spread Function* which facilitates the extraction of PSF images from one or multiple bead mesurements. In this tutorial, the use of the module shall be explained. The following topics are covered:

1. Bead measurements
2. Projection Settings View and Projection Settings View Cursor
3. Resampling and averaging the beads

Bead Measurements

The PSF is the image of a single point source recorded under the same conditions as the actual specimen. It can be approximated by the image of a fluorescing sub-resolution microsphere, a so-called bead. Performing good bead measurements requires some practice and expertise. In order to obtain good results, the following hints should be obeyed:

1. Use appropriate beads. It is important that the bead size be smaller than approximately $1/2$ full width at half maximum (FWHM) of the PSF. Good sources for obtaining beads suitable for PSF measurements are Molecular Probes (http://www.probes.com/) or Polysciences (http://www.polysciences.com/).
2. The beads must be solid. Besides solid beads, there are also beads with the shape of a spherical shell, allowing to check the focus plane of a mircoscope. Such beads cannot be used as a source for PSF generation in the current version of Avizo.
3. Don't record clusters of multiple beads. Sometimes multiple beads may glue together, appearing as a single big bright spot. Computing a PSF from such a spot obviously leads to wrong results.
4. Note that beads are not resistant to a variety of embedding media. In particular, beads will be destroyed in xylene-based embedding media such as Permount (Fisher Scientific) and methyl salicylate (frequently used to clear up the tissue). As a substitute, you might use immersion oil instead, which has a refractive index similar to methyl salicylate, for example.
5. Sample and beads should always be imaged as close to the coverslip as possible. When it is not possible to attach the sample to the coverslip, the beads should also be imaged in a comparable depth, embedded in the same mounting medium. Imaging the beads with better quality than the sample will yield a slightly blurred deconvolution result. However, when the PSF used for deconvolution is too wide, artifacts can arise during deconvolution.

   The objective lense should always be selected according to the mounting medium, i.e., if the sample is attached to the slide and embedded in a buffer of refractive index close to water, a severe loss of image quality can be expected when using an oil-immersion objective without a correction collar. Deconvolution of properly imaged data will allways be superior to deconvolution of data suffering from aberrations.
6. Problems occur if the mounting medium remains liquid. In that case, the sample distribution

may not be permanent. If your specimen is to be embedded in water, you can try to immerse the beads in an agarose gel of moderate concentration instead. Attaching the small beads to the coverslip (for example by letting them dry) is often also sufficient for immobilization.

An example of an image data set containing multiple beads is provided in the file *beads.am* in the directory AVIZO_ROOT/*data/deconv*.

- Load the data set *beads.am*.
- Visualize the data using a *Image Ortho Projections* module.

Projection View and Projection Settings View Cursor

The bead data set contains five different beads which can be clearly seen in the three orthogonal planes of the *Image Ortho Projections* module. In order to obtain a single PSF, we first want to select several "good" beads. These beads are then resampled and averaged, thus yielding the final PSF. A bead can be considered as "good" if it is clearly visible and if it is not superimposed by other beads (even when defocused),

Selecting "good" beads is an interactive process. It is most easily accomplished using the *Projection Cursor* module. This module allows you to select a point in 3D space by clicking on one of the three planes of the *Image Ortho Projections* module. The third coordinate is automatically set by looking for the voxel with the highest intensity. Points selected with the *Cursor* module can be stored in a *Landmarks* data object.

- Attach a *Projection Cursor* module to the *Image Ortho Projections* module.
- Click on any bead on one of the three planes.
- Store the current cursor position in a *Landmarks* object by pressing the *Add* button.
- Select and add some other beads too.

The landmarks do not need to be located exactly at the center of a bead. The exact center positions can be fitted automatically later on.

You can remove incorrect bead positions from the landmark set by invoking the landmark set editor. In order to activate the editor, select the landmark set object and press Landmark Editor button. If you want to add additional bead positions to an existing landmark set object, make sure that the *master port* of the landmark set object is connected to the *Cursor* module. Otherwise, a new landmark set object will be created.

Resampling and Averaging the Beads

Now we are ready to extract and average the individual beads. This is done by means of the *Extract Point Spread Function* module.

- Connect an *Extract Point Spread Function* module to the *Landmarks* object.
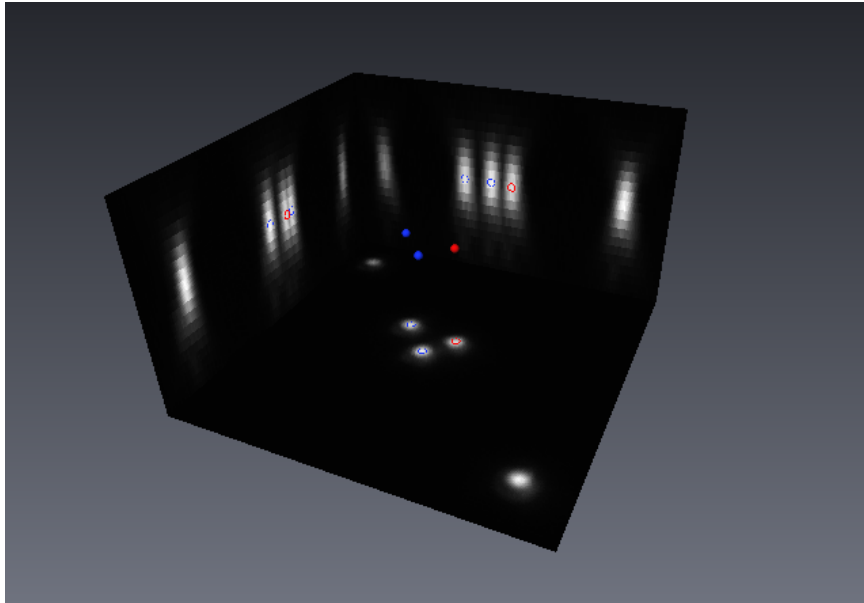
Figure 3.48: Individual beads can be interactively identified using a *Projection Cursor* module.

- Make sure that the *Data* port of *Extract Point Spread Function* is connected to the bead data set *beads.am*. If the landmarks are still connected to the beads via the *Projection Cursor* and *Image Ortho Projections* modules, the connection is established automatically.

The *Extract Point Spread Function* module provides two buttons called *Adjust centers* and *Estimate size*, which should be invoked in a preprocessing step before the beads are actually extracted.

The first button (*Adjust centers*) modifies the landmark positions so that they are precisely located in the center of gravity of the individual beads.

The second button (*Estimate size*) computes an estimate for the number of voxels of the PSF image to be generated. This button is only active if no PSF image is connected as a result to *BeadExtract*. If there is a result object, the resolution and voxel sizes of the result are used and the port becomes insensitive.

Any of the actions of the two preprocessing buttons can be undone using the *Undo* button. This can be necessary for example if two beads are too close so that no correct center position could be computed.
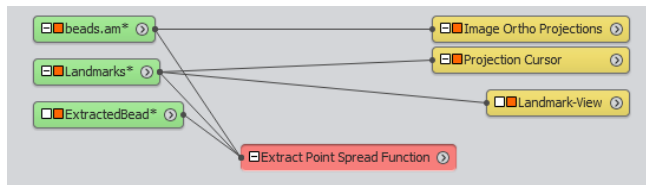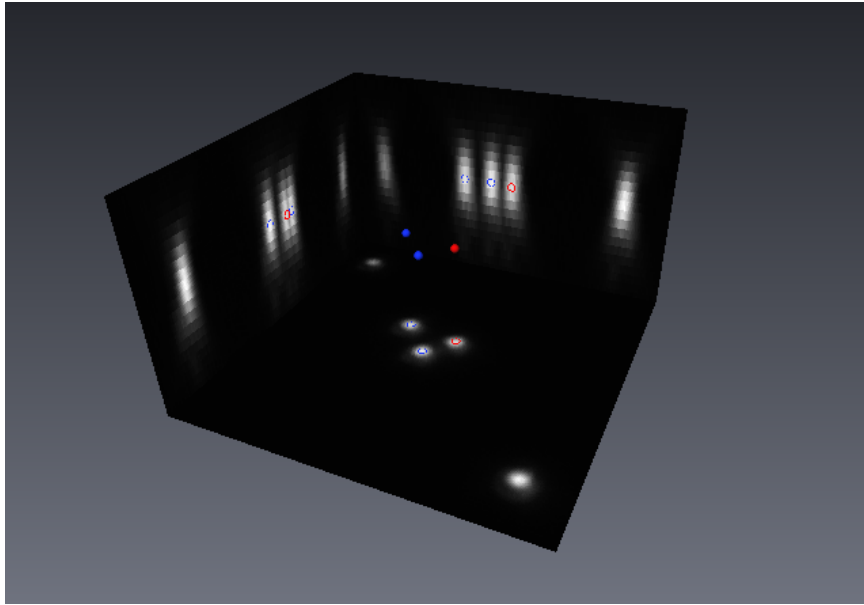
Figure 3.49: The *Extract Point Spread Function* module resamples and averages multiple beads.

In general, overlaps between neighboring beads should be avoided. Small overlaps might be tolerated because during resampling intensities are weighted according to the influence of surrounding beads.

- Perform the preprocessing steps *Adjust centers* and *Estimate size*.
- Compute a resampled and averaged PSF by pressing the *Apply* button.

The data type of the resulting PSF will be *float*, irrespective of the data type of the input image. The individual beads will be weighted on a per-voxel basis and added to the result. No normalization will be performed afterwards. You may investigate the resulting PSF image using any of the standard visualization modules. In Figure 3.50 a volume rendering of the resulting PSF is shown.

In some cases, you may want to average multiple beads recorded in different input data sets. This can be easily achieved by creating a *Landmarks* object for each input data set. For the first input data set, extract the beads as described above. For the other input data set, also use the *Extract Point Spread*
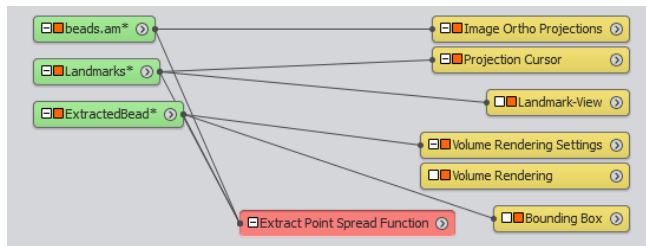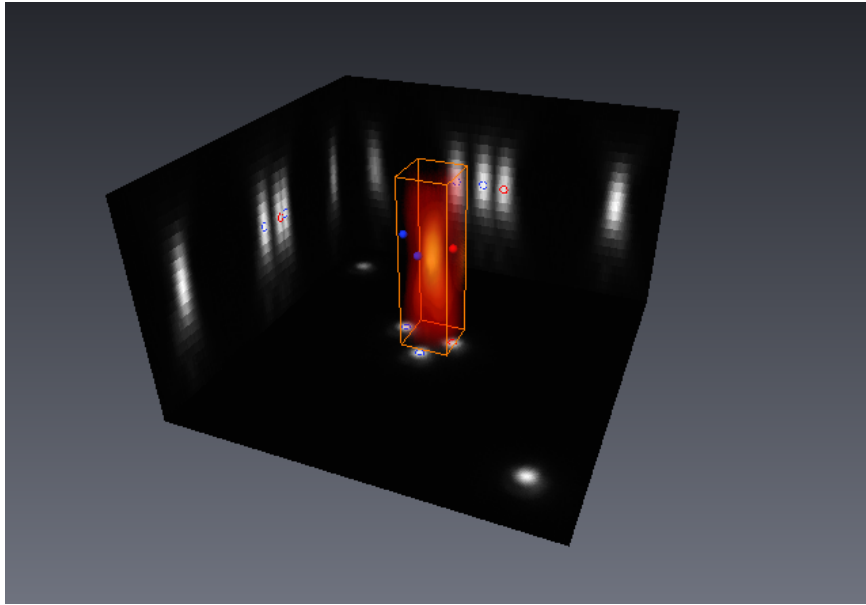
Figure 3.50: The final PSF visualized using a *Volume Render* module.

Function module. However, make sure that the PSF obtained from the first input data set is connected as a result object to *BeadExtract* before pressing the *Apply* button. In order to use an existing PSF as a result object, connect the *Master* port of the PSF to *Extract Point Spread Function* (once this is done the *Resolution* and *Voxel size* ports of *Extract Point Spread Function* become insensitive, see above). If an existing result is used, new beads simply will be added into the existing data set. Therefore, data sets should be scaled in intensity according to their quality prior to bead extraction and summation to obtain a suitable weighting of the individual extracted beads in the final result.

3.6.6 Performance issues and multi-processing

Iterative maximum-likelihood deconvolution essentially is the most powerful and most robust technique for the restoration of 3D optical sections. However, it is also computationally very demanding. It can take several minutes (sometime even hours) to process large 3D data sets. This is not due to an improper implementation, but due to the algorithm itself. Both the blind and the non-blind variant of the method rely heavily on fast Fourier-transforms in order to efficiently compute convolutions. If you want to improve performance, try to adjust the size of your data volumes so that the number of voxels plus the border width is a power of two. Sometimes it is worth it to enlarge the border width a little bit in order to get a power of two. Although the algorithm works with data of any size, powers of two can be transformed somewhat faster.

Another issue is memory consumption. Internally, several copies of the data set need to be allocated by the deconvolution algorithm. These copies should all fit into memory at the same time (a specific variant of the algorithm suitable for working under low memory conditions will be provided in a later version). Besides the input data itself, the following number of working arrays are required by the different methods:

- 3 working arrays for the non-blind algorithm with no or with fixed overrelaxation
- 5 working arrays for the non-blind algorithm with optimized overrelaxation
- 5 working arrays for the blind algorithm

The number of voxels of a working array is the product of the number of voxels of the input data set plus the border with along each spatial dimension. The primitive data type of a working array is a 4-byte floating point number. For example, if the number of voxels of the input data set plus the border width is 256 x 256 x 256 (as for the *alphalobe.am* data set in the blind deconvolution tutorial), each working array will be about 64 MB, irrespective of the primitive data type of the input data set. Therefore, at least 192 MB (3x4x256x256x256 bytes) are required for non-blind deconvolution with fixed overrelaxation, and 320 MB (5x4x256x256x256 bytes) for blind deconvolution. Keep this in mind when configuring the computer on which to perform deconvolution! However, also note that for most platforms it usually doesn't make sense to have more than 1.5 GB of main memory. For more memory, a 64-bit operating system is required.

Finally, it should be mentioned that the deconvolution algorithm can make use of a multi-processor CPU board. Although you do not get twice the performance on a dual-processor PC, a speed-up of almost 1.5 can be achieved. By default, Avizo uses as many processors as there are on the computer. If for some reason you want to use less processors, you can set the environment variable AVIZO_DECONV_NUM_THREADS to the number of processors you actually want to use simultaneously.

Example 1: Confocal Data

The original data set is provided under AVIZO_ROOT*/data/deconv/polytrichum.am*. The images below were created using the *Image Ortho Projections* module.

The properties of the data set are as follows:

- Numerical aperture 1.4
- Wavelength of the emitted light 0.58 micrometers
- Refractive index 1.516 (oil)
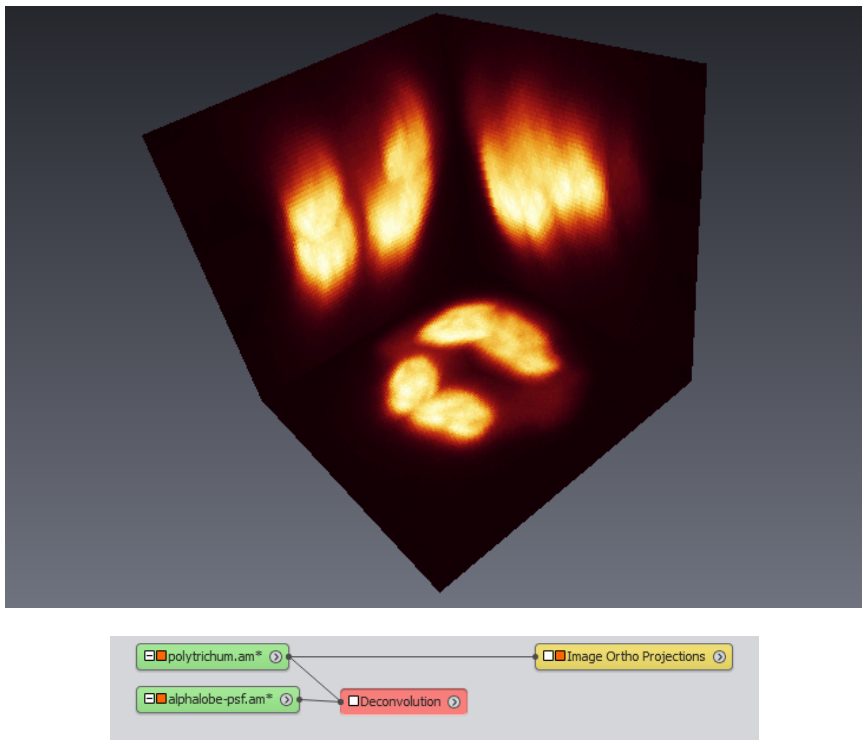
Figure 3.51: *polytrichum.am* before deconvolution.

Example 2: Widefield Data

The original data set is provided under `AVIZO_ROOT`/*data/deconv/alphalobe.am*. The images below were created using the *Image Ortho Projections* module.

The properties of the data set are as follows:

- Numerical aperture 0.5
- Wavelength of the emitted light 0.58 micrometers
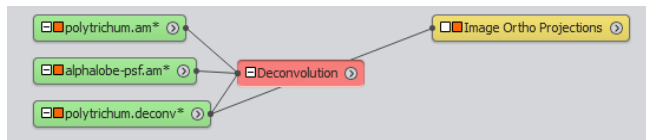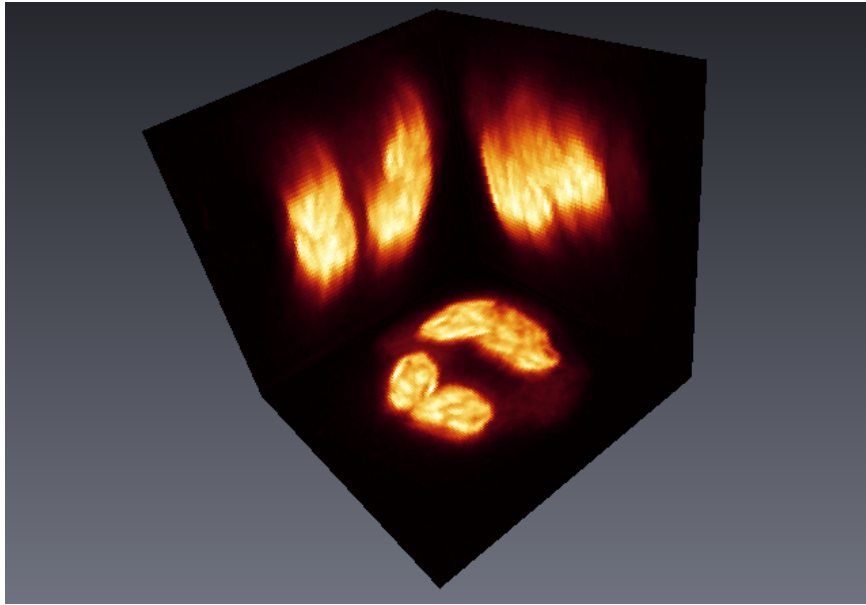
Figure 3.52: *polytrichum.am* after deconvolution.

- Refractive index 1.0 (air)
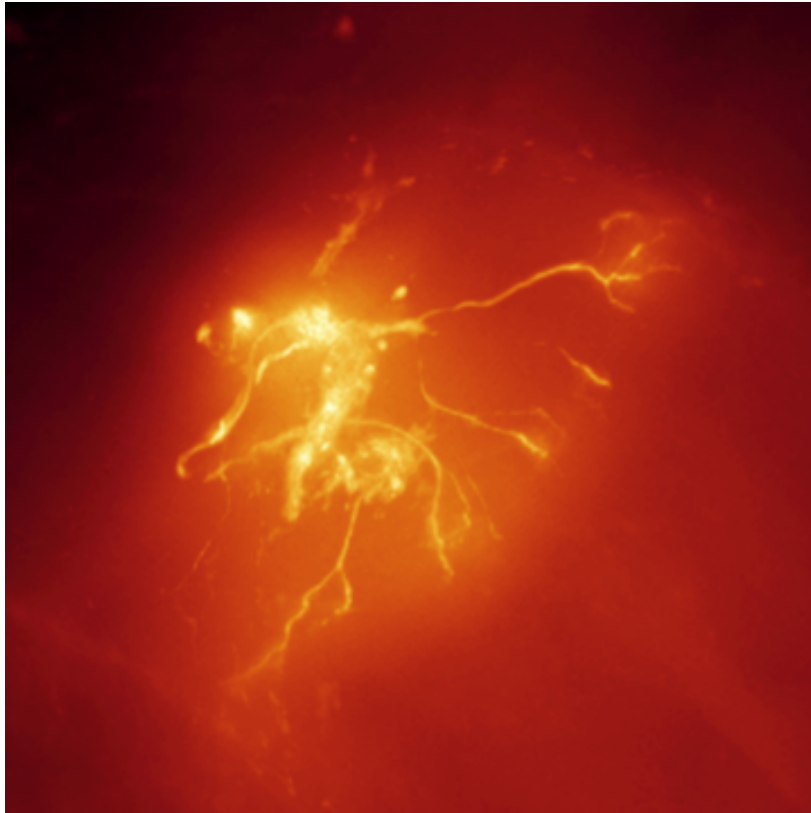
Figure 3.53: XY-maximum intensity projection of *alphalobe.am* before deconvolution.
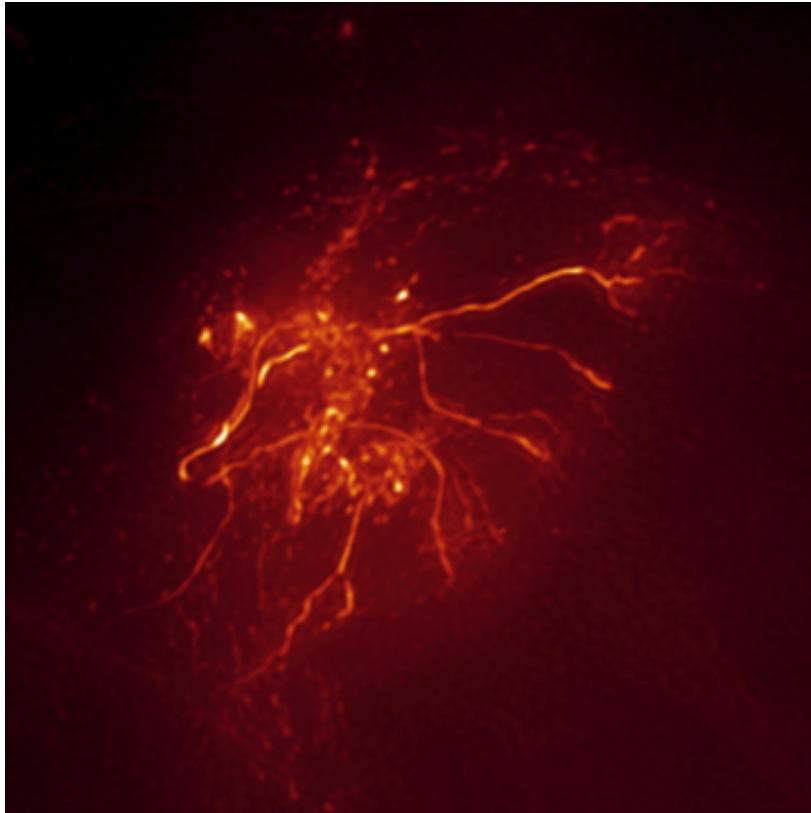
Figure 3.54: XY-maximum intensity projection of *alphalobe.am* after deconvolution.

Chapter 4

Tutorials: Advanced Image Processing, Segmentation and Analysis

The following tutorials require an Avizo license (Avizo Lite Edition license is not sufficient).

- *Getting Started with Advanced Image Processing and Quantitative Analysis* - the basics for using Avizo for image processing and analysis
- Examples:
    - *Measuring a Catalyst* - Masking and Distance Map
    - *Separating, Measuring and Reconstructing* - Watershed Separation
    - *Distribution of Pore Diameters in Foam* - Custom Measurement
    - *Average Thickness of Material in Foam* - Separation Thickness
- *Watershed Segmentation* - advanced Segmentation
- *More about Image Filtering* - reduce image noise or artifacts or enhance features of interest
- *More about label measures* - manage several groups of measures

4.1 Getting Started with Advanced Image Processing and Quantitative Analysis

In this step-by-step tutorial, you will learn the basics for using Avizo for image processing and analysis. The example used below can be easily extended to other applications and follows a typical workflow

of image analysis:

1. Image enhancement
2. Features extraction
3. Data measures and analysis

This section has the following parts:

- *Processing grayscale images*
- *3D versus 2D stack interpretation*
- *Binarization of grayscale images*
- *Separation*
- *Analysis and measures*
- *Interactive selection*
- *Measure filters*
- *Sieves*
- *Label images*
- *Processing images in memory (in-core) and on disk (out-of-core)*
- *Scripting*

You should be familiar with the basic concepts of Avizo to follow this tutorial. In particular, you should be able to load files, to interact with the 3D viewer, and to connect modules to data modules. All these issues are discussed in Avizo chapter 2 - Getting started. For routine use of Avizo you may benefit from familiarity with Avizo image filtering and segmentation (see chapter 3 - Images and volumes visualization and processing).

4.1.1   Processing images

First of all, you have to :

- Load in Avizo the 3D microtomography volume `data/images/foam/foam.am` from the `AVIZO_ROOT` directory. The data object appears in the Project View.
- If Auto-Display is disabled, attach an *Ortho Slice* module to visualize the data. To do so, right-click on the green data icon. In the object popup, choose the category entry called *Display*, then double-click on the entry *Ortho Slice* (or click on *Create*). You can also use the search field and type the first letters of *Ortho Slice*, then selecting the module in the list will automatically create it in the Project View.
- Attach two other *Ortho Slice* modules. In the Properties panel of the second *Ortho Slice*, select the *xz* orientation and the *yz* orientation in the Properties panel of the third *Ortho Slice*. See the resulting display on Figure 4.1.

Figure 4.1: Initial data

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-1-LoadData.hx will complete the steps above.

Improving image quality is often necessary to obtain the best results with image analysis. The next step illustrates how to process images in Avizo with an image filter commonly used for smoothing or noise reduction. You can learn more about image filters in tutorial section 4.7 - More about image filtering.

- Attach a *Median Filter* module to the data. (Use the search field and type the first letters of *Median Filter*, then select the module in the list.)
- Select 3D in the *Interpretation* port of *Median Filter*.
- Press on the *Apply* button.

Once computed, the result is stored in a new image object foam.filtered (see Figure 4.2).

- Attach the three *Ortho Slice* to the resulting image. To do so, change the *Data* of each slice in the Properties panel. See the resulting display on Figure 4.3.

Figure 4.2: Median Filter network

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-2-ImageProcessing.hx will complete the steps above.
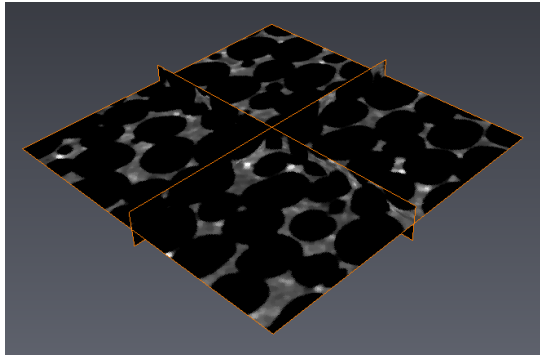
Figure 4.3: After removing noise with a median filter.

4.1.2   Interpretation as 3D image or 2D image stack

Sometimes it can be useful to interpret the input data of an image processing algorithm as a 3D volume, or as a sequence of 2D planes. For instance, a number of image filters and image processing algorithms can be performed either on each XY-slice of the volume using a 2D kernel or on the whole volume using a 3D kernel. In some cases, it may be preferred to use 2D algorithm, either for performance or for more appropriate effect depending on the data and the desired outcome.

In many Avizo modules, an interpretation port shows the state of the current module (i.e, *XY planes* or *3D*). If the state of the port is *XY planes*, it means that the module will perform on each XY-slice. If the state of the port is *3D*, it means that the module will perform on the entire three-dimensional image at once.

In some cases, the intepretation port cannot be changed (it is grayed), for instance, when processing can only be applied in XY planes.

4.1.3   Getting more help

Press the question mark button ![?] in the Properties panel of a module to display the module help. This help may be contextual depending on interpretation mode, or type of processing selected in the module.

4.1.4   Binarization

Binarization means transforming a grayscale image into a binary image (i.e., a label image with only interior and exterior materials). Threshold binarization is used when the relevant information in the grayscale image corresponds to a specific gray level interval. Thresholding is a simple *segmentation* method - more sophisticated automatic, semi-automatic or manual segmentation tools are also avail-

able in Avizo. Threshold binarization can be done with the *Interactive Thresholding* module which prompts you to set the levels with a visual feedback.

- Attach an *Interactive Thresholding* module to the filtered data.

You can interactively modify the threshold values with immediate 2D or 3D visual feedback. The selected pixels appear in blue in the displayed image.

- In the Properties panel of the module, you can set the *Intensity range* port to the range 0-30 for instance.
- Check and uncheck the *3D* option in the *Preview Type* port to get a 2D only or 3D preview (see Figure 4.4).
- Press the *Apply* button to start the module.

The output binary image named `foam.thresholded` is generated in the Project View (see Figure 4.5).
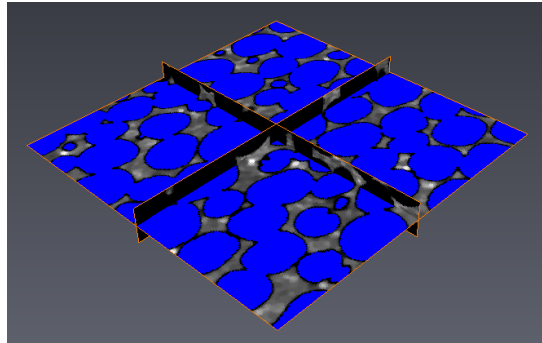
Figure 4.4: *Interactive Thresholding* 2D preview

In the output binary image, all pixels with an initial gray level value lying between the two bounds are set to 1, all the other pixels are set to 0.
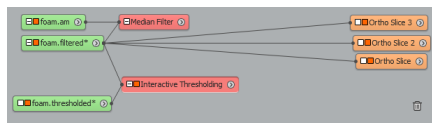
Figure 4.5: *Interactive Thresholding* network

The *Interactive Thresholding* module has created a binary image. For binary images, the Avizo displays the voxels of intensity 1 with a blue color. If you attach an Ortho Slice to the resulting image, an

appropriate colormap is selected by default.

- Hide the *Interactive Thresholding* preview by switching of the orange visibility button in the module icon in the Project View or next to the module name in the Properties panel.
- Connect the first *Ortho Slice* to the thresholded data. See the resulting display on Figure 4.6.
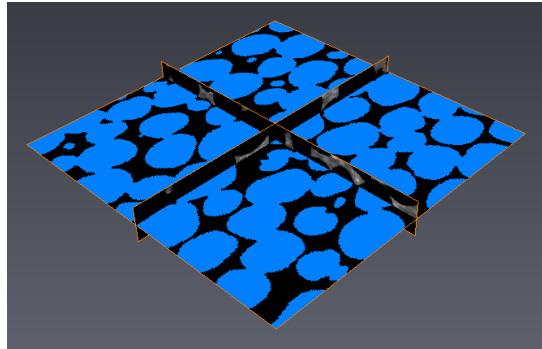
Figure 4.6: Binary image displayed with an *Ortho Slice*

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-3-Binarization.hx will complete the steps above.

4.1.5 More about binary images

In a binary image, all pixels that meet some set of conditions (here the condition is pixel intensity within the two bounds set on *Interactive Thresholding*) are set to a value of 1 (the pixels of interest), and all other pixels are set to 0 (the background). There is no particular type for binary images in Avizo binary images are simply label images with only one label (8/16/32-bit per voxel with value 1; exterior with value 0). However, some of Avizo modules may explicitly require binary image as input data.

4.1.6 More hints about binarization

(This section is optional and is not required reading for completing the subsequent tutorials.)

Extensive tools are available in Avizo for effective data binarization and segmentation of images. The process can be automated in many cases, possibly by combining a number of steps, sometimes requiring user input. In some cases, it may be necessary or easier to proceed to semi-automatic or manual segmentation: in particular, the *Segmentation Editor* is designed for that purpose (note that

the *Segmentation Editor* only supports 8-bit label images). Also keep in mind that improving image acquisition might be much easier than analyzing bad images.

Here are some of the Avizo modules that can facilitate automated binarization:

- *Auto Thresholding* automatically computes a threshold. You can choose the criterion best suited to your data, generally *factorisation*.
- *Interactive Top-Hat* is a powerful tool for segmenting areas with non uniform backgrounds, when simple thresholding fails to capture wanted features without unwanted noise. A top-hat transform can be seen as a "local thresholding". Top-hat results are often combined with threshold results using logical operation *OR Image*.
- *Hysteresis Thresholding* is used to achieve intermediate binarization between low and high thresholds defining a safe retained area and a rejected area. You may use, for instance, *Interactive Thresholding* interface to interactively choose the thresholds before using *Hysteresis Thresholding*. See also the *Canny Edge Detector* module.
- Many image filters like gradient or Laplacian can be used to help binarization, e.g., for *Edge Detection*.
- Filtering regions based on measures, as shown later in this tutorial, can also be a powerful technique for segmentation.

After binarization, it may be necessary to separate some objects, as shown in the next section.

4.1.7   Separation

In the example data set, some of the pores in the foam appear to be touching, but ideally, should be separated for proper analysis. Thresholding cannot avoid this type of output when the acquisition data is too coarse or noisy, because the gray levels of the considered objects are not uniform enough across the volume, or because the resolution is too low to distinguish some objects' boundaries. You can use the *Separate Objects* module to separate connected particles.

- Attach a *Separate Objects* module to the thresholded data.
- Set the *Marker Extent* port value to 1 instead of the default value 4. This is a contrast factor that controls the size of seeds marking objects to be separated. Increasing this value can merge some markers and therefore decrease the number of separated objects.
- Then press the *Apply* button. A `foam.separate` data is generated in the Project View.
- Attach the first *Ortho Slice* to the new data. See the resulting display on Figure 4.7.

The principle of the *Separate Objects* module is to compute watershed lines on a distance map. The *Separate Objects* module is a high-level combination of the watershed, distance map and *H-Maxima*. It can be used as a simple and straightforward separation tool, satisfying in many cases. You may notice, however, that some separation may be missing or unwanted, in particular with non-convex shapes (considering also 3D). For more details and advanced separation, see *Example 2: Separating,*

Measuring, and Reconstructing Individual Objects - Pores in Foam.
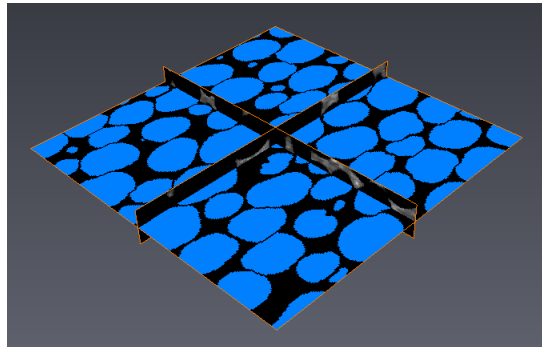
Figure 4.7: The particles are now separated

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-4-Separation.hx will complete the steps above.

4.1.8 Analysis

You can then use an analyze module to get the volume, surface area, mean value, number of voxels, etc., individually for each separated particle. This analysis on the stack of images is undertaken by using the *Label Analysis* module to extract statistical and numerical information, including the measure of objects.

- Attach a *Label Analysis* module to the separated data.
- Set foam.am as *Intensity Image* in the dedicated port of the module.
- Press the *Apply* button.

A new *label* image data object foam.label is created in the Project View, and the Tables panel is displayed, showing a spreadsheet-style table of results: the analysis *foam.Label-Analysis*, also created in the Project View (see Figures 4.8 and 4.9).
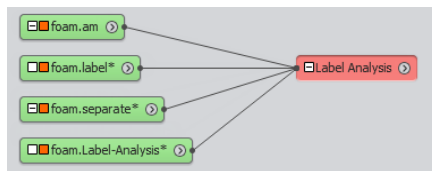
Figure 4.8: Analysis network

Figure 4.9: Analysis spreadsheets

The toolbar offers the possibility to:

- copy parts of the table ![icon]
- export the spreadsheet in several formats ![icon]
- sort columns in ascending or descending order ↑ ↓
- plot the histogram corresponding to a measurement ![icon] (see below)
- do label seek ⊕ (see below section 4.1.9 - Interactive selection).

The *basic* measures (as selected in the *Measures* port of the module) are displayed in the Tables panel as shown below (Volume3d, Area3D, etc.).

- Select the Volume3d column in the lower spreadsheet.
- Press on the histogram button of the toolbar (see Figure 4.10).

Figure 4.10: Analysis Panel toolbar: the histogram button

A window opens, displaying the Volume3d histogram as shown on Figure 4.11.

In the *Measures* port, *basic* is a group of measures that contains the most commonly used measures: Volume3d, Area3d, BaryCenterX, BaryCenterY, BaryCenterZ and Mean. It is possible to define new groups of measures, composed of pre-defined measures but also of user-defined measures. To learn
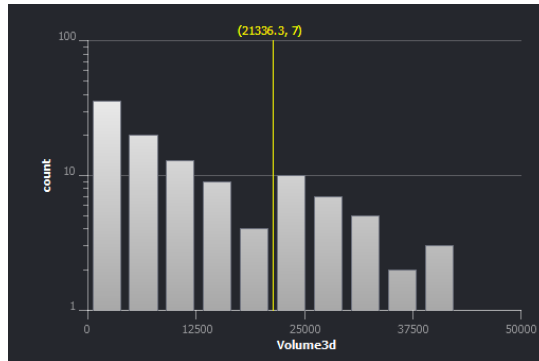
Figure 4.11: Volume3d histogram

more on the subject, please refer to the dedicated tutorial in chapter 4.4 - Further Image Analysis.

Note: By default, if the unit management of Avizo is disabled, the results are displayed with "pixel" or "voxel" as the unit. You can enable the unit management of Avizo to display data and measurements with units. Please refer to chapter 8.2.9 - Units in Avizo for all the details on how to use the unit management in Avizo.

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-5-Analysis.hx will complete the steps above.

4.1.9 Interactive selection

Avizo allows you to link the images in the 3D viewer to their corresponding rows in the analysis panel in order to locate individual objects with corresponding measures.

Figure 4.12: Analysis Panel toolbar: the label seek button

- Click on the label seek button of the analysis panel toolbar (see Figure 4.12). A new *Ortho Slice* is automatically attached to the separated image and displayed in the 3D viewer, as well as a point dragger (see Figure 4.13).
- Select a cell of the analysis lower table. The point dragger will move to the corresponding object location in the 3D view. If the histogram of one of the analysis measures is displayed, a vertical line appears that displays the position and value of the selected row, as shown on Figure 4.14.
- In the viewer window, you can move the dragger using the rectangular handles, then upon button release, the analysis table will highlight the corresponding object row. In order to move the

dragger, you must set the viewer into interaction mode (press the ESC key). Then move the mouse over one of the dragger's crosshairs and press the left mouse button. The color of the picked crosshair changes. The movement of the dragger is restricted to the corresponding plane.

- You can also click with the middle mouse button over a pickable object in the scene displayed in the 3D viewer, for instance, over a particular pore on a displayed slice: the dragger will move to the picked point and the corresponding spreadsheet row will also be highlighted.
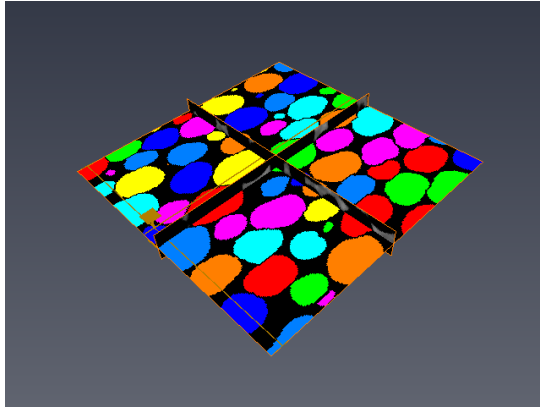
Figure 4.13: Point dragger (brown lines crossing) in the 3D viewer corresponding to row 82
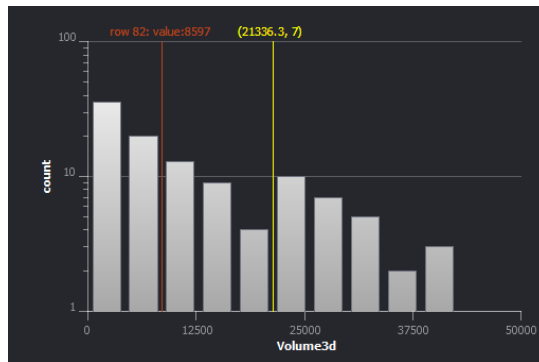
Figure 4.14: Line in the histrogram corresponding to row 82

4.1.10 Filtering based on measures

You can filter particles displayed in your 3D viewer. For example, you can decide to visualize only particles which Volume3d belongs to a specified range.

- Attach an *Analysis Filter* to `foam.Label-Analysis`.
- Connect the *Image* port to `foam.label`.
- Create a new filter by entering `Volume3d >= 30000` in the *Filter* port. To insert Volume3d in the formula you can type it or double-click on it in the list displayed below the formula field.
- Press on the *Apply* button.

This creates a new analysis with fewer objects.

- You can verify that fewer objects have been created by connecting the *Ortho Slice* to the new label image `foam.label-filtering`, as shown on Figure 4.16.
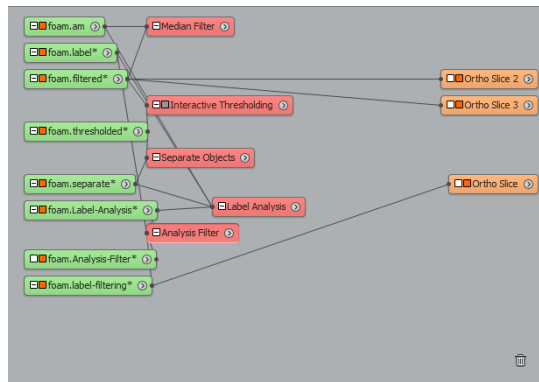
Figure 4.15: *Analysis Filter* workflow
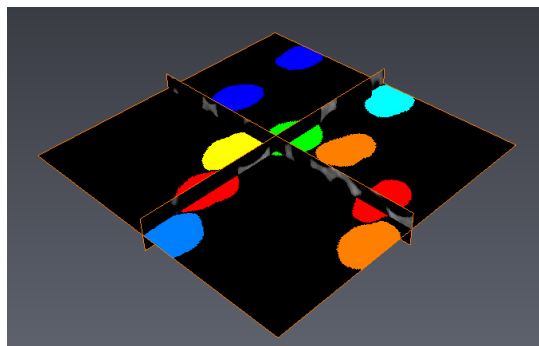
Figure 4.16: Filtering result

Tip: Filtering driven by measures can be a powerful tool for data segmentation. It allows you to select or eliminate regions based, for instance, on size, shape factor, orientation, or combinations of several

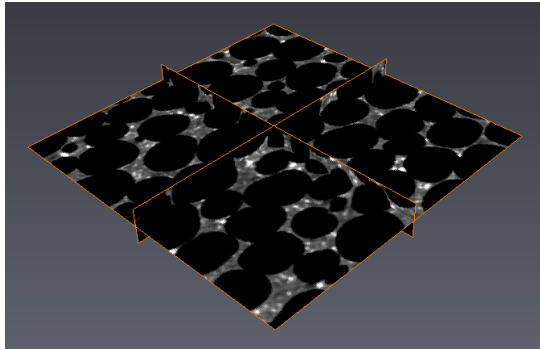criteria.

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-6-FilterAnalysis.hx will complete the steps above.

4.1.11 Classifying measures with sieves

You can define a set of value ranges, that can then be used for displaying a histogram using this distribution, or for creating a new label image showing the classification.

- Attach a *Sieve Analysis* module to `foam.Label-Analysis`.
- Connect the *Data* port to `foam.label`.
- Add a value by changing the *Number of values* to 4.
- Modify the suggested values by editing them or by moving the corresponding marker in the histogram. You can also press on the *Detect* button to get regular intervals.
- Press the *Apply* button. A new label image `foam.Sieved` has been created.
- Display the label image using a *Volume Rendering* module, as shown on Figure 4.18.
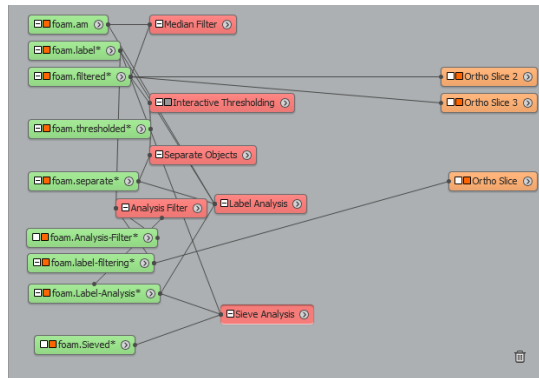
Figure 4.17: *Sieve Analysis* workflow

Loading the project data/tutorials/image-processing-advanced/GettingStartedBasics-7-SieveAnalysis.hx will complete the steps above.

4.1.12 Label images

- Hide the *Volume Rendering*
- Attach the first *Ortho Slice* to the label image `foam.label`.
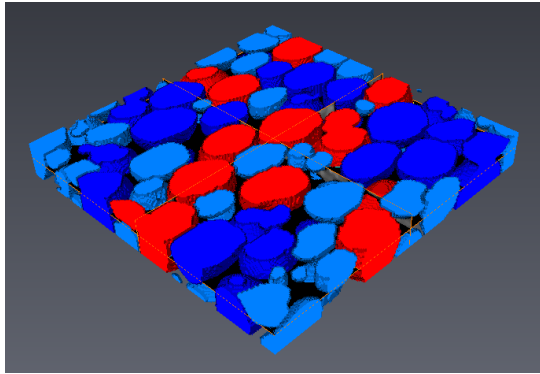
Figure 4.18: Sieve analysis result displayed with a *Volume Rendering*

In the *label* image created along with the result spreadsheet, each particle has been identified and assigned a unique index. This label data is stored in this case as a 16-bit label image. Such images are displayed by default using a cyclic colormap so that particles in close proximity are more likely to be shown in a different color, as shown on Figure 4.19.
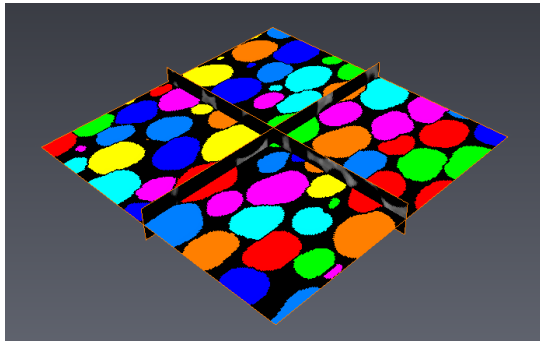
Figure 4.19: Each particle has been identified and assigned a unique index

In such cases, it is mandatory to convert a 16-bit per voxel label image into a 8-bit per voxel label image.

Note: Avizo label images coming from the *Segmentation Editor* and *Multi-Thresholding* modules are 8-bit per voxel label images.

4.1.13 Processing data on disk

In some cases, you may decide to work with Visilog (.im6) data format in order to be able to "keep data on disk" when opening data files, instead of loading data fully in memory. That way, some Avizo modules can, for instance, load and process image data slice by slice (out-of-core), avoiding the need to load the full data in memory (in-core). This allows processing of data that is much larger than the available memory on your system, at the expense of processing time.

By default, module outputs will be created in the same way as the module inputs. As a consequence, in order to process data fully on disk (input and output), you need to choose to *Stay on disk* when opening the input data.

In addition to Visilog format files, you can use *Stay on disk* with other data formats such as .lda. You can also load uncompressed Avizo files, Raw files as *Large Disk Data*.

4.1.14 Scripting

A complete processing sequence can be put in a script in order to automate analysis for routine tasks. For details about scripting with Avizo see the chapter 4.3 - Example 3: Separating, Measuring and Reconstructing.

4.1.15 Conclusion

This tutorial has introduced you to:

- Avizo interface modules and help menu,
- grayscale image processing modules,
- binary image processing modules,
- label images for storing indexed (segmented) images,
- 3D versus 2D stack processing modules,
- in-core versus out-of-core processing,
- how to compute measurements analysis,
- using filters to pare down segmented data,
- using sieves to interpret your measurements,
- scripting.

These concepts can be extended in countless ways to tackle new challenges. Try to relate your image processing problems back to this simple workflow:

- image processing to make data easier to binarize,
- binarization by tools like thresholding or top-hat (and sometimes combining two techniques),
- labeling to index all of the disconnected objects,

- measuring key properties for all indexed objects,
- analysis of measured data by visual inspection as well as filtering or sieving.

This introduction has highlighted how Avizo can be used to perform sophisticated segmentation and analysis of 3D data, but there are many more processing operations and measurements in addition to those presented here.

Avizo gives you this extensive toolkit so that you can map the appropriate tools to any processing challenge that confronts you.

4.2   Example 1: Measuring a Catalyst

This tutorial illustrates more techniques using Avizo:

1. Using masks to isolate object of interest.
2. Using distance map.
3. Using image arithmetics and distribution histogram.

To follow this tutorial, you should have read the first tutorial chapter 4.1 - Getting Started with Advanced Image Processing and Quantitative Analysis and be familiar with basic manipulation of Avizo. Display module visibility can be managed in the usual way with Avizo by clicking on the orange square button in the icon visible in Project View, or beside the module title in the Properties panel. See chapter 8.1.9 about Project Views.

The 3D image used in this example is acquired by microtomography: an almost spherical support contains catalyst and pores. The catalyst appears with dark levels in the image (low intensity voxels). The pores and background appear with bright levels (high intensity voxels). Intermediate gray levels correspond to the support.
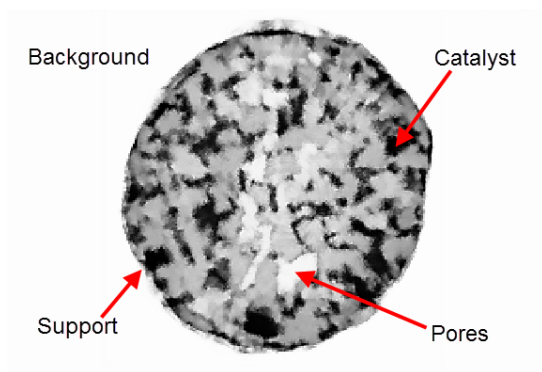
Figure 4.20: Microtomography image of catalyst and pores

The goal of this example is to get a distribution of distances between the catalyst voxels and the background (exterior). Here, a difficulty is the exterior intensity is close or identical to the intensity within the pores, which prevents the use of simple thresholding to isolate exterior. Moreover, some pores are connected with exterior, which prevents the use of "flood fill" approaches like *magic wand* of the *Segmentation Editor*, or the *Reconstruction From Markers* module.

Note: in this tutorial, you will find hints on how to manage an arbitrary region of interest. Another common example of a similar problem is to isolate the pore space of a rock core sample from core exterior, in order to compute rock porosity, for instance.

The process is split in several steps/sections that describe a step-by-step measurement workflow:

- *Object Detection and Masks*
- *More about Region of Interest and Masks*
- *Using Distance Map*
- *More about Distance Maps*
- *Measurement Distribution*

4.2.1 Object Detection and Masks

- Start by loading `data/tutorials/image-processing-advanced/Catalyst.am` from the `AVIZO_ROOT`directory.
- If Auto-Display is disabled, attach an *Ortho Slice* module to the `Catalyst.am` image icon in the Project View to display this image.

Loading the project data/tutorials/image-processing-advanced/CatalystDistribution-1-LoadData.hx will complete this tutorial step (see Figure 4.21).

Now, you can start with the first step used in this example for detection of the object: *thresholding*, then *closing* in order to "fill" the object and prepare a mask. The next section will give more hints on possible ways to create masks and arbitrary region of interest for your data.

Thresholding

- Attach an *Interactive Thresholding* module to the `Catalyst.am` module.

For searching an appropriate threshold, you can directly change the *Intensity Range* port. According to the *Preview Type* port, the 2D or 3D preview is interactively rendered. Remember to check through the whole volume by changing the *Preview Slice Number* and *Preview Orientation* ports. Other Avizo modules can also be helpful for this task (see Section 3.2 Visualizing 3D Images).
Here, thresholding the image between 0 and 225 gives a binary image where:

- intensity level = 1 −> support or catalyst (material),
- intensity level = 0 −> pore or exterior background.
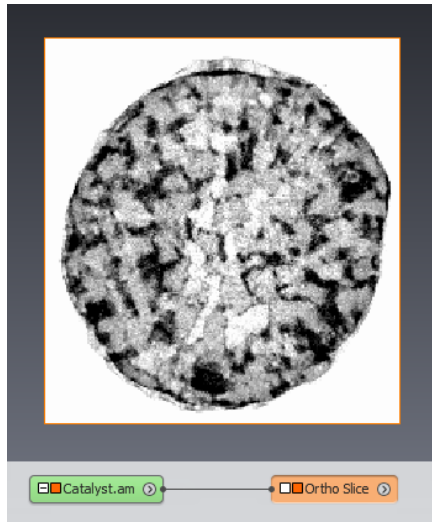
Figure 4.21: The initial image
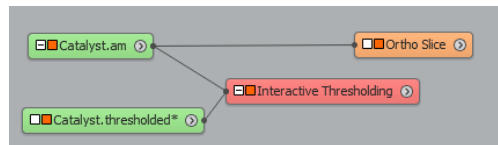
Figure 4.22: The project view

Applying the *Interactive Thresholding* module will create a binary image (image label with only interior and exterior materials) as described below:

- Select the *Interactive Thresholding* module in the Project view, then change the *Intensity Range* values to the range 0-225 by using the slider handles or the text areas of this port.
- Change the *Preview Slice Number* port to 45 by dragging the slider of this port.
- Press the *Apply* button to start processing.
- Check 3D in the *Preview Type* port to get a 3D preview and uncheck 3D to undo this preview.
- Attach the *Ortho Slice* currently linked to Catalyst.am to the resulting image, by click on and dragging the link of Ortho Slice to Catalyst.thresholded; an appropriate colormap will be selected by default.
- Hide the *Interactive Thresholding* preview by clicking on the orange square of this module in the Project view.

Loading the project CatalystDistribution-2-InteractiveThresholding.hx will complete this tutorial step (see Figure 4.23).
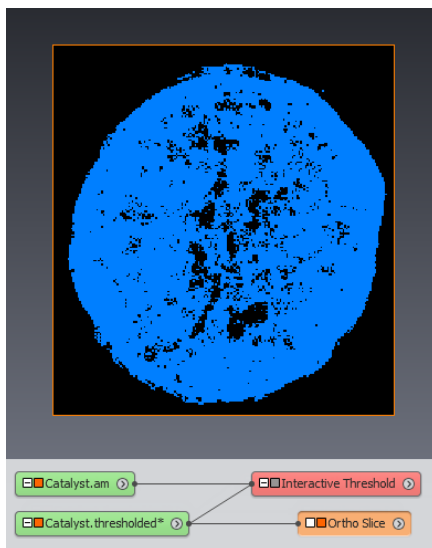
Figure 4.23: Material binary image after *Thresholding*

Morphological: Closing object

In order to detect the object shapes, you can apply morphological modules now. The mathematical morphology modules are transforms based on shape and size criteria.

The morphological *Closing* module applied on a binary image gives another binary image where:

- small holes inside objects are filled,
- objects boundaries are smoothed,
- close objects are connected.
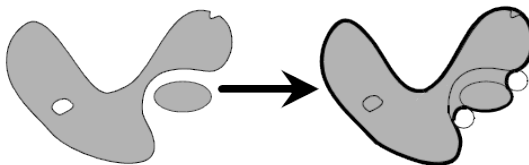
Figure 4.24: Effect of a *Closing* module on a binary image

The *Closing* module actually makes a dilation of the binarized regions, followed by an erosion: intuitively, the dilation fills holes and reconnects separated regions, then the erosion restores original exterior shape.

Here are the steps to fill the pores of the object:

- Attach a *Closing* module to `Catalyst.thresholded`.
- In order to fill any holes inside the object, you absolutely have to set the *Size* port to 6 (size of the structuring element). Such specific values may be found with a few trials and by checking through the whole volume by sliding an *Ortho Slice* for instance.
- Then push the *Apply* button to create the resulting binary image.
- Attach the already existing *Ortho Slice* to `Catalyst.closing`.

Loading the project CatalystDistribution-3-Closing.hx will complete this tutorial step (see Figure 4.25).
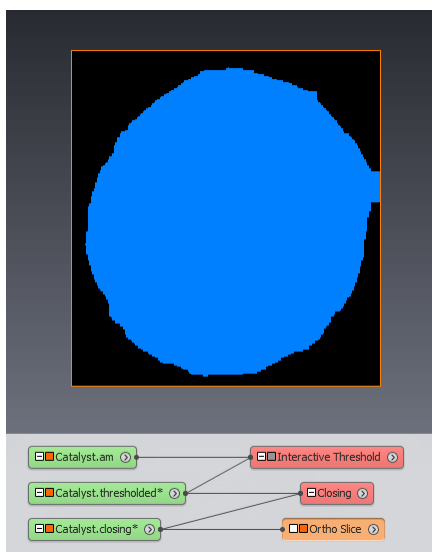
Figure 4.25: Object binary image after *Closing*

Note: The artifact you may notice on the right side of the image above is due the dilation too close to image border. To prevent this, the background border around the object should be larger than the size of closing. That could be easily solved by using the image *Crop Editor*: in this example, you could set, for instance, *Adjust* to 10, then uncheck *Replicate* for *Add mode* and set *Pixel value* to background intensity (i.e., 0); then pressing the *Enlarge* button would add a 10-voxel border. However, you can ignore this enlargement step in this tutorial.

4.2.2   More about Region of Interest and Masks

It is often necessary or useful to restrict measures or processing to a subset of the data.

If the subset is an axis-aligned box, you can use the following tools:

- Many modules support a *Region Of Interest* (ROI) input. You can attach a *ROI Box* module to your data, then connect the ROI input of the display or compute module to the ROI Box module.
- The image *Crop Editor* can cut or extend your data.
- The *Extract Subvolume* module copies a portion of your data in memory, possibly sub-sampled.

If you need an arbitrary mask or region of interest - for instance a cylindrical ROI, you can use the following tools:

- A number of modules sorted into *Image Processing/Image Morphology* from the popup menu of every binary image can be used to create or combine masks like the one shown above. Other examples include *Convex Hull* (applies slice by slice), *Fill Holes*, *Reconstruction From Markers*.
- The *Volume Edit* module is used to modify a volume with interactive tools like cylinder. It may also be used by script.
- The *Segmentation Editor* has a number of useful tools that can be used to quickly create masks, like brush, shaped lasso, *Selection/Interpolate*.

4.2.3   Using Distance Map

The second step is to compute a distance map of the catalyst. The next section gives more hints about distance maps.

Applying the distance map algorithm on a binary image gives a gray level image where each voxel intensity represents the minimal distance in voxels from the object boundary. For a given voxel intensity of the object distance map:

- Intensity level = 0 –> background
- Intensity level = 1 –> object envelope
- Others low level intensity –> part of the object close to the object envelope
- Others high level intensity –> part of the object far from the object envelope

Now, you can create this distance map as follows:

- Attach a new *Chamfer Distance Map* module to `Catalyst.closing`.
- Set the *Interpretation* to *3D* and click on *Apply*.
- Attach a new *Ortho Slice* to the result (`Catalyst.distmap`) to see the object's distance map.

Loading the project CatalystDistribution-4-DistanceMap.hx will complete this tutorial step (see Figure 4.26).
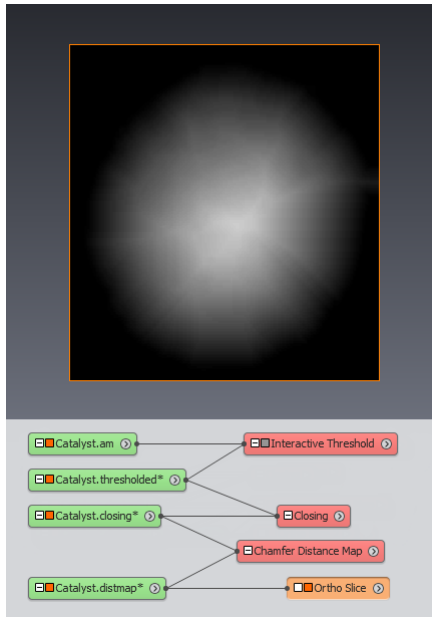
Figure 4.26: Object distance map

Masking

- Apply a *Interactive Thresholding* module to the initial image `Catalyst.am`.
- Set *threshold* between 0 and 100 and click on *Apply*.

This gives a binary image (`Catalyst2.thresholded`) where:

- Intensity level = 1 –> catalyst,
- Intensity level = 0 –> support, pore or background.

Loading the project CatalystDistribution-5-InteractiveThresholding.hx will complete this tutorial step (see Figure 4.27).

Masking will be used to compute the restricted distance map of the catalyst. The mask operation takes a gray level image for first input, a binary image for second input (mask image), and provides a gray level image for output where:

- each black voxel of the mask image is set to 0 in the output image,
- each blue voxel of the mask image is set to the initial level from the gray image.

Masking the distance map image by the catalyst image gives a gray image where:
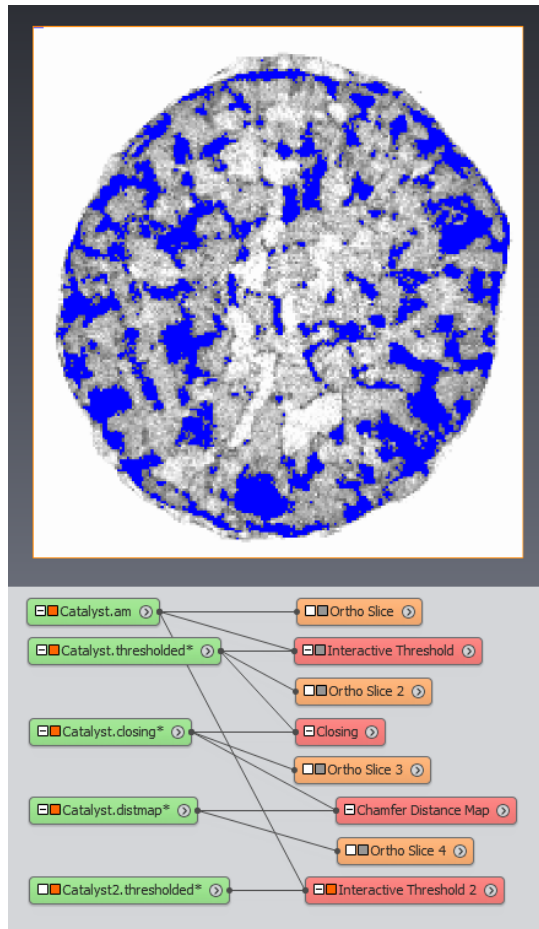
Figure 4.27: Catalyst binary image

- each non null intensity represents a voxel of the catalyst,
- the intensity value is equal to the distance in voxels from the object envelope.

You can create a such mask as follows:

- Apply a *Mask* module: the first input is `Catalyst.distmap` (the distance map image), and the second input is the catalyst binary image obtained previously `Catalyst2.thresholded`.
- Attach a new *Ortho Slice* to the result to see the object's distance map. To get a better ren-

dering, you can map the colormap range of the Ortho Slice to the full range (min-max) of `Catalyst.masked` : set 0...93 as min-max in the *Colormap* port.

Loading the project CatalystDistribution-6-Masking.hx will complete this tutorial step (see Figure 4.28).
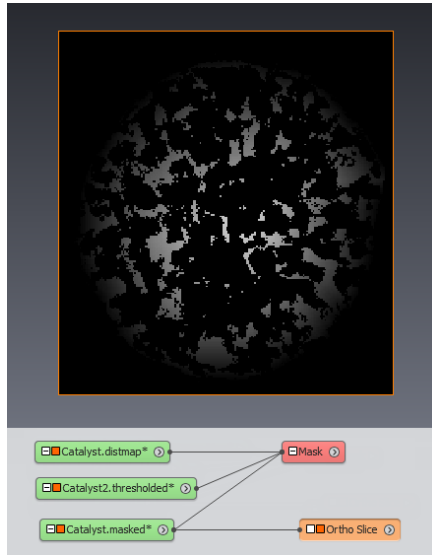
Figure 4.28: Catalyst distance map

4.2.4   More about Distance Maps

Distance maps (also called distance transforms) are powerful tools for a number of image processing techniques. The computed distance may be a discrete approximation (chamfer map) for faster computation. Avizo provides several versions of distance map that you may check for your specific purpose. Most of available distance modules are available in the subsection *Image Processing* of the module popup menu :

- *Chessboard Distance Map* (2D/3D chessboard),
- *2D/3D Chamfer Distance Map* (chessboard and diagonal),
- *Geodesic Distance Map* (based on mask to hide particular parts),
- *2D/3D Closest Boundary Points*,
- As a special case, *Propagation Distance* and related modules in *Image Processing/Image Morphology* group,
- *Distance Map*, using either chamfer or euclidian distances,

- *Image Processing/Distance Maps/Distance Map For Skeleton.* See Chapter 5.5 Skeletonization User's Guide for more details on distance maps and 3D skeletonization.
- *Image Processing/Distance Maps/Distance Map On Disk Data* that can operate only legacy LDD disk data.
- *Propagation Distance* (browsed-voxel distance) is referenced into the subsection *Image Processing/Image Morphology* of the module popup menu.

4.2.5 Measurement Distribution

You calculated a chamfer map measuring distances based on voxel units. For consistent results, you have to consider the voxel calibration: multiplying the distance image by the voxel size converts the image intensities into the metric system (micrometers).

The module group *Image Processing/Arithmetics Operations* is available for operations between two images or between an image and a constant.
Tip: the *Arithmetic* module also provides a flexible way to perform calculations with images.

You can get the distribution of the distances now:

- Use the *Multiply By Value* on `Catalyst.masked` as first input (*Input Image 1* port). Set the *Value* port to 5 (assuming a voxel size of 5 micrometers).
- Click on Apply to get `Catalyst.mult` as result.
- Retrieve the maximum value with the *Info* port displayed in the Properties panel when selecting the image icon in the Project View.
- Attach an *Histogram* module on `Catalyst.mult` to compute and plot for each gray level i, the number of voxels at intensity i. The number of points per each level will be graphed as a histogram. Set the *Range* port to {3,400} and the *Max Num Bins* port to 80. Uncheck *logarithmic* in the *Options* port and click on *Apply* to display the histogram window. Using the File menu, you can take a snapshot of the histogram or save the histogram data to a csv file.
- Applying an *Histogram* module on the catalyst distance map generates a graph showing the number of catalyst voxels located at a given distance from the object envelope.

Loading the project CatalystDistribution-7-Histogram.hx will complete this tutorial step (see Figure 4.29).

4.3 Example 2: Separating, Measuring and Reconstructing

This tutorial gives more details about object separation and extraction of geometric information:

1. *Principle of the Watershed Algorithm, an essential tool for image processing*
2. *Prior Segmentation*
3. *Object Separation using Watershed step-by-step*
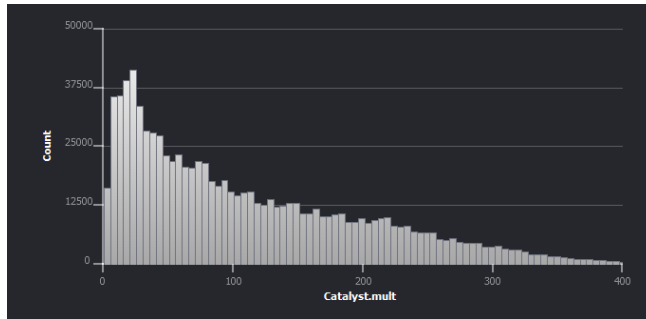
Figure 4.29: Distribution of the distances

4. *Separation Troubleshooting*
5. *Filtering Individual Objects*
6. *Geometry Reconstruction*

To follow this tutorial, you should have read the first tutorial chapter 4.1 - Getting Started with Advanced Image Processing and Quantitative Analysis and be familiar with basic manipulation of Avizo.

The 3D image used in this example was generated using data from several slices of foam.

The aim of the example is to isolate and quantify these bubbles, in a more detailed way than the *Getting Started* example, for better controlling on the results. Separation is essential in a number of cases for compensating too low image resolution, noise, or intensity variations across the image.

Figure 4.30: Foam image

Example 2: Separating, Measuring and Reconstructing   **149**
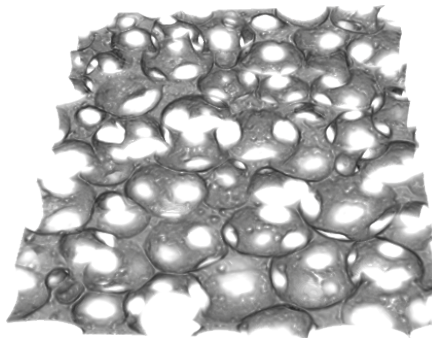
4.3.1 Principle of the Watershed Algorithm

The watershed algorithm is a powerful method that has many applications in image processing, for instance, for automated objects segmentation or separation.

This algorithm simulates the flooding from a set of labeled regions in a 2D or 3D image. It expands the regions according to a priority map, until the regions reach at the watershed lines. The process can be seen as progressive immersion in a landscape.
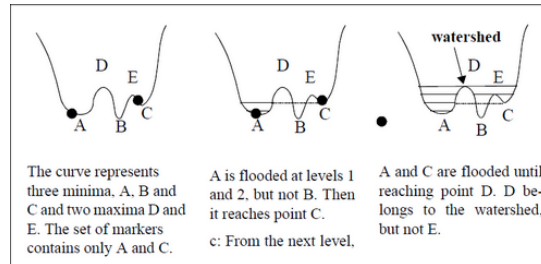
Figure 4.31: Watershed

This algorithm depends on two inputs:

1. A label image containing labeled marker regions that are used as seed areas for the flooding. There will be at the end of the process as many separated object as markers labeled differently. These are like rivers for which one want to retrieve the catchment area.
2. A gray image playing the role of the landscape height field or altitude map that controls the flood progression and finally the location of watershed separations. These separations are located on the crest lines between valleys of our landscape.

Different applications can be achieved by carefully choosing both inputs: markers and priority map.

An example is the *Separate Objects* module used for separating foam pores in the *Getting Started* tutorial. It first computes a distance map (see the chapter 4.2 - Measuring a Catalyst tutorial for more about distance maps). This distance map provides the priority map input for a watershed process. Maxima regions of the distance maps - the most inner areas of the pores - provide the markers input used for the watershed. The process is described in details in the next section.

4.3.2 Prior Segmentation

First, you need to segment the data, in order to get a binary image of the pores.

- Start by loading in Avizo the image stack `data/images/foam/foam.am` from the `AVIZO_ROOT` directory.

- You might want to perform some kind of noise reduction on your data. You could typically apply a filter such as *Median Filter* with *3D Interpretation*; the median filter is a non linear digital filtering technique, often used to preserve edges while removing noise. Such noise reduction is a typical pre-processing step to improve the results of later processing like segmentation or edge detection.

  Nevertheless, you can skip this stage in this example and start creating a binary image by thresholding.

- Attach an *Interactive Thresholding* module to the `foam.am` object in the project view.

- By using the cursors of the *Intensity Range* port slider, set the low and high threshold levels to 0 and 38 and click on *Apply*. In the preview, you can see that the resolution and intensity distributions in the image do not allow you to directly segment separated pores: some pores remain connected, whatever threshold is chosen, unless too much noise is left in pores.

- Attach an *Ortho Slice* to `foam.thresholded` to visualize the result.

Loading the project WatershedSeparation-1-Thresholding.hx will complete this tutorial step (see Figure 4.32).
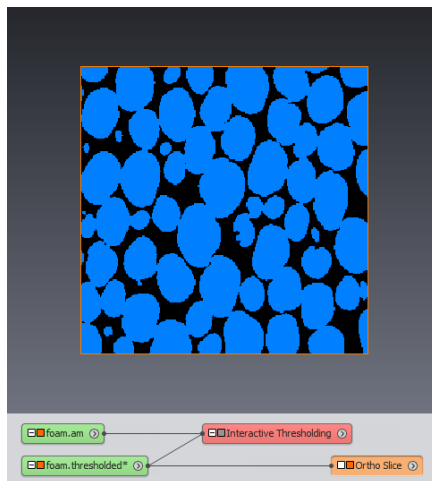
Figure 4.32: Binary image

By dragging the cursor of the *Slice Number* port, you may notice some artifact holes (e.g., in slices 4, 5, 6), mostly related to higher intensity rings crossing the pores that appear on the gray image. Attach an *Ortho Slice* to the gray image and set *Mapping type* to *histogram* to highlight these. Hide or remove the previous *Ortho Slice*.

Rather than doing upstream correction of gray image or even acquisition, it may be more effective in some case to correct the binary image, for instance filling holes.

Optionally, it is recommended to fill holes within objects to separate because the separation method based on distance map can be sensitive to these artifact holes.

- You can attach a *Fill Holes* module to `foam.thresholded`.
- Set *Interpretation* to *3D* and click on *Apply* to get `foam.filled` as result.

4.3.3 Separation using Watershed step by step

Starting from binary-image, you can proceed to pores separation now. You will compute a distance map, create markers from maxima regions of the distance map, then apply the fast watershed algorithm.

- Attach a *Chamfer Distance Map* module to the binary-image object (`foam.thresholded`).
- Set *Interpretation* to *3D* and click on *Apply* to get `foam.distmap` as result.
- Attach an *Ortho Slice* to control the result. Each voxel within a pore receives a value corresponding to its distance to the black background (the foam).

Loading the project WatershedSeparation-2-DistanceMap.hx will complete this tutorial step (see Figure 4.33).
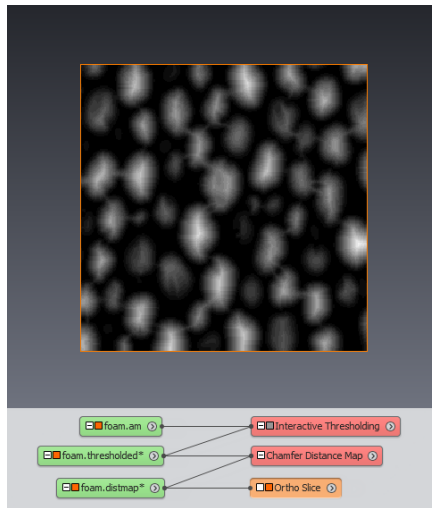
Figure 4.33: Chamfer distance map

- Attach a *H-Maxima* module. Leave the *Contrast* port to default value (4) and click on *Apply*.
- Attach an *Ortho Slice* to `foam.hMaxima` and set the *Transparency* type to *Alpha*.

Loading the project WatershedSeparation-3-MergedMaxima.hx will complete this tutorial step (see Figure 4.34).
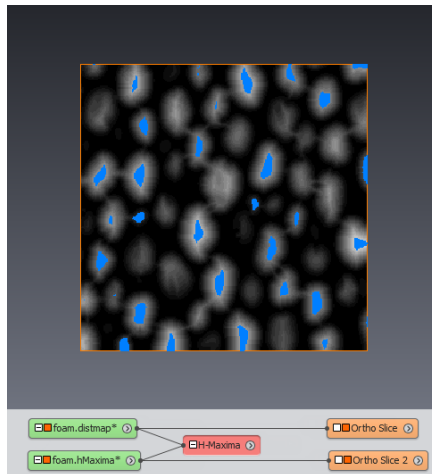
Figure 4.34: Merged maxima of distance map

This module creates a binary image containing the regional maxima of the input distance map image, which are "merged" within the contrast variation given as parameter. Since distance map is used as input, the result is the set of most inner regions within objects.
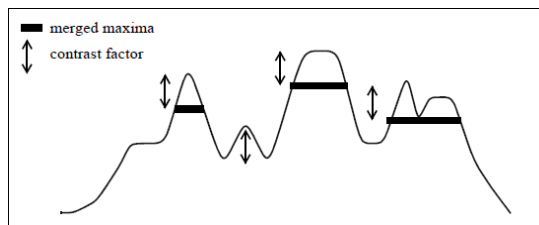
Figure 4.35: Principle of H-Maxima (merged maxima) shown on a image profile sample

The watershed algorithm requires a unique label for each region finally separated. Two regions with the same value in input image would be merged.

- Attach a *Labeling* module to `foam.hMaxima` and click on *Apply*.
- Attach an *Ortho Slice* to `foam.labels` and set the *Transparency* type to *Alpha*.

Loading the project WatershedSeparation-4-Markers.hx will complete this tutorial step (see Figure 4.36).

Example 2: Separating, Measuring and Reconstructing 153
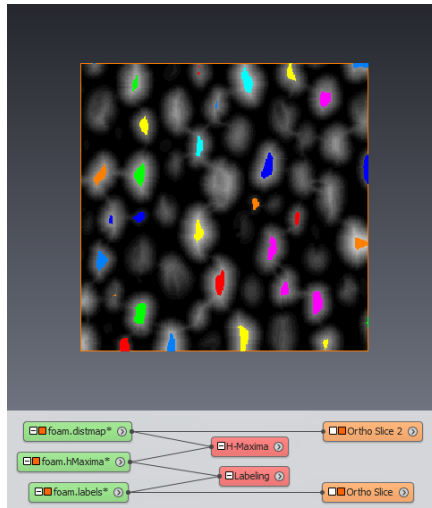
Figure 4.36: Markers created by the *Labeling* module

The distance map also needs to be inverted, as the watershed algorithm will expand the markers towards increasing values of the input priority map (i.e., landscape altitude).

- Apply a *NOT* module to `foam.distmap`.
- Attach an *Ortho Slice* to `foam.not`.

Loading the project WatershedSeparation-5-ReversedDistanceMap.hx will complete this tutorial step (see Figure 4.37).

You can compute the image of watershed separation lines now.

- Attach a *Marker-Based Watershed* module to the reversed-distance data object (`foam.not`). Set `foam.labels` as the *Input Label Image*, and the *Type* to *Watershed*. Press Apply.
- Attach an *Ortho Slice* to the result to see watershed lines. Set *Alpha* or *Binary* as the *Transparency* type to overlay lines onto gray image.
- Attach an *Ortho Slice* to `foam.am`. Set the *Colormap* minimum to 0.

Loading the project WatershedSeparation-6-SeparationLines.hx will complete this tutorial step (see Figure 4.38).

To complete the separation, you can subtract separation lines from the binary image of pores:

- Apply an *AND NOT Image* module to `foam.thresholded` as first input and `foam.watershed` as second input.
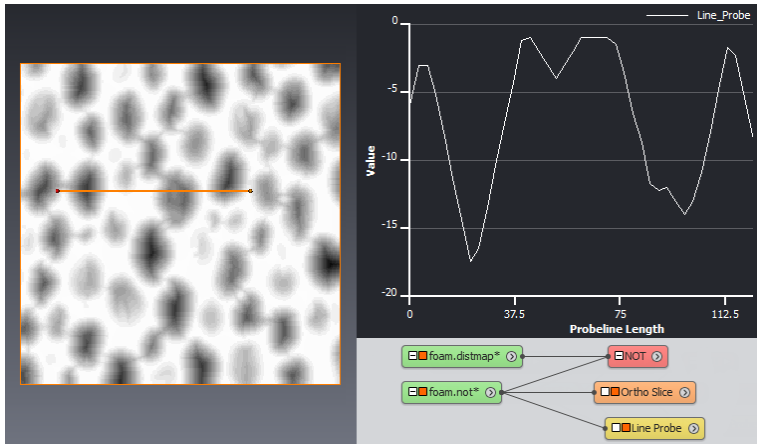
Figure 4.37: Inverted distance map displayed using a *Line Probe* module
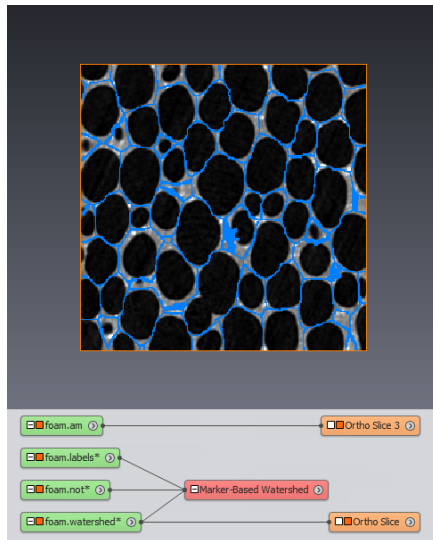
Figure 4.38: Watershed separation lines

- Attach an *Ortho Slice* to see the result `foam.sub`.

Loading the project WatershedSeparation-7-SeparatedPores.hx will complete this tutorial step (see Figure 4.39).
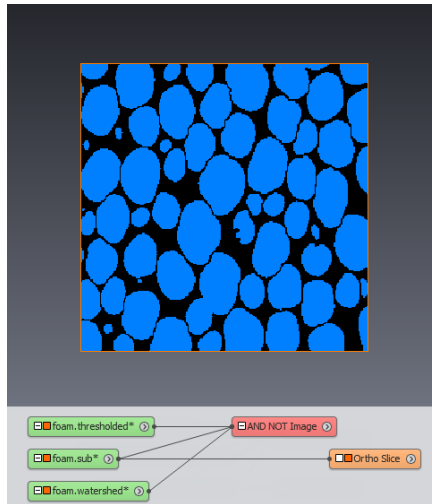
Figure 4.39: Separated pores
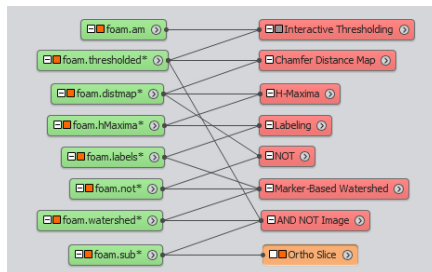
Figure 4.40: The watershed separation workflow

4.3.4 Separation Troubleshooting

You may see in the result unseparated pores, or on the contrary, unwanted separations of pores. Because it is based on the geometrical distance criterion, the separation will work best for convex, almost spherical objects. Here are some guidelines for improving separation :

- If you look to markers image on a particular slice, markers may seem missing in some pores just because they lay somewhere else in 3D.
- In some case, however, a marker can be really missing because two objects are considered merged from the standpoint of distance map regional maxima, then a separation will also be missing. You can try lowering the contrast factor of *H-Maxima* (or *Separate Objects*) to make markers smaller and more separated.

- If a separation is missing, it means that a marker is missing,

- Separation 'lines' may look too thick or wrong, just because the slice you are looking at is somewhat tangent to a separation face.

- Unwanted separation may occur if the object shape is non-convex: this leads multiple local maxima, therefore separated object. Small concavity in an object can lead to a separation across it. A solution may be to increase the contrast factor of *H-Maxima* in order to make markers larger and merged,

- A separation based on distance map may follow the shortest path, i.e., straight line instead of the desired shape. This is because the watershed is driven by the distance: the separation is driven by geometrical criterion.

- *Chamfer Distance Map* is a discrete chamfer map. You could use instead a more accurate euclidian distance map (see *Distance Map* or other distance map types referenced into *Image Processing/Distance Maps* from the object popup); however, this has little impact in most cases.

As a main guideline, there will be as many separated objects as markers.
If results are not satisfying, there can be two cases:

- The number of objects is not satisfying: in this case, you must work on markers.

- The lines of separation are not satisfying: you must work on the priority map, the 'landscape height field'.

For markers, the *Separate Objects* solution is to work on the distance map. In this case, you take into account the geometry (most inner regions), but markers may be obtained by other ways. Sometimes it may be interesting to use the grayscale image (intensity) if the centers of grains are darker or lighter.
If the objects are fairly homogeneous, you must stick with geometric information. One may need to adjust the contrast factor setting of *H-Maxima*. Basically, the *H-Maxima* parameter corresponds to the minimum depth of between two maxima. With geographic analogy: the difference in height between the collar and a summit so that you keep the two distinct peaks.
It can be easier to make trials on a cropped data set showing object(s) that should not be split and tune settings in order to make sure that only one marker is inside each object. One possibility for improvement is to do a *H-Maxima* and then mask the result with a threshold image of the distance map. This avoids keeping markers for small connected parts, since you keep only markers corresponding to a maximum of distance map with a minimal value for distance.

Once satisfied with markers, corresponding to the number of objects, you can also improve the lines of separation. Once again, the process described above (*Separate Objects*) relies on the geometry, you can also use as a function of depth - either directly grayscale intensity or the gradient of intensity. It may be a little difficult to combine geometrical information and intensity information. In some cases, you can get by with a combination of the distance function and a function of intensity, for example, with the *Blend With Image*, *Blend With Value* or *Arithmetic* module, somehow simulating the complex way the human eye can combine both informations. Note that the watershed seeks for lines peak separations, therefore local maxima - as for distance map, you may need to use the negative of function to bring to this case.

Example 2: Separating, Measuring and Reconstructing **157**

An additional powerful technique to improve separation is to filter the separations added by watershed method, based on some criteria:

- isolate the added separations: get the difference between original and separated image using the *AND NOT Image* module for instance,
- label these separations with the *Labeling* module,
- use the *Label Analysis* module to do some measurement in each separation, for instance, the maximum of distance map value within a separation region could indicate whether a separation goes too deep inside an object.

You will find more example applications of watershed in the next tutorials, using different methods to create the markers and the priority map.

4.3.5 Filtering Individual Objects

You can measure individual objects that have been separated.

First, you need to convert the binary image of separated pores to a label image with each pore uniquely identified:

- Apply a *Labeling* module to the separated-pores image (`foam.sub`).
- You can attach a *Voxelized Rendering* module to see the labeled image.

Loading the project WatershedSeparation-8-SegmentedPores.hx will complete this tutorial step (see Figure 4.41).
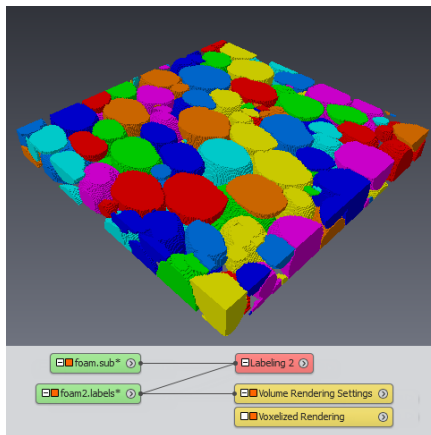
Figure 4.41: Segmented pores

Secondly, you want to filter unwanted small objects, and finally, measure the volumes of individual

pores.

In order to remove small objects, you can use measures and measure filters as described in the *Getting Started* tutorial. This involves two steps:

- Attach a *Filter By Measure* module to the label image.
- Set `foam.am` as *Input Intensity Image*.
- Select *Volume3d* in the huge list of available measure of the *Measure* port.
- Set the *Number Of Objects* port to 15 to keep only the 15 biggest volumes. Press *Apply*.
- A *Voxelized Rendering* or *Boundary Rendering* module can be used to see the biggest volumes.
- You can also use a *Label Analysis* module to display and manage specific measurement in the Tables panel.

In one of the next tutorials (chapter 4.4 - Further Image Analysis), you will find more example applications of measurement and advanced analysis.

Loading the project WatershedSeparation-9-FilteredPores.hx will complete this tutorial step (see Figure 4.42).
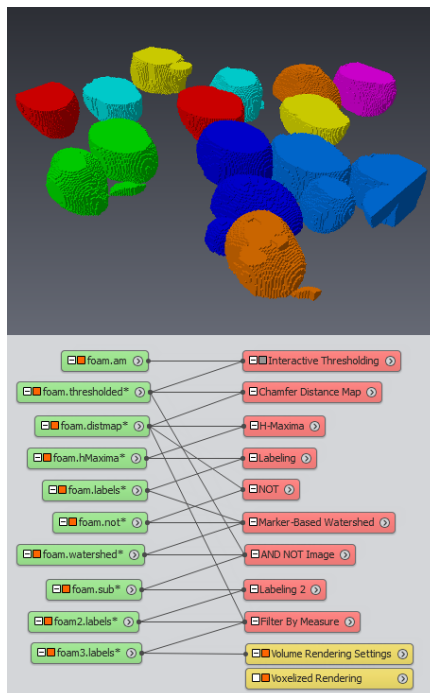
Figure 4.42: Segmented and filtered pores

4.3.6 Geometry Reconstruction

Finally, you can reconstruct the geometry of the bubbles:

- Attach a *Generate Surface* module to the result label image (`foam3.labels`), uncheck *Adjust Coords* in the *Border* port, and press *Apply* to generate `foam3.surf`.
- Display the resulting surface by attaching a *Surface View* module.

Loading the project WatershedSeparation-10-Reconstruction.hx will complete this tutorial step (see Figure 4.43).
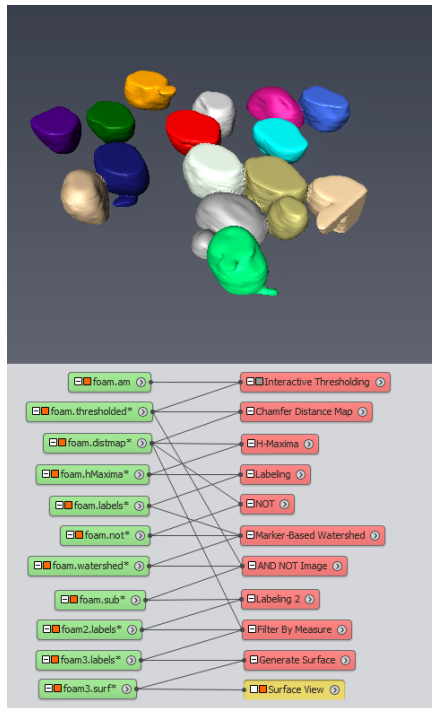
Figure 4.43: Surface reconstruction of filtered pores

As the result image already contains integer values for material labels, it can be used directly for surface reconstruction. Other image types may require conversion to Avizo label data (for instance by using the *Convert Image Type* module). Notice that the *Generate Surface* module can work with more than 256 labels.

Displaying the resulting surface with a Surface View module may be very slow due to the large number of surface polygons. In this case, a prior simplification of the surface is recommended for faster display

on your hardware.

Avizo also allows you to export the surfaces to various file formats, or to generate and export a tetrahedral model suitable, for instance, for finite element simulation with some external solver.

A demo script corresponding to this whole tutorial can be found in:
data/tutorials/image-processing-advanced/PorositySurfaceReconstruction.hx
It uses a script object located in data/tutorials/image-processing-advanced for automating the processing. It matches the workflow described on the figure 4.40.
Once the script is loaded, you can optionally change several port values and click on the *Apply* button of the *Action* port to start the processing.

4.4   Example 3: Further Image Analysis - Distribution of Pore Diameters in Foam

This example shows how to compute the distribution of pore diameters in a foam sample and how to define a custom measure to compute the sphericity of pores.
To follow this tutorial, you should have read the chapter 4.1 - Getting Started with Advanced Image Processing and Quantitative Analysis and be familiar with basic manipulation of Avizo.

This section is divided in the following steps:

- *pore detection*,
- *pore post-processing*,
- *custom measure group definition to determine the distribution of pore diameters*,
- *custom measure definition to compute the sphericity of pore.*

The image used in this example is acquired by microtomography. It represents foam that consists of material and pores. Pores appear with dark levels in the image (low intensity voxels). Material appears with luminous levels (high intensity voxels).

- Start by loading `data/tutorials/image-processing-advanced/FoamPoro.am` from the `AVIZO_ROOT` directory.
- If Auto-Display is disabled, connect an *Ortho Slice* to visualize the data in the 3D viewer as shown on Figure 4.44.

4.4.1   First step: pore detection

- Connect an *Interactive Thresholding* to the data.
- Use the *Threshold* port to threshold the image between 0 and 50.
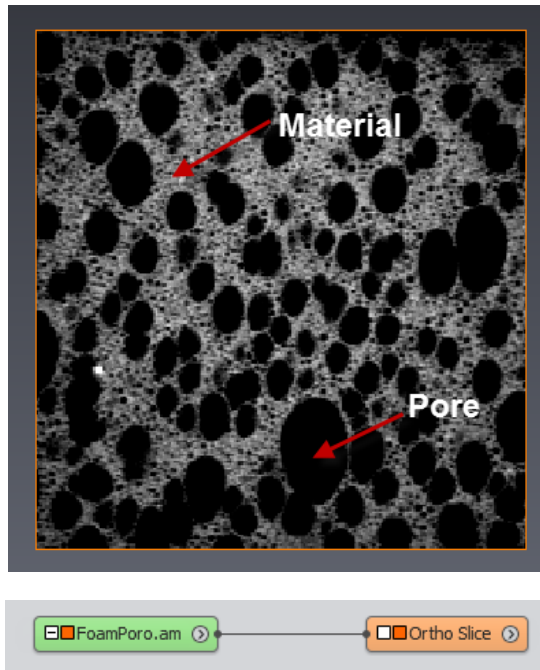
Figure 4.44: Initial microtomography image of foam pores

- Press Apply.
- Hide the *Interactive Thresholding* module and connect the *Ortho Slice* to the output `FoamPoro.thresholded.`

As shown on Figure 4.45, thresholding the image between 0 and 50 gives a binary image where: intensity level 1 = porosity, intensity level 0 = support (material).

4.4.2 Second step: pore post-processing

The morphological *Opening* operator applied on a binary image gives another binary image where: small objects are removed, objects boundaries are smoothed, some objects may be disconnected.

- Connect an *Opening* module to the binary image.
- Set the *Size [px]* port to 1.
- Press Apply.
- Connect the *Ortho Slice* to the output `FoamPoro.opening.`
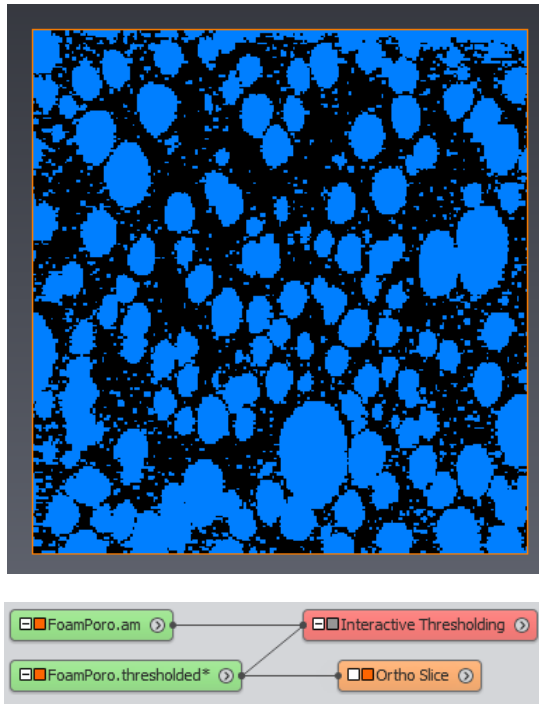
Figure 4.45: Pores

Applying morphological opening on the previously computed pores binary image gives a filtered image where noise and artifacts are reduced, as shown on Figure 4.46.

The *Separate Objects* module detects surfaces that separate agglomerated particles. These surfaces are subtracted from the initial image, see Figure 4.47.

- Connect a *Separate Objects* module to the filtered binary image.
- Set the *Marker Extent* to 1.
- Press Apply.
- Connect the *Ortho Slice* to the output `FoamPoro.separate` (see Figure 4.48).

4.4.3 Third step: custom measure group definition to determine the distribution of pore diameters

The Avizo*Label Analysis* module allows computation of a set of measures for each particle of a 3D image. Once the individual analysis is performed, a histogram of a given measure may be plotted in
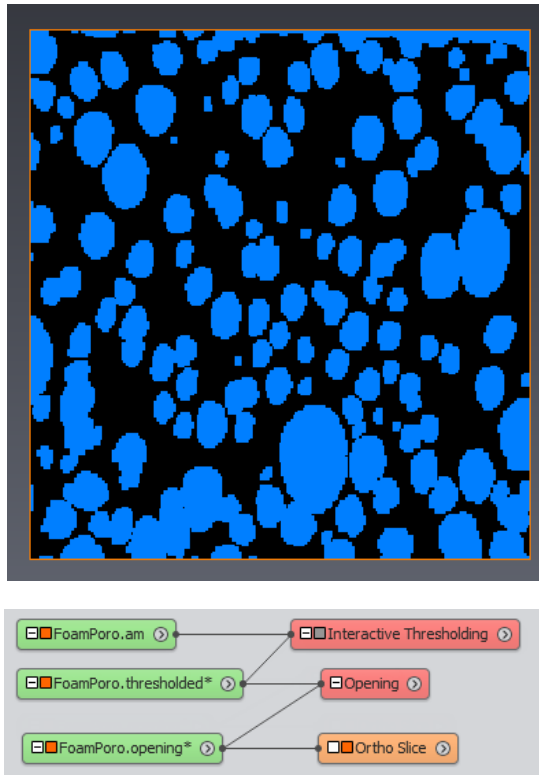
Figure 4.46: Filtered pores

Input binary image   Line of separation image   Output binary image
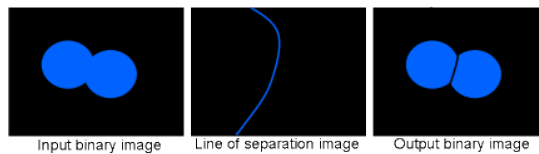
Figure 4.47: Pore post-processing

order to produce a representation of the measure distribution.

- Connect a *Label Analysis* module to the separated binary image.
- Set `FoamPoro.am` as *Intensity Image*.
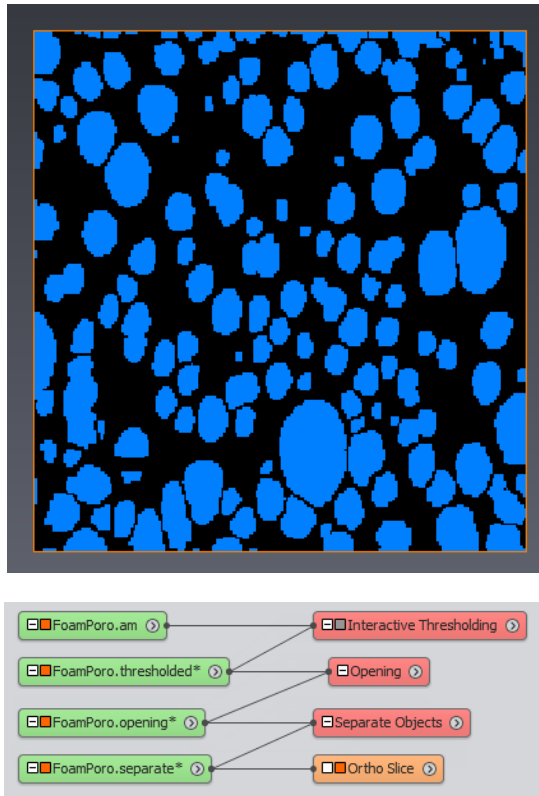
Figure 4.48: Separated pores

In the *Measures* port of the module, *basic* is a group of pre-selected native measures. It might happen that you don't need all the measures of the *basic* measure group, or you would like to bundle a different set of measures in the analysis table. For these cases, you can create your own measure group.

For a given particle, the equivalent diameter measure computes the diameter of the spherical particle of same volume. So the equivalent diameter is given by the following formula:

$$EqDiameter = \sqrt[3]{\frac{6 \times Volume3d}{\pi}}$$

- Press on the configuration button of the *Measures* port (the button with 3 dots). A panel opens for the selection of measure groups (see Figure 4.49).
- Create a new measure group by pressing the dedicated button next to the measure group selector (1).

Example 3: Further Image Analysis - Distribution of Pore Diameters in Foam   **165**

- In the popup window, name the new group *diameter* and press OK.
- Select EqDiameter in the native measures list (2) and use the arrow button to add it to the group (3).
- Press OK.
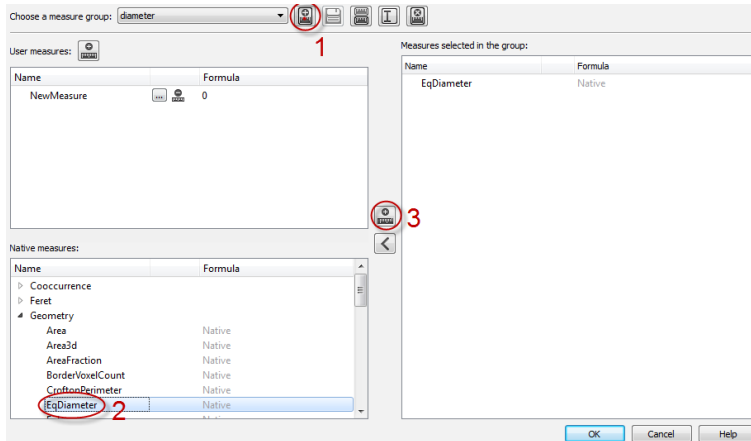
Figure 4.49: Selection of measure groups

The new *diameter* group, containing only the equivalent diameter measure, is now selected in the *Measures* port of the *Label Analysis* module.

- Press the *Apply* button.

A new *label* image data object `FoamPoro.label` is created in the Project View, and the Tables panel is displayed, showing a spreadsheet-style table of results: the analysis *FoamPoro.Label-Analysis*, also created in the Project View (see Figures 4.51 and 4.52).

- Select the EqDiameter column in the lower spreadsheet.
- Press on the histogram button of the toolbar.

A window opens, displaying the EqDiameter histogram, as shown on Figure 4.53.

Loading the project data/tutorials/image-processing-advanced/CustomerMeasurements-1-EquivalentDiameter.hx will complete the steps above.

4.4.4 Fourth step: custom measure definition to compute the sphericity of pore

Avizo provides a set of pre-defined native measures but it is also possible to save user-defined custom measures.
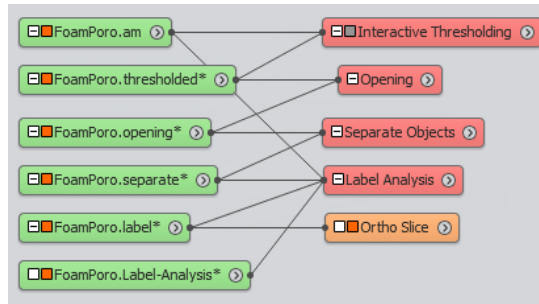
Figure 4.50: Analysis project

Figure 4.51: Analysis project

Figure 4.52: Analysis spreadsheets

- Select the *Label Analysis* module.
- In the Properties panel, press on the configuration button of the *Measures* port (the button with 3 dots).
- Choose the *diameter* group if it is not yet selected.
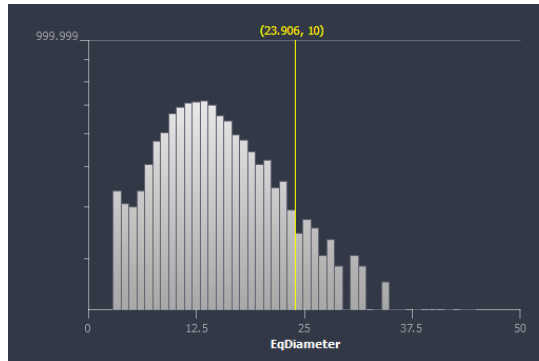
Figure 4.53: EqDiameter histogram
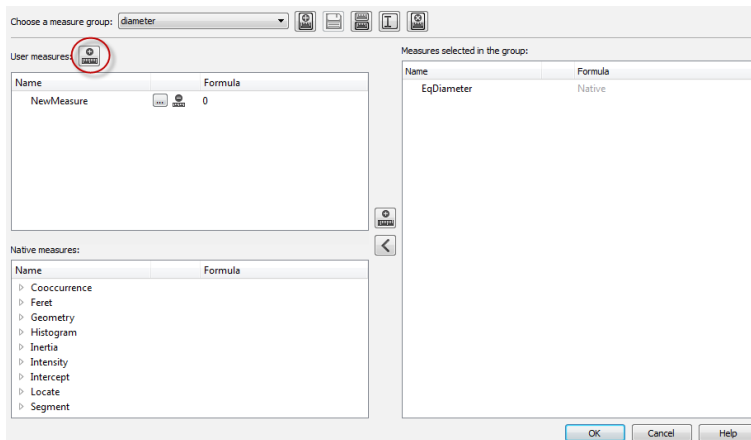
Figure 4.54: User measures definition button

- Create a user measure by pressing the dedicated button (see Figure 4.54).
- Name it *Sphericity* and press OK.

The Measure Edition panel opens.

The sphericity is a measure of how spherical an object is and is expressed as:

$$\Psi = \frac{\pi^{1/3}(6V)^{2/3}}{A}$$

where V is the volume of a particle and A is its surface area.

It is the ratio of the surface area of a sphere (with the same volume as the given particle) to the surface area of the particle. The sphericity of a sphere is 1 and, by the isoperimetric inequality, any particle which is not a sphere will have sphericity less than 1. Here are several examples of object sphericity :

Object	Volume	Area	Sphericity
cone : $h = 2\sqrt{2}r$	$\frac{2\sqrt{2}}{3}\pi r^3$	$4\pi r^2$	≈ 0.794
cylinder : $h = 2r$	$2\pi r^3$	$6\pi r^2$	≈ 0.874
torus : $R = r$	$2\pi^2 r^3$	$4\pi^2 r^2$	≈ 0.894
sphere	$\frac{4}{3}\pi r^3$	$4\pi r^2$	1
icosahedron : $20\,faces$	$\frac{5}{12}(3+\sqrt{5})s^3$	$5\sqrt{3}s^2$	≈ 0.939
dodecahedron : $12\,faces$	$\frac{1}{4}(15+7\sqrt{5})s^3$	$3\sqrt{25+10\sqrt{5}}s^2$	≈ 0.910
octohedron : $8\,faces$	$\frac{1}{3}\sqrt{2}s^3$	$2\sqrt{3}s^2$	≈ 0.846
cube : $6\,faces$	s^3	$6s^2$	≈ 0.806
tetrahedron : $4\,faces$	$\frac{\sqrt{2}}{12}s^3$	$\sqrt{3}s^2$	≈ 0.671

This can be expressed, with respect to Avizo measures, as:
`(pi**(1/3)*(6*Volume3d)**(2/3))/Area3d`

- Enter the shericity formula in the dedicated field of the panel (see Figure 4.55).
- Press Close.
- Select Sphericity in the user measures list and use the arrow button to add it to the *diameter* group.
- Press OK.
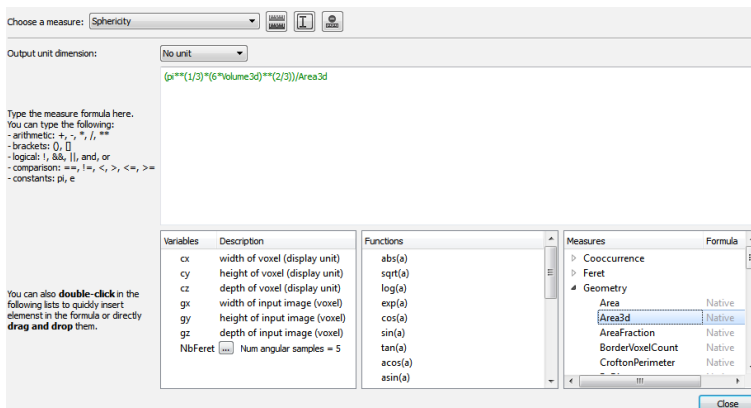
Figure 4.55: Measure edition

- Press the *Apply* button of the *Label Analysis* module.

Example 3: Further Image Analysis - Distribution of Pore Diameters in Foam   **169**

The Analysis panel is updated and displays the EqDiameter and Sphericity measures.

Tip: Using a custom group to select only the needed measures can help to speed up the analysis process.

Note 1: User-defined data such as custom measure groups or custom measures are persistent at the end of work session as local settings, so they can be retrieved when you restart Avizo. These custom data are also saved in project scripts.

Note 2: It is important to mention that the sphericity computed could be superior to 1 for small pores (i.e., composed of few voxels). This is due to that the Area 3D measure is computed with chordal approximations (which gives generally a better approximation of the area) whereas it is not the case for the Volume 3D measure which does not use any approximation.

Avizo offers powerful ways to define custom measures. See the user's guide and reference guide for further details.

Loading the project data/tutorials/image-processing-advanced/CustomerMeasurements-2-Sphericity.hx will complete the steps above.

4.5 Example 4: Further Image Analysis - Average Thickness of Material in Foam

This tutorial shows how to compute the thickness of material in a foam sample.

To follow this tutorial, you should have read the first tutorial chapter 4.1 - Getting Started with Advanced Image Processing and Quantitative Analysis and be familiar with basic manipulation of Avizo.
At the end of this tutorial, you will find a link to a corresponding demo script.

Start by loading data/tutorials/image-processing-advanced/FoamPoro.am, an image acquired by microtomography and stored into the AVIZO_ROOT directory.
It represents foam that consists of material and pores. Pores appear with dark levels in the image (low intensity voxels). Material appears with luminous levels (high intensity voxels).

The algorithm may be divided into 4 steps:

- *Porosity Detection*
- *Detection of the Separation Surfaces*
- *Distance Map of the Material*
- *Calculation of the Material Average Thickness*
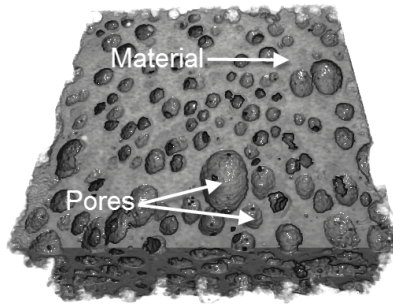
Figure 4.56: Microtomography image of foam pores

4.5.1 Porosity Detection

Reproduce the first steps of the previous tutorial: use thresholding, morphological opening and separating modules on the initial image to get a binary image of the filtered and separated pores.

Loading the project PorosityThickness-1-SeparationObjects.hx will complete this tutorial step (see Figure 4.57).
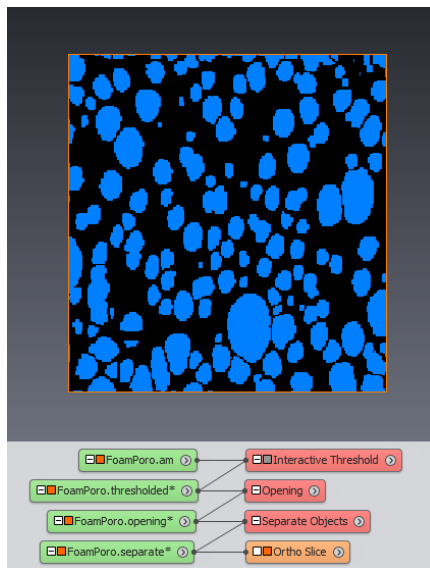
Figure 4.57: Binary image of porosity

4.5.2 Detection of the Separation Surfaces

The goal of this step is to detect surfaces that cross the material at an equidistance from two pores. The *Influence Zones* module:

- takes a binary image as input,
- gives a binary image as output where:
  - each blue voxel of the output image is closer to the object located at the center of the zone in the input image,
  - each black voxel is equidistant from at least two closer objects.

Apply the *Influence Zones* module on the binary image of porosity `FoamPoro.separate`.

Loading the project PorosityThickness-2-InfluenceZones.hx will complete this tutorial step (see Figure 4.58).
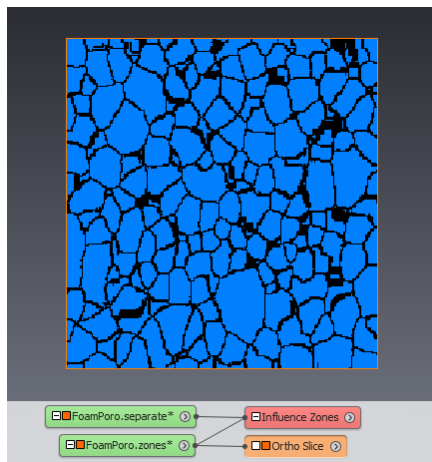
Figure 4.58: Image/skeleton by Influence Zones

The *NOT* module inverts the levels of a binary image. Applying *NOT* on an skeleton by influence zone (SKIZ) gives a binary image where:

- each black voxel of the output image is closer to the object located at the center of the zone in the input image,
- each blue voxel is equidistant from at least two closer objects.

So the *Influence Zones* and *NOT* combination provides a binary image of surfaces that separate pores through the material.

Apply the *NOT* module to `FoamPoro.zones`.

Loading the project PorosityThickness-3-SeparationSurfaces.hx will complete this tutorial step (see Figure 4.59).
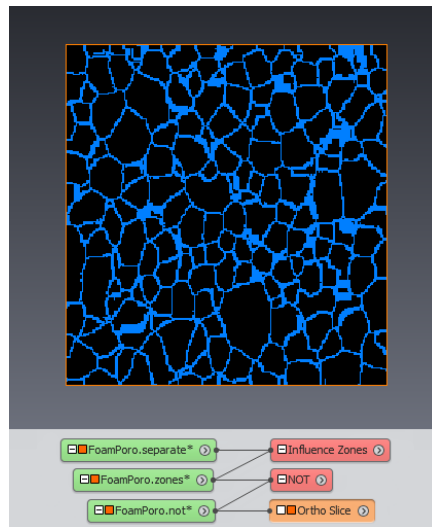
Figure 4.59: Separation surfaces image

4.5.3   Distance Map of the Material

Apply *NOT* on the binary image of porosity `FoamPoro.separate`.
This gives a binary image where:

- each black voxel of the output image represents a background or porosity voxel,
- each blue voxel represents a material voxel.

The *Chamfer Distance Map* module applied on a binary image gives a gray level image where each voxel intensity represents the minimal distance in voxels from the object boundary. For a given voxel intensity:

- intensity level = 0 corresponds to the pores or background
- intensity level = 1 corresponds to the material envelope
- other low level intensities correspond to the part of material close to the material envelope
- other high level intensities correspond to the part of material far from the material envelope
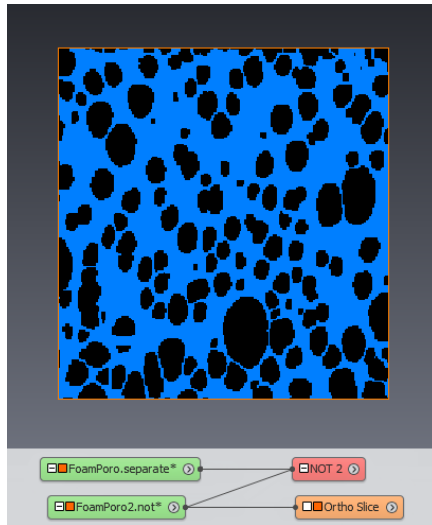
Figure 4.60: Material binary image

Apply a *Chamfer Distance Map* module with *3D Interpretation* on the material binary image `FoamPoro2.not`.

Loading the project PorosityThickness-4-DistanceMap.hx will complete this tutorial step (see Figure 4.61).

The *Mask* module:

- takes a gray level image as the first input,
- takes a binary image as the second input (mask image),
- provides a gray level image as the output where:
    - each black voxel of the mask image is set to 0 in the output image,
    - each blue voxel of the mask image is set to the initial level from the gray image.

Apply a *Mask* module to the distance map image `FoamPoro2.distmap` and set the *Input Binary Image* to the separation surfaces image `FoamPoro.not`.
Masking the distance map image by the separation surfaces image gives a gray image where:

- each non null intensity represents a voxel of a separation surface,
- the intensity value is equal to the half distance in voxels between the two closest objects.

Loading the project PorosityThickness-5-Mask.hx will complete this tutorial step (see Figure 4.62).
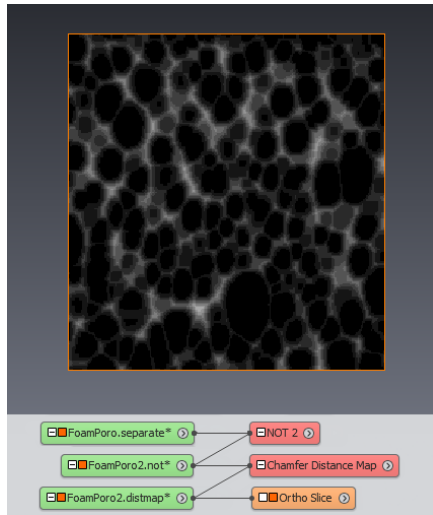
Figure 4.61: Distance map of material
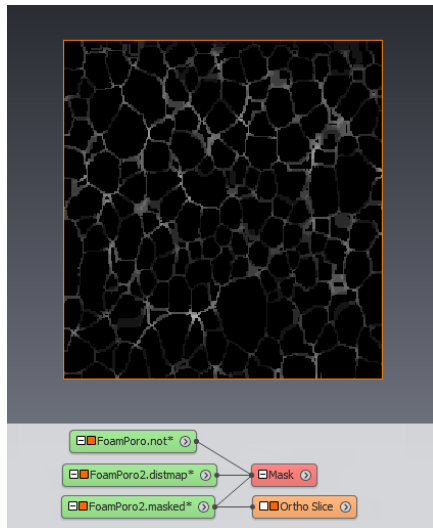
Figure 4.62: Distance map of the separation surfaces

4.5.4 Calculation of the Material Average Thickness

The average thickness of the material is given by the following formula:

$$\mu(Thk) = \frac{2 \times V_{size} \times \Sigma_i}{NbVoxSep}$$

where:

- V_{size} = voxel dimension in micrometers
- Σ_i = sum of the voxel intensities in the distance map image of the separation surfaces
- $NbVoxSep$ = number of voxels in the binary separation surfaces image

The *Volume Fraction* module with *3D Interpretation* gives on the column "Label Voxel Count" the number of labeled voxels for each label in a 3D image. A binary image has only one label so "Label Voxel Count" returns $NbVoxSep$.

The *Intensity Integral* module with *3D Interpretation* gives on the column "Volume" the sum of the voxels intensities in a 3D image i.e., Σ_i.

Loading the project PorosityThickness-6-Thickness.hx will complete this tutorial step.

4.6 Watershed Segmentation

Due to artifacts associated with image acquisition and reconstruction, the commonly used segmentation method by simple thresholding may give inaccurate and even often wrong results, especially in phase transition regions.

- Correct or unique threshold cannot be easily or accurately determined with edges blurred by noise and partial volume effect. Partial volume effect is caused by resolution limits in image acquisition, which blurs the transition between phases and features, i.e., voxels do not map single homogeneous physical volumes.
- When segmenting more than two phases, a transition between high and low intensity phases may introduce artifacts with unwanted intermediate "coating" phase.
- Variations in illumination or intensity across image may lead to different thresholds on different regions.

The watershed technique provides an effective solution for these issues in many cases. See *documentation about watershed principle*.

In this tutorial, you will learn how to use watershed for segmentation using different Avizo tools:

- Interactive watershed tool in the *Segmentation Editor*.
- Segmentation of multi-material or multiple phases using the *Watershed Segmentation wizard*.

To follow this tutorial, you should have completed Avizo tutorial on *Image Processing and Analysis* and be familiar with basic manipulation of Avizo. Preferably, you should also be familiar with the use of the *Segmentation Editor*.

4.6.1 Segmenting sand pack with watershed tool in the Segmentation Editor

For this tutorial, we will use a subset image of a compacted silica sand sample, partially saturated with water (see Figure 4.63). This sample was acquired by X-ray micro-tomography with a voxel size of about 11.2 μm (data courtesy Mr. Felix Kim and Dr. Dayakar Penumadu, University of Tennessee, Knoxville, TN, USA).
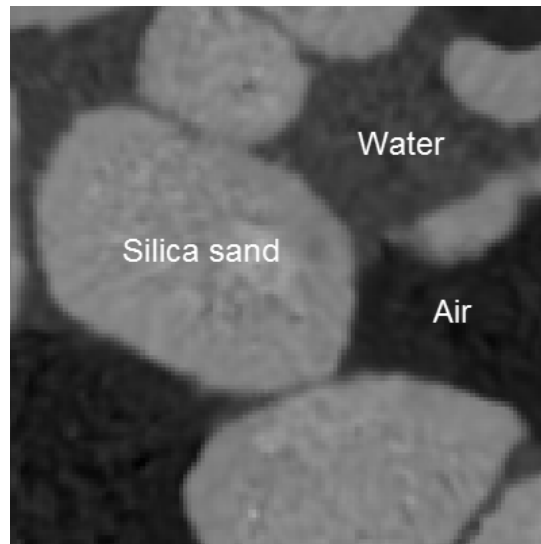
Figure 4.63: Compacted silica sand sample, partially saturated with water.

There are actually 3 phases that can be clearly distinguished in this data set:

- Air
- Water
- Silicate grains, with a few higher density areas in places

For this tutorial, we are interested here only in pores space vs. grains.

- Open `data/sandpack/sandpack128-filtered.am`. This image was obtained by applying the *Non-Local Means Filter* to raw image in order to reduce noise.
- Use *Edit New Label Field* to create and edit a label image with the *Segmentation Editor*.

- In *Materials* list, rename "Inside" as "Pore_Space". For this, double-click on item "Inside" or right-click to access material's popup menu.
- Add a new material "Grains": press the "Add" button below the material list, then rename it as above.
- In the bottom toolbar, select the *Threshold Tool*.
- Set Masking bounds to 0-6000, check *All slices*, and click on "Select Masked Voxels". You can see selected voxels in red, that will be assigned to "Pore_Space" phase.

The threshold may look too low and leave unselected areas, but a higher threshold might capture noise within the grains, or select voxels across the actual phase boundary. We want here a "safe" selection, which will be filled and expanded to the actual phase boundaries later on thanks to the watershed tool (see Figure 4.64).
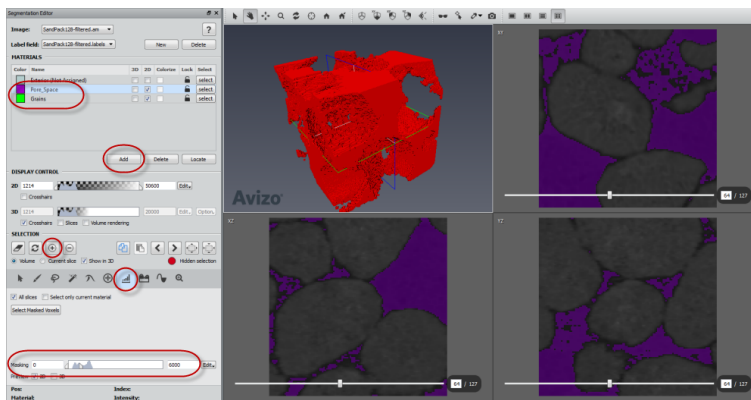
Figure 4.64: Roughly assign "Pore_Space" phase in Segmentation Editor.

- In the *Materials* list, select "Pore_Space", then in the *Selection* group, press the button with a large "+" or press key 'A'. The current selection voxels are then labeled as the "Pore_Space".
- Set bounds of *Threshold Tool* to 10000-51000 and click on "Select Masked Voxels". *All slices* should still be checked.
- Select "Grains" in *Materials* list, use the large + button in the *Selection* group or press key 'A' to assign the selection to "Grains" material (see Figure 4.65).

The labels we have defined will now be used as seed markers for watershed expansion.

- Select the *Watershed Tool*.
- In option *Landscape image:* press *Create a new gradient image*.

A gradient magnitude image is calculated using quick Canny method, that will provide the landscape
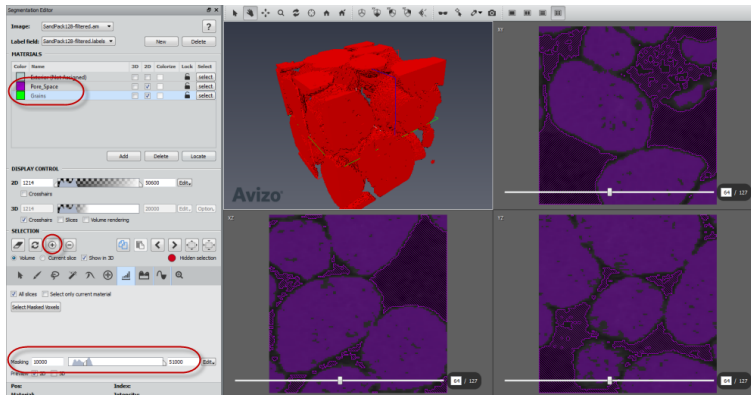
Figure 4.65: Roughly assign "Grains" phase in Segmentation Editor.

image controlling the expansion of markers.

Tip: Depending on your application, you might select instead any image previously loaded or created in your project, such as a smoother or more accurate gradient image, or a distance map for watershed separation (see *related tutorial in Avizo User's Guide*).

- One can choose the materials used as markers for watershed in the "Marker" column in the materials list, which appeared when you invoked the *Watershed Tool*. You can just leave all materials checked by default for now.
- Leave the option *Output catchment basins* to default value *side-by-side* in order to get contiguous labels instead of labels separated by exterior voxels.
- Press *Apply and create new label field* (see Figure 4.66).

A new label image is created with markers expanded to the edges of the landscape image (i.e., maxima of gradient magnitude gradient). This becomes the current label image being edited in the Segmentation Editor, instead of the original marker labels. The boundaries are fitting the optimal location in the transition between phase intensities, based on the seed markers and the gradient image.

Tips:

1. When not satisfied by result, you may easily cancel the watershed step by deleting the current label: this will take you back to the original marker labels, that you could then adjust or complete before applying again the watershed tool (see Figure 4.67).
2. You may also use the *Label field:* selector menu to simply go back and forth between markers and watershed results or different segmentation results.
3. You may also use the *Image:* selector menu to temporarily change segmented image used as background, then go back to actual image being segmented. This also allows the user to display
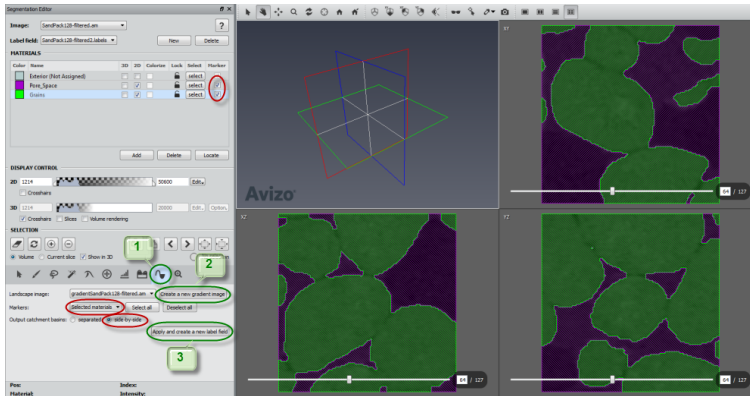
Figure 4.66: Apply Segmentation Editor watershed.
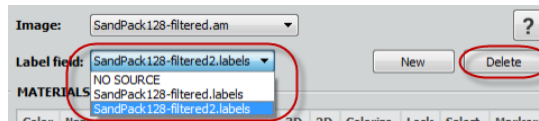
Figure 4.67: Deleting current label or switching to another one.

in the background of the labels the landscape image created using *Create a new gradient image*. You may need to adjust the data for better visualization.

4. You can notice that grains may not be separated as one could expect: that could happen because of truly consolidated grains with an intermediate intensity phase in some case, or because of image resolution and partial volume effects. This could then require improving the seed markers (see for instance *TopHat* tool) and/or gradient (see *Image Gradient* module), or this could be solved by separating particles using for instance *Separate Objects*.

5. After using the watershed tool, you may want to polish the segmentation for instance by using *Smooth labels...* or *Remove islands...* in Segmentation menu (illustrated in the *tutorial about advanced meshing*).

4.6.2 Segmenting multiphase using the Watershed Segmentation wizard

For this tutorial, we will use the data set chocolate-bar.am.

- Open data/tutorials/chocolate-bar.am.

Auto-Thresholding

This dataset contains regions with mousse, caramel, chocolate and air.

One could try first segmenting 3 phases (air, mousse filling, chocolate) using for instance *Multi-Thresholding*, or the Segmentation Editor. We will try using *Auto Thresholding*: this module analyses the image histogram to guess 1 or 2 threshold(s) separating voxels classes statistically. This module provides a quick way to segment images automatically.

- Attach *Auto Thresholding* to `chocolate-bar.am`.
- Set the module configuration *Type* to *Auto Segment 3 Phases*. The *Criterion* port should be set to *factorisation* (also known as Otsu method).
- Press *Apply* to create a label image.
- Attach an *Ortho Slice* to the label image.
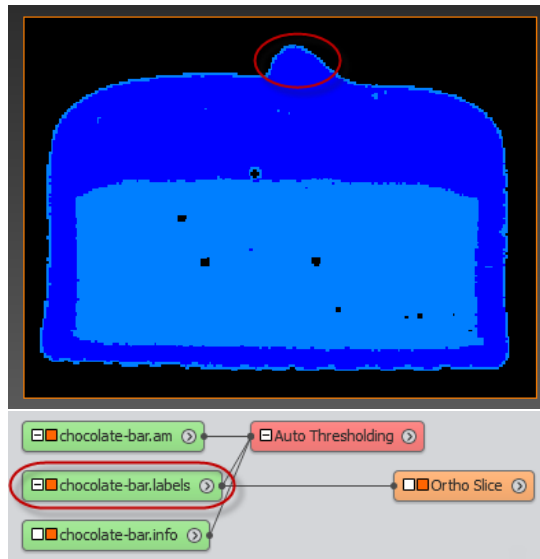
Figure 4.68: Automatically segmenting the image with *Auto Thresholding*.

A mousse layer (light blue) seems to surround the chocolate topping (dark blue): the mousse between the exterior and the chocolate is actually an artifact due to partial volume effect (see Figure 4.68 and Figure 4.69). Adjusting the thresholds will not solve that issue. As illustrated in figure 4.70, the image gradient gives useful hints on actual boundaries. Instead of using direct intensity thresholding, a more robust method is shown in the next section that takes advantage of the image gradient.
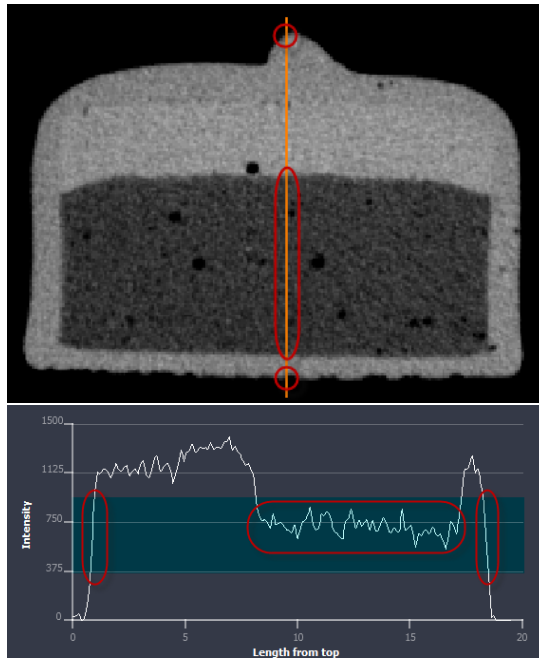
Figure 4.69: A simple thresholding selects unwanted "coating" voxels in the transition from high to low intensities due to partial volume effect, as shown by this intensity profile using a *Line Probe* module.
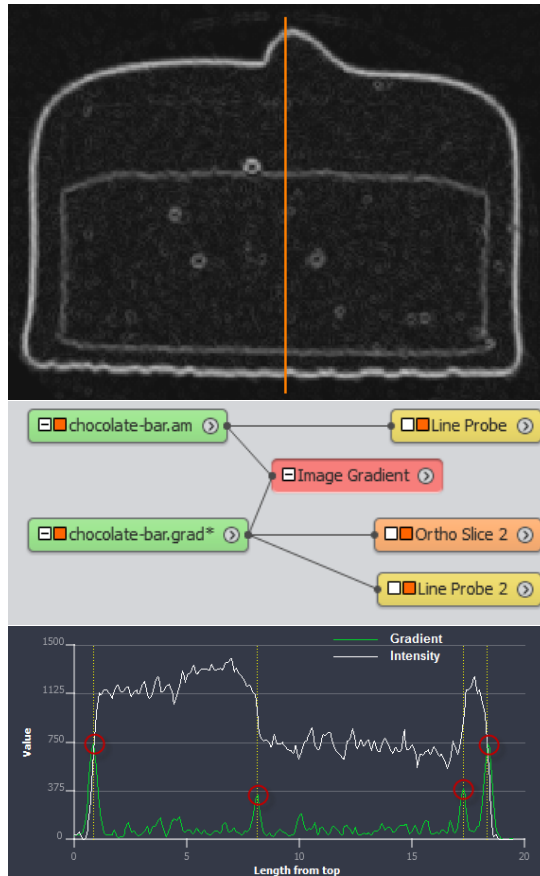
Figure 4.70: The peaks of *Image Gradient* help to better locate the phase boundaries

Watershed Segmentation Wizard

We will use the *Watershed Segmentation wizard* module to solve this. This wizard is a script module that will simply guide you step by step through the following segmentation process:

1. Define "conservative" thresholds defining initial seed markers for air, mousse and chocolate,
2. Calculate image gradient magnitude mapping material edges "sharpness",
3. Mark sharp edges between air and steel to prevent "coating effect" by masking the seed markers,
4. Complete watershed expansion of the seed markers towards objects edges.

- Remove *Auto Thresholding* module and its result data, leaving only `chocolate-bar.am` in the Project View.

- Attach *Watershed Segmentation* wizard module to `chocolate-bar.am`.

The wizard module being selected in the Project View, you can see in the Properties panel the module's ports showing the current step with possible options and parameters (see Figure 4.71). You can go back and correct previous actions at any step.
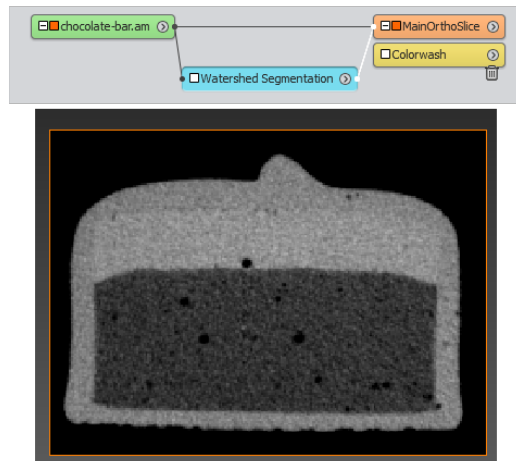
Figure 4.71: The *Watershed Segmentation* wizard at step 1.

Tip: The wizard keeps intermediate data for going back in the steps. For large data that might exceed available memory, you may want to show and remove intermediate when going to next step (use object menu "show"). Unrolling the steps will then no longer be possible.

- At any time you can move the slice or change its orientation. The current view is XY slice number 147.
- At first step, you need to input the number of separate materials or phases to segment: let the number of phases to 3 for air (exterior), caramel and chocolate.
- Press the *Apply* button (in the port *Action*).

The next step is intended to allow attaching optionally a pre-computed gradient image (port *Gradient*)

- Simply press *Apply* to trigger computation of gradient magnitude for the images.

At the next step "Threshold Gradient Magnitude", you can use a slider to adjust a threshold to mark sharp edges based on gradient obtained at previous step.

- Uncheck auto checkbox in order to show *Gradient Threshold* port

- You can set the *Gradient Threshold* value to 707 for a masking the air-chocolate transition areas.
- At this point, you may verify proper marking by changing the slice number or orientation.
- You may select the module *MainOrthoSlice* and adjust the colormap range of the edges area or transparency for better seeing the edge areas (see Figure 4.72).
- Then select again the *Watershed Segmentation* module (see Figure 4.73).
- Press *Apply* in *Action* port of Watershed Segmentation.
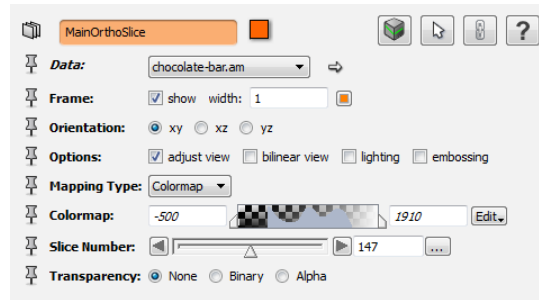
Figure 4.72: The *MainOrthoSlice* module properties.
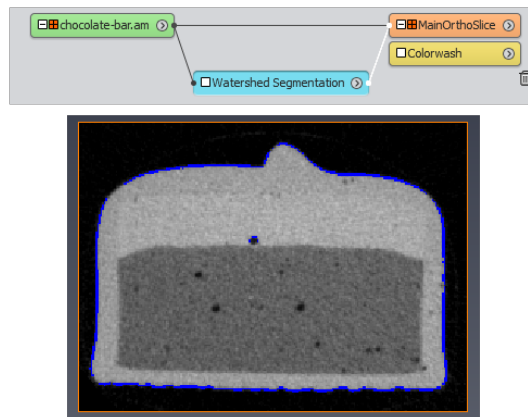
Figure 4.73: The *Watershed Segmentation* wizard at step 3.

The three next steps "Threshold Phase..." allow you to define marker labels for each material.

First, you need to set a marker range for air (phase 0). We do not attempt fitting accurately to the object edge at this point, but rather safely avoid areas when the actual material transition might occur. The holes in the mousse should be marked in the phase 0. Otherwise, they will be included in the mousse area.

- You can use 0-435 as range (see Figure 4.74).
- Press *Apply* to complete this step.
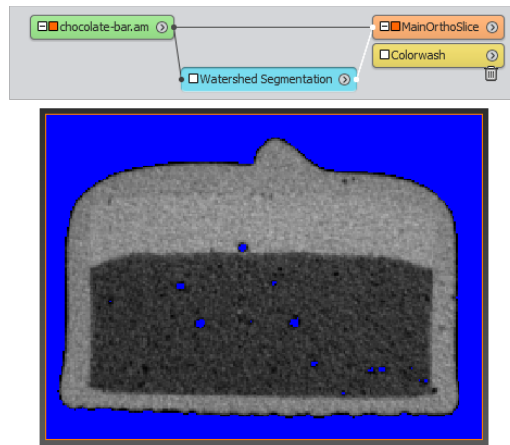
Figure 4.74: The *Watershed Segmentation* wizard at step 4.

In the next step, you can set the range for phase 1 (mousse).

- You can use 629-674 as range here. Again, the goal is to mark inner areas of the given material. (see Figure 4.75). If you select a lesser lower range, areas between the air and the chocolate will be selected, and it will result in the same artifact than the one introduced in Figure 4.68.
- Press *Apply* to complete this step.
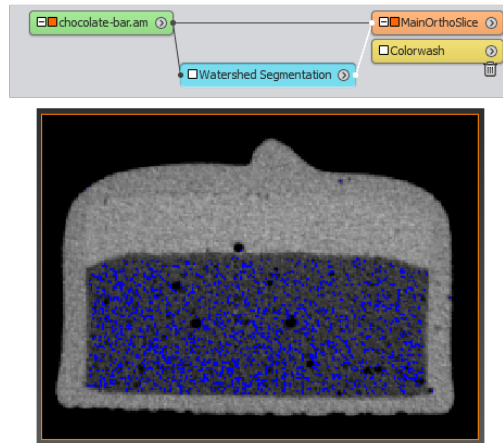
Figure 4.75: The *Watershed Segmentation* wizard at step 5.

Finally, in the next step, you can set the range for phase 2 (chocolate).

- You can use 1041-1910 as range here (see Figure 4.76). The mousse area should be avoided as much as possible.
- Press *Apply* to complete this step.
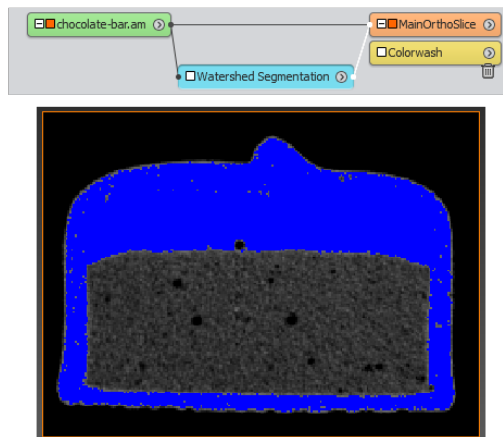
Figure 4.76: The *Watershed Segmentation* wizard at step 6.

In the next step, you can trigger watershed computation and get the final label image result.

- Simply press *Apply* to trigger computation (see Figure 4.77).
- You may verify the result by changing slice number or orientation, and possibly go back to previous steps to modify some parameter.
- You can then remove the *Watershed Segmentation* module, which will clean up also the ancillary display modules.
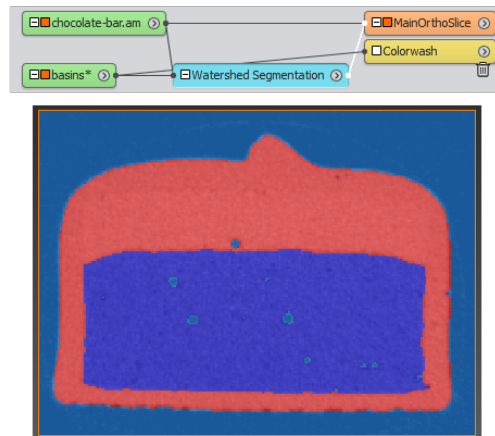
Figure 4.77: The *Watershed Segmentation* wizard completed.

You may want to get rid of the air phase that has been segmented as label 1 by the wizard. To do this, you can simply use the *Subtract Value* module with value 1, or the more general *Arithmetic* module with expression "A-1" (see Figure 4.78).
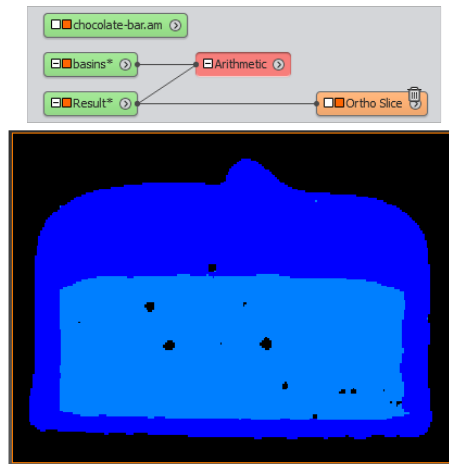
Figure 4.78: Remove the air phase using *Arithmetic* module.

More segmentation hints

The watershed technique shown above can be used in many different workflows to achieve more demanding or specific segmentation depending on desired result.

For instance, in the chocolate-bar dataset, one may slightly distinguish a 4th phase made of caramel. The image noise prevents to directly mark the proper region by thresholding. However applying first a smoothing filter such as Non-Local Means Filter can help further segmentation. A simple trial using the Watershed Segmentation wizard (4 phases, gradient threshold 100, phase marker thresholds 0-435, 630- 750, 800-1210, 1250-1459) leads to the result on figure 4.79 .
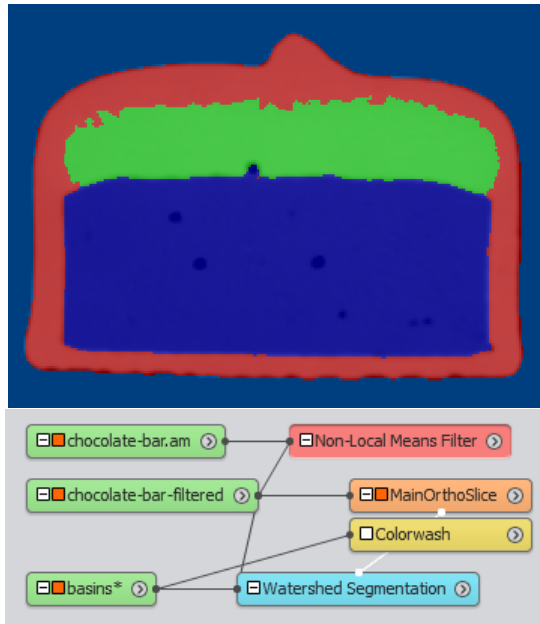
Figure 4.79: Example of Watershed Segmentation wizard use after *Non-Local Means filter*.

If not satisfying, the segmentation editor can be used to correct or refine the segmentation in a few steps. It is possible to use the segmentation editor to adjust interactively marker regions, and then apply the watershed tool as shown in previous tutorial.

As another example, small pores could be separately segmented, for instance using the Interactive Top-Hat module, and combined with previous segmentation using the Segmentation Editor or *Arithmetic* module.

An example result is `data/tutorials/chocolate-bar-labels-4-phases.am` (See Figure 4.80).
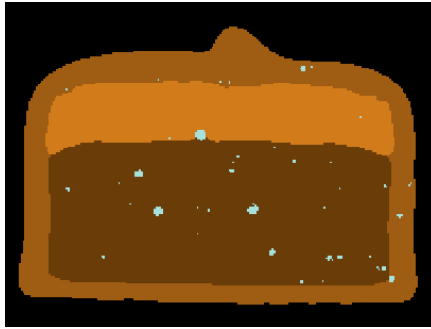
Figure 4.80: Example of 4 phases result

4.7 More about Image Filtering

It is often necessary to reduce image noise or artifacts and enhance features of interest before segmentation. Digital image filters are tools used to enhance image or highlight image features. These tools can be based on sophisticated algorithms and may have variable effects and performance depending on input image and control parameters, which may require experimentation in order to achieve the desired results.

In this tutorial, you will learn how to:

1. Distinguish the different types of image filters available in Avizo, and the main filters typically used.
2. Use image filters effectively to tune parameters and compare results.
3. Compose application of several image filters.

To follow this tutorial, you should have read the first tutorial chapter 4.1 - Getting Started with Image Processing and Analysis and be familiar with basic manipulation of Avizo. In particular, in the Getting started tutorial, you can see how to apply an image filter and the complete quantification workflow afterwards.

4.7.1 Choosing image filters

For convenience, image filters are sorted in categories according to their main function. You can browse filter categories in the explorer panels of the object popup, within the top category *Image Processing*. You can also use the search tool of the object popup to quickly retrieve a filter by name. In the sections below the different categories are briefly described, and a selection of image filters is suggested along with some tips.

4.7.1.1 Smoothing

This is the most important category for preparing data for image segmentation. These filters help smooth noisy images. Some users call these "denoising" filters. They can effectively reduce noise, but may require careful use to not alter information contained in the image, especially for quantification purposes. The most commonly used smoothing filters in Avizo are:

- *Median Filter* - a basic filter preserving edges. Very effective on salt-and-pepper noise (scatter dots).
- *Bilateral Filter* - filter balancing smoothing and edge preserving (in particular sharp angles). It requires an Avizo license.
- *Non-Local Means* (GPU accelerated). This filter is extremely effective on noisy data while preserving edges (best with white noise). It is generally the first choice for noisy images. However, it can be very time consuming. In practice, it is used in 2D mode. Reducing the search window will decrease computation time, but it may also reduce the smoothing efficiency (depending on the noise distribution). Edge enhancement should preferably not be applied before this filter. It requires an Avizo license.
- *Edge-Preserving Smoothing* - a diffusion filter preserving edge
- *Anisotropic Diffusion* - a GPU accelerated diffusion filter. It requires an Avizo license.
- *Curvature-Driven Diffusion* - a diffusion filter that may better preserve thin structures. It requires an Avizo license.

Tip: Make sure that the filter parameters such as contrast thresholds for edge preserving diffusion are set according to your data range. Default ranges are usually intended for 8-bit data, and need to be dramatically increased for higher dynamics of 16-bit images.

4.7.1.2 Sharpening

These filters help reinforce the contrast at edges and make details appear sharper. Commonly used sharpening filters are:

- *Unsharp masking*
- *Delineate* - also acts as smoothing filter. It requires an Avizo license.

4.7.1.3 Edge detection

These filters highlight boundaries between different materials or phases. They can be used, for instance, to directly extract feature contours and edges, or in watershed-based segmentation.

(See tutorial 4.6.1 - Advanced segmentation).

Here are the most commonly used modules in that category:

- *Sobel Filter* - quick basic edge detection, that can be used as approximation of gradient magnitude

- *Image Gradient* - comprehensive module supporting quick approximation (Canny) or noise reducing gradient (Canny Deriche, Gaussian, Sobel, Prewitt)

4.7.1.4 Frequency domain

These filters operate by transformation in frequency domain.

- *FFT* - Fourier transform, basis module for many image filtering techniques
- *Deconvolution* - specific module for deconvolution of 3D light microscopy images

4.7.1.5 Grayscale transforms

Modules in that category are not strictly speaking image filters. Filters mentioned above usually operate on pixels by examining a neighborhood of intensity values around each pixel. Grayscale transforms independently act on pixels, i.e., without considering neighboring pixel values. The following modules can be useful to correct globally image grayscale or shading:

- *Shading Correction*, *Shading Correction Wizard*, *Correct Z Drop*, *Background Detection Correction* - these modules can help to compensate non-uniform background in images
- *Match Contrast* - adjust image dynamics according to a reference. It requires an Avizo license.

4.7.2 Tuning image filters

It is always a good practice to first adjust a processing workflow on a limited subset of the data. Image filters do not make an exception, especially since it is very useful to adjust interactively parameters with visual feedback. Here are some methods that can help to adjust the image filters faster:

- *Crop Editor* - remove useless parts of your data
- *Extract Subvolume* - extract a subset for trials
- *Slice* - apply some image filters on the displayed slice
- Image filters 2D/3D interpretation port - 2D processing (per slice) is generally significantly faster and may give similar results
- Image filters kernel size, number iterations, window size - filter performance is often dependant on such parameter
- *Filter Sandbox* - preview filter effect on an image region. It requires an Avizo license.

Here is how to use the Filter Sandbox module. The Filter Sandbox can be very helpful to choose a filter and adjust its parameters. Note however that this convenience script module proposes only a selection of the most commonly used filters, among the filters available in Avizo.

- Start a new project (Ctrl+N) and open `data/tutorials/chocolate-bar.am`.
- Attach a *Filter Sandbox* module. You can find it in the right-click object popup, in Image Processing category, or typing in the search field some letters of *Filter Sandbox*.

An Ortho Slice is displayed in the 3D viewer, with an overlaid preview box surrounded by a dragger with blue tabs.
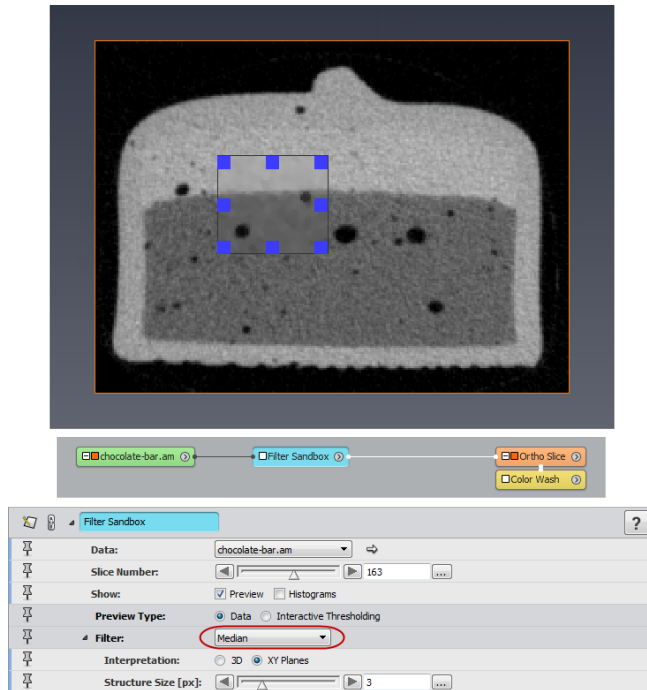
Figure 4.81: Starting Filter Sandbox

- In the properties panel, set the *Filter* port to *Median*. You can see that the filter is applied in the preview area.
- You may pick and drag or resize the preview box (first press the ESC key to set the 3D viewer in interaction mode). Keeping a small size at first may save time when trying filters.
- You can change filter parameters such as filter size. Note that when setting interpretation to 3D, the filter is applied to a 3D slab with a depth depending on parameters. 3D filtering can be significantly slower, even on a limited preview area.
- You can temporarily hide preview (port *Show*) to see original vs. filtered.
- Still using the port *Show*, you can display *histograms* calculated on original and filtered preview area, in order to see how the filter helps to better separate peaks. You can right-click in the

histogram plot to switch plot scaling between linear and logarithmic. A statistics summary is displayed in the Properties panel.

- Change *Preview type* to *Interactive Thresholding*: you can then adjust a threshold and verify the impact of the filter on such a threshold segmentation.

- Pressing the *Apply* button will apply the filter to the whole input image and create or update a result image.
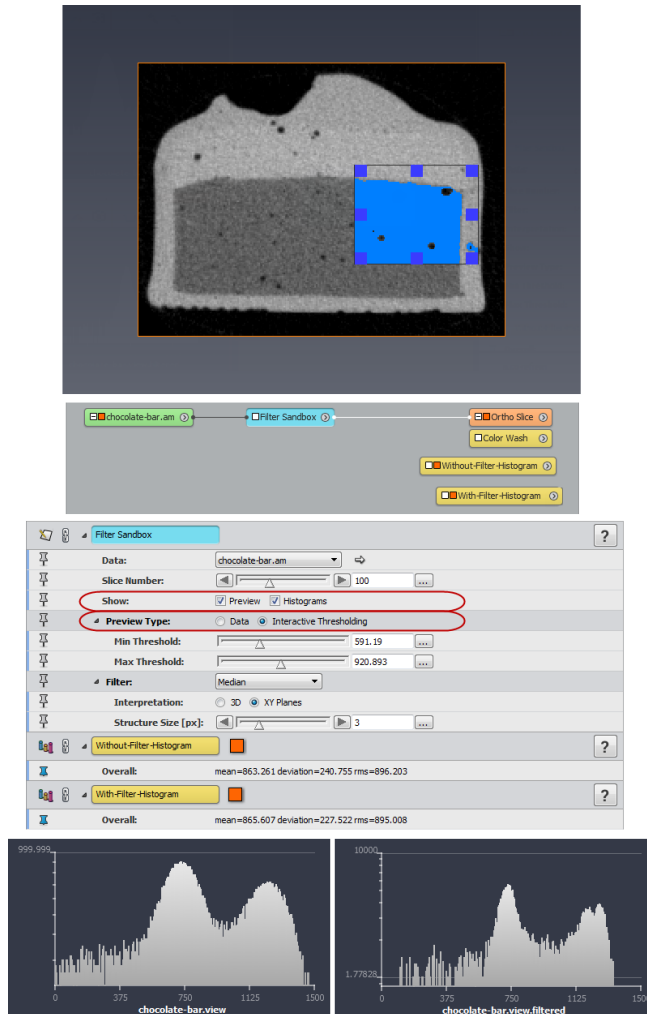
Figure 4.82: Using Filter Sandbox in Interactive Thresholding mode

For information about how to compare results of image filters, see also the tutorial chapter 6.2 Data fusion, comparing and merging data, in particular the example showing *how to synchronize views and display modules.*

4.7.3   Compositing image filters

Sometimes the best results may be derived by applying two or more filters sequentially. The *Slice* module allows the user to optionally apply a sequence of filters (2D) to the displayed slice: this can provide an easy way to try individual filters or filter combinations. Most of the image filters are not commutative so you should be careful to note your order of operations.

Another way to enhance your images is to make a composition of several filters. You may simply attach in a chain image filter modules to intermediate results. Tip: checking *auto-refresh* toggle will update the results upon change in inputs or any module port.

In some cases, a more complex combination may be useful. Some filters are known for detecting or preserving edges, other ones are used for smoothing or denoising. Filters may be contradictory and may not always be simply applied sequentially.

You will see below how to make step by step a filter composition that allows you to:

- preserve edges (limited edge smoothing),
- smooth uniform yet noisy areas

The following example illustrates possible techniques for filter combination, however it is not intended to show the preferred solution in a specific realistic case, nor a general solution. In general, filters such as Bilateral or Non-Local Means can achieve a good job at smoothing while preserving edges and should still be tried first.

The process is split in several steps:

- *Strong Smoothing using Median Filter*
- *Slight filter on the edges using Bilateral filter* (it requires an Avizo license)
- *Edge detection and masking using Sobel filter*
- *Compositing filters with mask*

4.7.3.1   Strong Smoothing using Median Filter

- Load `AVIZO_ROOT/data/tutorials/image-processing-advanced/Catalyst.am`.
- Attach a *Median Filter* module to the `Catalyst.am`.
- Set *Interpretation* to 3D and press *Apply*.
- Attach an *Ortho Slice* module to `Catalyst.filtered`.
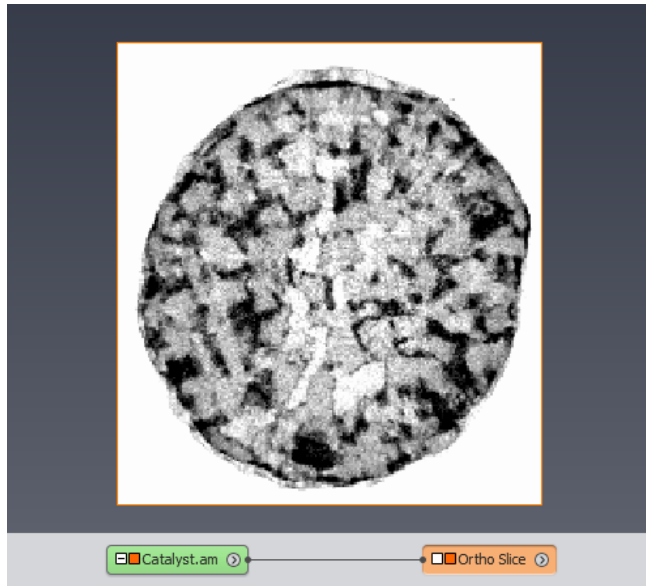
Figure 4.83: The initial image

Loading the project data/tutorials/image-processing-advanced/FilterCompositing-1-MedianFilter.hx will complete this tutorial step (see Figure 4.84).

Noise has been reduced in uniform areas of the image but edges are then very blurred.

4.7.3.2 Slight Filter on the Edge using Bilateral Filter

- Attach a *Bilateral Filter* module to the `Catalyst.am`.
- Set *Interpretation* to 3D, *Kernel* sizes to 3 and press *Apply*.
- Attach an *Ortho Slice* module to `Catalyst2.filtered`.

Loading the project data/tutorials/image-processing-advanced/FilterCompositing-2-BilateralFilter.hx will complete this tutorial step (see Figure 4.85).

Some noise has been removed from edges and edges are clearly preserved. Nevertheless, remaining noise is visible mainly on uniform areas of the image.

4.7.3.3 Edge Detection and Masking using Sobel Filter

- Attach a *Sobel Filter* module to the `Catalyst.am`.
- Set *Filter* to 3D and press *Apply*.
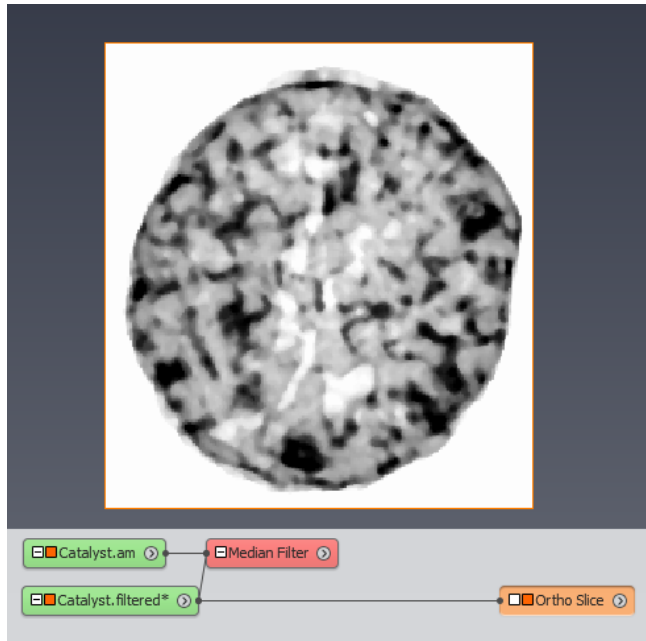
Figure 4.84: Median filter

- Attach an *Ortho Slice* module to `Catalyst-filtered`.

Loading the project data/tutorials/image-processing-advanced/FilterCompositing-3-SobelFilter.hx will complete this tutorial step (see Figure 4.86).

This filter highlights edges: edge areas are white, uniform areas are black and noisy areas are gray.

4.7.3.4 Compositing Filters with Mask

Now, you will use an *Arithmetic* module to basically compose the filtered image using the bilateral-filtered image (B) for the edge areas and the median-filtered image (C) for the uniform areas. Both images are blended linearly using the normalized Sobel-filtered image (A) as follows:

$$B * A + C * (1.0 - A)$$

(A) should be a normalized float image with a range from 0 to 1.

- Attach a *Convert Image Type* module to `Catalyst-filtered.am`.
- As `Catalyst-filtered.am` type is 8-bit unsigned byte image with 0-255 as range, set *output type* to *32-bit float* and *Scaling: scale* so that output range is 0...1 (use 0.00392157,
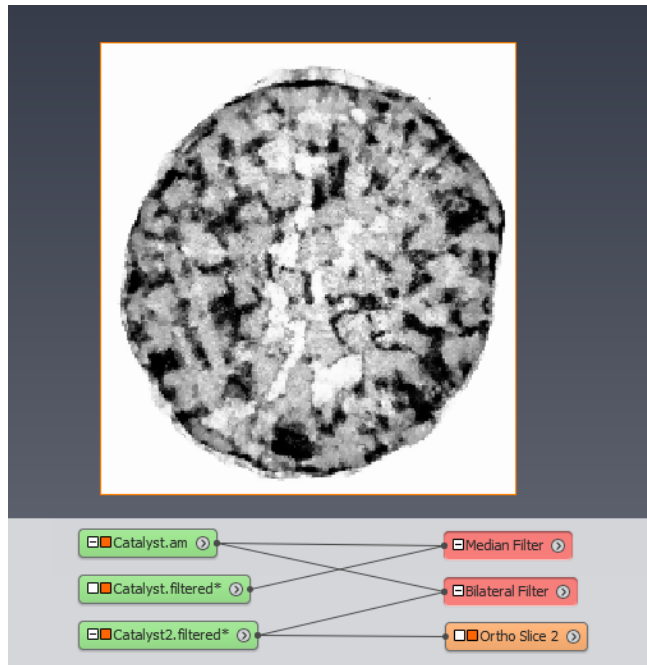
Figure 4.85: Bilateral filter

almost equal to 1/256). You could alternatively change A by A/256 in expression below.

- Press Apply to create `Catalyst-filtered.to-float`
- Attach an *Arithmetic* module to the `Catalyst-filtered.to-float`.
- Set *Input B* to `Catalyst.filtered` (result of Median Filter), *Input C* to `Catalyst2.filtered` (result of Bilateral Filter).
- Set *Expr* to *B*A + C*(1-A)* and press *Apply*.
- Attach an *Ortho Slice* module to `Result`.
- Set *Colormap* range to 0-255.

Loading the project data/tutorials/image-processing-advanced/FilterCompositing-4-Arithmetic.hx will complete this tutorial step (see Figure 4.87).

This composition takes advantages of both median and bilateral filters: edges are preserved and uniform parts are smoothed. You can improve this kind of compositing by:

- filtering the initial image to remove the circular acquisition artifacts,
- applying a shading correction and/or histogram equalization,
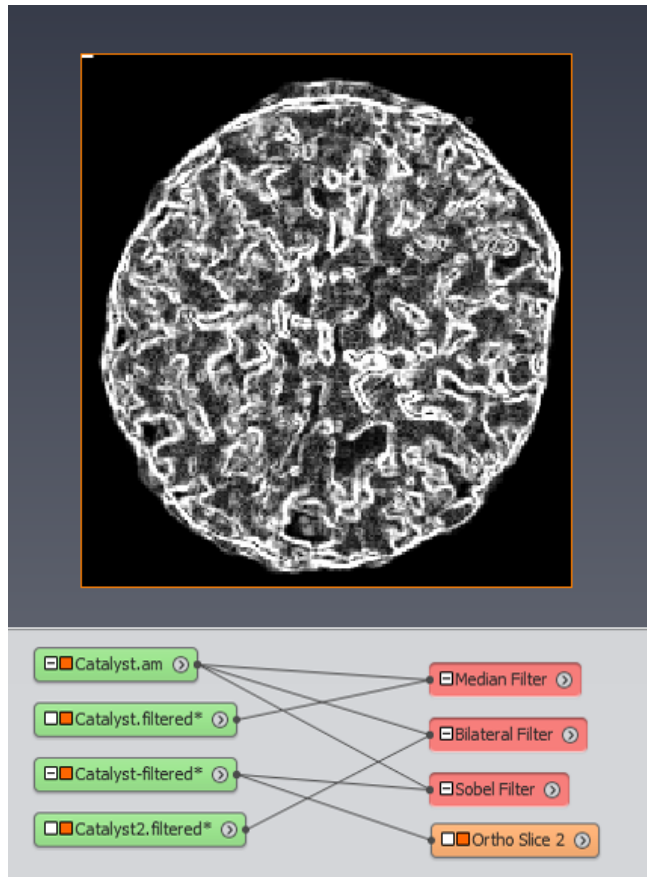
Figure 4.86: Sobel filter

- using some other filters,
- filtering the mask,
- tuning the compositing formula,
- adding a series of arithmetic operations based on new masks.
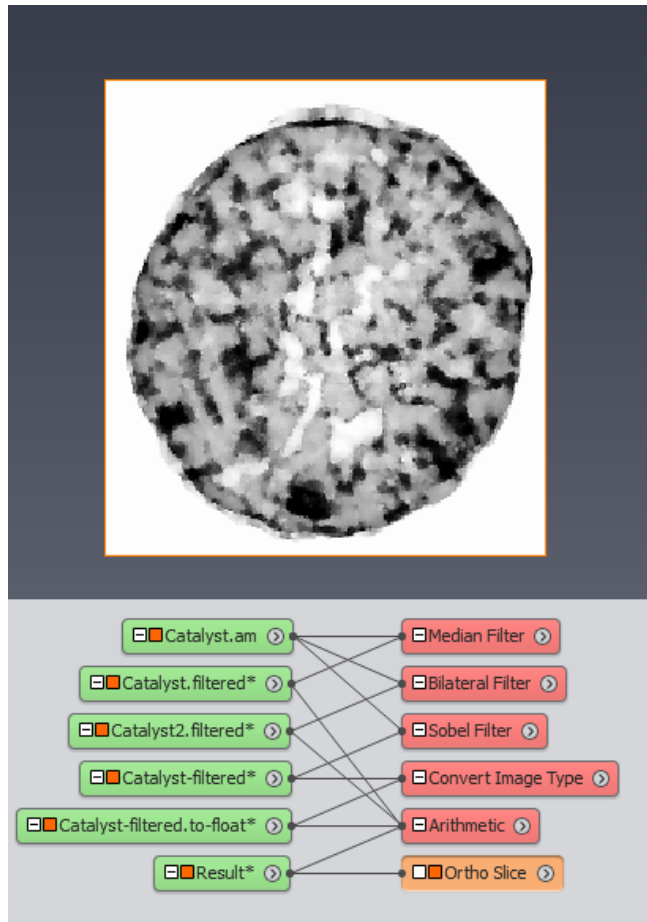
Figure 4.87: Filters with mask

4.8 More about label measures

4.8.1 The Measures Group Selection dialog

This dialog gives you the ability to manage several groups of measures. Those groups of measures can be modified independently. They are automatically stored in the user settings so that modifications are persistent when restarting the application. You can select any measure from the list of measures provided with the application. More details about each measure can be found in the *list of available measures*. Custom measures can also be created from the existing ones.
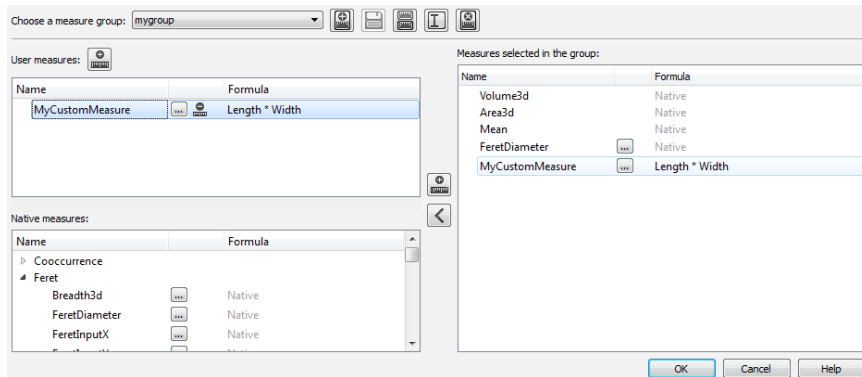
Figure 4.88: The Measures Group Selection dialog.

Managing the measure groups

The top part of the dialog displays the list of all measure groups and a set of tools. The list can be used to browse the measure groups and select a group.

When the dialog is opened from a module such as *Label Analysis*, the group selected in the measure selection port is directly displayed in the dialog. On the other side, when the dialog is validated, the measure selection port is updated with the measure group selected in the dialog.

With the measure group tools, you can:

- create a new empty group,
- save a group loaded from a project file or a script (groups created from GUI are automatically saved),
- copy an existing group into a new one,
- rename an existing group,
- remove an existing group.

The following four default groups are not editable (Yet you can copy it with a new name and edit it).

- *basic* group, used for 3D images, contains the measures *Volume3d*, *Area3d*, *BaryCenterX/Y/Z* and *Mean*.
- *basic2D* group, used for 2D images, contains the measures *Area*, *BaryCenterX/Y* and *Mean*.
- *Standard Shape Analysis* group contains standard shape parameters measures.

Selecting measures

Left panels list user and native measures. The right panel lists the measures selected in the current group. To add a measure to the selection group or remove it, just double-click on it in the list. To

add or remove several measures at once, select them and use the central buttons [>] and [<]. The <Delete> key shortcut is also available to remove the selected measures.

4.8.2 User measures

Custom measures can be created by combining existing measures in a mathematical formula. To create a new measure, click on the icon above the list of user measures. After prompting for a new measure name, this will open the Measure Editor.

Once created, new custom measures are listed in the user measures list. Some tool buttons are available next to each user measure to :

- re-edit the measure formula
- remove it from the list
- save it if the user measure has been loaded from a project file or a script (user measures created from GUI are automatically saved)

The Measure editor

All user measures are listed in the top drop-down menu. The formula area is updated according to the selected measure. The selected measure can be copied, renamed or deleted with the tool button next to the user measure list. Note that a user measure should be explicitly removed from all measure groups before being deleted.

The main part of the dialog deals with the editing of the selected measure. The first parameter is the unit dimension of the result values generated by this measure. This unit dimension will be combined with the unit of the analysed label image to determine the exact unit of the output result values. If the measure dimension is *Area* and the coordinate unit of the input image is cm, the unit of output values will be cm^2.

The second parameter is the measure's mathematical formula. The formula syntax supports a set of basic operators and mathematical functions to be used with existing measures. Supported operators are listed on the left of the formula area. Available functions and measures are listed below the formula area. As a shortcut, you can double-click or drag-and-drop keywords from the bottom list into the formula area.

Note: The working unit should be used in the formula. For more information about how working unit is set, see *Automatically determine or manually set the working coordinate units*

The formula definition is checked on-the-fly, and colorized according to its validity : green if the formula is correct, else red.

4.8.3 Configurable native measures

Some measures can be configured with additional parameters. Those measures are identified in the list of native measures with the [...] tool button. Click on this tool button to open the Label Measures
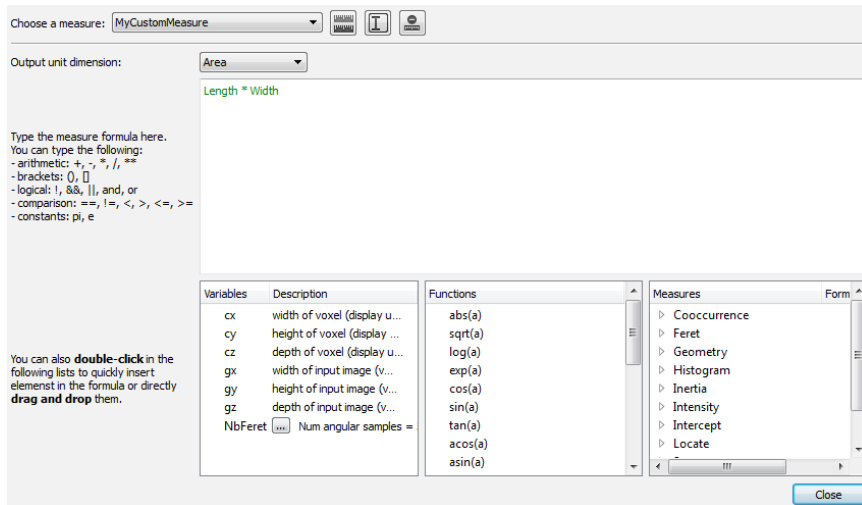
Figure 4.89: The User Measure Edition dialog.

Attributes Editor. There are five kinds of attributes that can be edited through this editor:

- *Feret angles* for *Feret 2D measures* which perform measurements on XY plane along a given numbers of diameters of each cell. Angles are uniformly sampled on the range [0,180]. By default, there are 10 angles, every 18 degrees.

- *Feret 3D angles* for *Feret 3D measures*, which perform measurements in 3D space around each cell. By default, 31 3D samples are used.

- *Co-occurrence directions* for measures based on the computation of a co-occurrence matrix to classify a given direction (dx,dy) in pixels pairs by their gray level. The co-occurrence matrix components are given by :

$$M(i,j) = number(\{x,y\}/I(x,y) = i \cap I(x + dx, y + dy) = j)$$

where I(x,y) is the image gray level at coordinates (x,y). This equation means that for a given pair (i,j), M(i,j) contains the number of pixels verifying I(x,y) = i and I(x+dx,y+dy) = j. This matrix is made symmetric and normalized such as :

$$\forall (i,j), M(i,j) = M(j,i) \text{ and } \sum_{i,j=1}^{N} M(i,j) = 1 \text{ with N the number of gray levels of the image}$$

These operations allow being independent to the image size and to hold properties on a direction and its symmetry.

- *Histogram parameters* for measures based on the computation of the gray level histogram of each label. Configurable attributes are in the range of gray values to consider and the size of the

More about label measures **204**
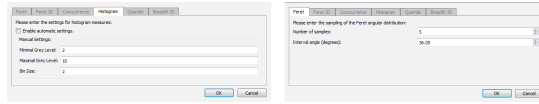
Figure 4.90: Edition of attributes of Histogram and Feret measures.
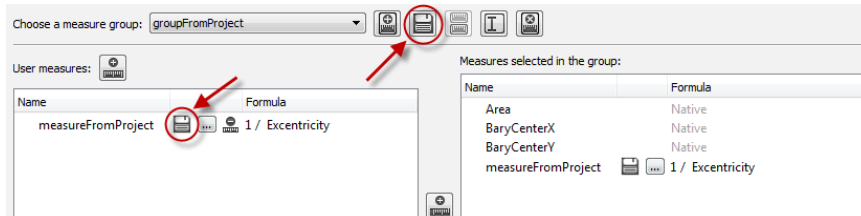
Figure 4.91: Unsaved measures loaded from a project.

bins to generate. By default, those settings are computed automatically.

- *Quantile values* for configurable HistoQuantile measures. See also *Quantile of an histogram*.

- *Breadth 3D sampling* for the Breadth 3D measures which search for the biggest orthogonal *Feret diameter* to the major axis found with *Feret 3D*. The *Breadth 3D* value defines the sampling which will be used on each orthogonal plane after *Feret 3D* has been used. The sampling of *Feret 3D* should be set to the desired value before using this measure.

IMPORTANT: Attribute values are common between measures. This means that when you edit the number of Feret angles for the measure FeretShape for example, all other measures based on the Feret angles will use this new value.

Note that only the attributes supported by the edited measure are enabled in the dialog.

4.8.4 About project backup

When you save your project, all the information required to redo the analysis are stored in the project file. This includes the measure group selected for the analysis, and the formula of the custom measures. Then, when this project is loaded on a another computer, those new definitions of measure group and custom measures are loaded in the new environment. Yet, the local storage of those new definitions on the new machine is optional, and a new *Save* button will appear near these unsaved items. There is no need to save the new items, they are fully functional and can be used until the application is closed. Note that if a custom measure or group are already defined on the machine but with different values, the definitions from the project are renamed to avoid conflict.

4.8.5 Scripting tips

To define custom measures inside a script, you can use the Tcl command *labelMeasure create*. This command is a simple alternative to the syntax used in the project file. Indeed project files use a more elaborate mechanism to avoid conflicts and duplications when loading several custom measures from a project, but are not adapted to scripting.

The Tcl command *labelMeasure create* handles conflicts in a simpler way: If the measure name is already used, a new name is generated, and this name is returned as the result of the command. Then this returned name can be used in dependent measure formulas instead of hard-coding that name which would replace the dependency by a wrong one. Example:

```
labelMeasure create mymeasure length "Volume/Area"
labelMeasure create mymeasure2 length "mymeasure+1"
LabelAnalysis measures setState mygrp mymeasure mymeasure2
```

This is fine in the console since the user can control the output in the console, but it may cause hidden dependencies issues when used in scripts if a measure name "mymeasure" already exists for instance. In scripts, prefer:

```
set msr [ labelMeasure create mymeasure length "Volume/Area" ]
set msr2 [ labelMeasure create mymeasure2 length "$msr+1" ]
LabelAnalysis measures setState mygrp $msr $msr2
```

Chapter 5

Tutorials: Surfaces, meshes, skeletons, modeling geometry from 3D images

Avizo allows the user to create geometric models derived from various types of data: surfaces from point sets, surfaces from 3D images, tetrahedral grid from surfaces, center line spatial graphs from 3D images. For an overview of related features, please refer to the *Features overview* section. The tutorials in this chapter introduce the following topics:

- *Surface reconstruction* - surface reconstruction from 3D images
- *Grid generation* - creating a tetrahedral grid from a triangular surface
- *Advanced Surface and Grid Generation* - extracting a geometric model from 3D images
- *Visualization and Analysis of 3D Models and Numerical Data*
- *Skeletonization* - analysis of network or tree-like structures in 3D images

5.1 Surface Reconstruction from 3D Images

By following this step-by-step tutorial, you will learn how to generate a triangular surface grid for an object embedded in a voxel data set. A surface grid allows for producing a 3D view of the object's surface and can be used for numerical simulations.

The generation process consists of these steps:

1. Extracting surfaces from segmentation results
2. Simplifying the surface

As a prerequisite for the following steps, you need a label image, which holds the result of a previ-

ous image segmentation. Please load the provided *motor.labels.am* data set from the *data/tutorials* directory.

5.1.1 Extracting Surfaces from Segmentation Results

Now we let Avizo construct a triangular surface of the segmented object.

- Connect a *Generate Surface* module to the *motor.labels.am* data.
- Press the *Apply* button.

The *Border* option ensures that the created surface be closed. A new data object *motor.labels.surf* is generated. Again, it is represented by a green icon in the Project View.

5.1.2 Simplifying the Surface

Usually the number of triangles created by the *Generate Surface* module is far too large for subsequent operations. Thus, the number of triangles must be reduced in a *surface simplification* step. In Avizo a *Surface Simplification Editor* is provided for this purpose.

- Select the surface *motor.labels.surf*. Since the Simplifier is directly modifying the surface data, you may want to duplicate your initial surface before proceeding so you can start again if you do not reach the expected result.
- Click on the Simplification Editor icon in the Properties Area.
- You can set the desired number of faces in the *Simplify* port, or preferred method, you can assign a maximum and minimum size for the triangles. Try 0.02 for max distance and 0.01 for min one. You can estimate this size by looking for instance at the bounding box min and max value of the data. Giving a too small number of faces or a too large max distance may result in an over-simplified surface with self intersecting faces.
- Testing for intersections can be necessary, particularly if you are simplifying a multi-material surface or a complex surface. In order to activate this, you can check the *intersection tests strategies* toggle in the *Options* port and set your intersection tests strategy in the *Intersections* port.
- Push the *Simplify now* button in the *Action* port.

The number of triangles is reduced from over 1.2 millions to 50K. The progress bar tells you how much of the simplification task has already been done.

To examine the simplified surface, attach a *Surface View* module to the *motor.labels.surf* data object.

The *Surface View* module maintains an internal buffer and displays all triangles stored in this buffer. By default, the buffer shows all triangles forming the boundary to the exterior. If you change the selection at the *Materials* port, the newly selected triangles are highlighted, i.e., they are displayed using a red wireframe representation. The *Add* and *Remove* buttons cause the highlighted triangles to be added to
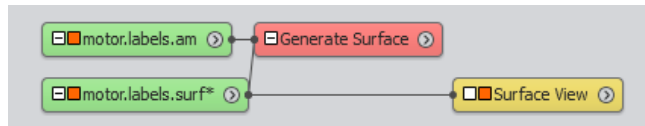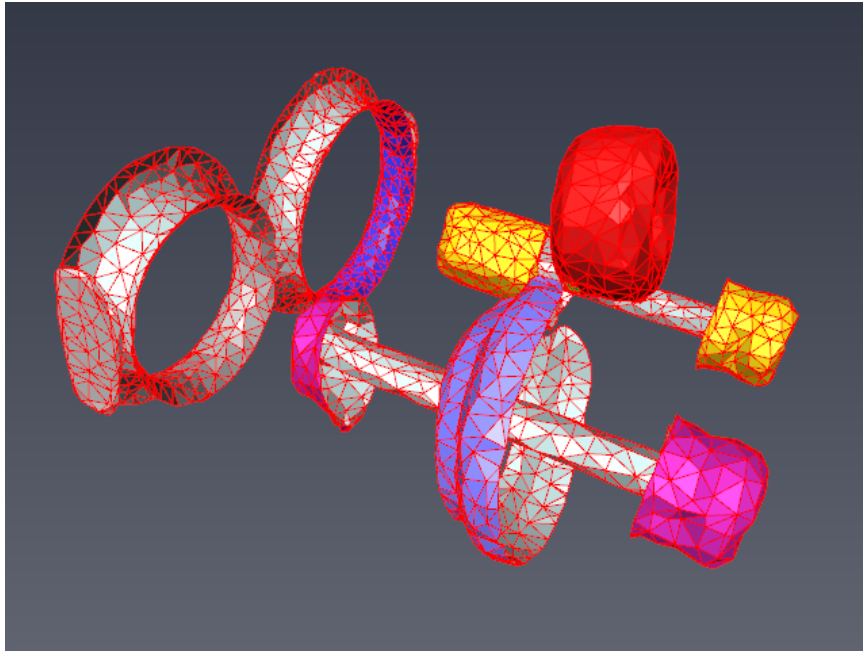
Figure 5.1: Surface representation of engine as triangular grid

or removed from the buffer, respectively. You may easily visualize a subset of all triangles using a 3D selection box or by drawing contours in the 3D viewer. Press the *Clear* button of the *Buffer* port to see the display shown in Figure 5.1.

5.2 Creating a Tetrahedral Grid from a Triangular Surface

By following this step-by-step tutorial, you will learn how to generate a volumetric tetrahedral grid from a triangular surface as created in the previous tutorial. A tetrahedral grid is the basis for producing various views of inner parts of the object, e.g., cuts through it, and is frequently used for numerical simulations.

The generation process consists of these steps:
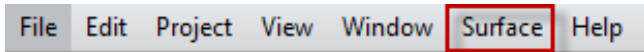
Figure 5.2: The surface menu

1. Simplifying the surface (see *Surface Reconstruction* tutorial)
2. Editing the surface
3. Generating a tetrahedral grid

As a prerequisite for the following steps, you need a triangular surface, which is usually the result of a previous surface reconstruction. Please follow the *Surface Reconstruction* tutorial and keep the last surface in the Project View.

5.2.1   Editing the Surface

As a second step of preparation for tetrahedral grid generation, invoke the *Surface Editor*.

- Select the surface *motor.labels.surf*.
- Leave the *Surface Simplification Editor* (Simplifier) by again clicking on the Simplification Editor icon.
- Enter the *Surface Editor* by clicking on the Surface Editor button in the Properties Area.

Automatically, a *Surface View* module will be attached to the *motor.labels.surf* surface. For details about that module see, its description.

When the *Surface Editor* is invoked, the *Surface* menu is added to Avizo's menu bar and a new toolbar is placed just below Avizo's viewer toolbar. The *Surface/Tests* menu contains 8 specific tests which are useful for preparing a tetrahedral grid generation. Each of the tests creates a buffer of triangles which can be cycled through using the back and forward buttons.

- Select *Intersection test* from the *Surface/Tests* menu. The total number of intersecting triangles is printed in the viewer window. Intersections shouldn't occur too often if toggle *fast* was switched off during surface simplification. In case they occur, the first of the intersecting triangles and its neighbors are shown in the viewer window.
- You can manually repair intersections using four basic operations: *Edge Flip*, *Edge Collapse*, *Edge Bisection*, and *Vertex Translation*. See the description of the *Surface Editor* for details.
- After repairing, invoke the intersection test again by selecting it from the *Surface/Tests* menu or by pressing the *Compute* button.
- When the intersection test has been successfully passed, select the *Orientation test* from the *Surface/Tests* menu. After surface simplification, the orientation of a small number of triangles may be inconsistent, resulting in a partial overlap of the materials bounded by the triangles. In case of such incorrect orientations, which should occur quite rarely, there is an automatic repair.

If this fails, the detected triangles will be shown, and you can use the above mentioned manual operations for repair.

Note: There are two prerequisites for the orientation test: the surface must be free of intersections, and the outer triangles of the surface must be assigned to material *Exterior*. If the surface does not contain such a material or if the assignment to *Exterior* is not correct, the test will falsely report orientation errors.

A successful pass of the intersection and orientation test is mandatory for tetrahedral grid generation. These tests are automatically performed at the beginning of grid generation. So you can directly enter the *Generate Tetra Grid* module (see below) and try to create a grid. If one of the tests fails, an error message will be issued in the console window. You can then go back to the *Surface Editor* and start editing.

- Select *Aspect ratio* from the *Surface/Tests* menu. This computes the ratio of the radii of the circumcircle and the incircle for each triangle. The triangle with the worst (i.e., largest) value is shown first, and the actual value is printed in the viewer window. The largest aspect ratio should be below 20 (better below 10). Fortunately there is an automatic tool for improving the aspect ratio included in the *Surface Editor*.

- Select *Flip edges* from the *Surface/Edit* menu. A small dialog window appears. In the *Radius ratio* area, set the value of the "Try to flip an edge if ratio is worse than" field to 10. Select mode *operate on whole surface*. Press the *Flip* button. All triangles with an aspect ratio larger than 10 will be inspected. If the aspect ratio can be improved via an edge flip, this will be done automatically. The console window will tell you the total number of bad triangles and how many of them could be repaired. Press the *Close* button to leave the *Flip edges* tool.

- Select again *Aspect ratio* from the *Surface/Tests* menu. Only a small number of triangles with large aspect ratio should remain after applying the *Flip edges* tool.

- Select *Dihedral angle* from the *Surface/Tests* menu. For each pair of adjacent triangles, the angle between them at their common edge will be computed. The triangle pair including the worst (i.e., smallest) angle is shown and the actual value is printed in the viewer. The smallest dihedral angle should be larger than 5 degrees (better larger than 10).

- For a manual repair of a small dihedral angle, proceed as follows: select the third points of both triangles (i.e., the points opposite to the common edge) and move them away from each other. For moving vertices, you must enter *Translate Vertex* mode by clicking on the first icon from the right on the top of the viewer window or by pressing the "t" key. If the viewer is in viewing mode, switch it into interaction mode by pressing the ESC key or by clicking on the arrow icon (the first icon from the top) on the right of the viewer window. Click on the vertex to be moved. At the picked vertex, a point dragger will be shown. Pick and translate the dragger for moving the vertex.

- In some cases, an edge flip might also improve the situation. Enter *Edge Flip* mode by clicking on the third icon from the right on the top of the viewer window or by pressing the "f" key. Switch the viewer into interaction mode. Click on the edge to be flipped.

- Select *Tetra quality* from the *Surface/Tests* menu. For each surface triangle, the aspect ratio of the tetrahedron, which would probably be created for that triangle, will be calculated. The aspect ratio for a tetrahedron is defined as the ratio of the radii of the circumsphere and the inscribed sphere. The triangle with the worst (i.e., largest) value is shown and the actual value is printed in the viewer. The largest tetrahedral aspect ratio should be below 50 (better below 25). If all small dihedral angles have already been repaired, the tetra quality test will mainly detect configurations where the normal distance between two triangles is small compared to their edge lengths. Again, the *vertex translation* and the *edge flip* operation are best suited for a manual repair of large tetrahedron aspect ratios.
- Leave the *Surface Editor* by again clicking on the Surface Editor button in the Properties Area.

Hint: In order to see the entire surface again, select the Surface View icon, then press its *Clear* button and press the *Add* button, then press the *ViewAll* button in the viewer toolbar.

5.2.2   Generation of a Tetrahedral Grid

The last step is the generation of a *volumetric tetrahedral grid* from the surface. This means that the volume enclosed by the surface is filled with tetrahedra.

Because the computation of the tetrahedral grid may be time consuming, it can be performed as a *batch job*. You can then continue working with Avizo while the job is running. However, for demonstration purposes, we want to compute the grid right inside Avizo.

- Connect a *Generate Tetra Grid* module to the *motor.labels.surf* surface by choosing *Compute / Generate Tetra Grid* from the popup menu over the *motor.labels.surf* icon. You can also choose to start by loading the motor.simplified surface from the tutorial directory.
- Leave toggle *improve grid* switched on and toggle *save grid* switched off at the *Options* port. The *improve grid* option will invoke an automatic post-processing of the generated grid, which improves tetrahedral quality by some iterations that move inner vertices and flip inner edges and faces. See the description of the *Grid Editor* for details.

  If toggle *save grid* is selected, an additional port *Grid* appears, where you can enter a filename. The resulting tetrahedral grid will be stored automatically under that name. If you want to run grid generation as a batch job, you must select the *save grid* option.
- Press the *Meshsize* button of the *Action* port. An editor window will appear. It allows you to define a desired mesh size, i.e., mean length of the inner edges to be created, for each region. For this you must enter the bundle of that region, and select parameter *MeshSize*. Then you can change the value in the text field at the lower border of the editor. There are some predefined region names in Avizo for which a default mesh size will be automatically set. Make sure that the default values are suitable for your application. If you are not sure about a suitable value, set the desired mesh size to 0. In this case, the mean edge length of the surface triangles will be used.
- Press the *Run now* button at port *Action*. A popup dialog appears asking you whether you really

want to start the grid generation. Click *Continue* in order to proceed.

Once grid generation is running, the progress bar informs you about the number of tetrahedra which already have been created. In some situations, grid generation may fail, for example, if the input surface intersects itself. Then an error message will occur at the *Console Window*. In this case, go back to the *Surface Editor* to interactively fix any intersections.

After the tetrahedral grid has been successfully created, a new icon called *motor.labels.grid* will be put in the Project View. You can select this icon in order to see how many tetrahedra the created grid contains. If grid generation takes too long, you may also load the pre-computed grid *motor.tetragrid.am* from the *data/tutorials* directory.

As the very last step, you may want to have a look at the fruits of your work:

- Hide the *Surface View* module by switching off its visibility toggle or removing it from the Project View.
- Attach a *Tetra Grid View* module to the *motor.labels.grid*.
- Select the *Tetra Grid View* icon
- Set the Materials port to *Material 3*
- Press the *Remove* button of the *Buffer* port.
- Select *outlined* in the port *Draw Style*

The *Tetra Grid View* module maintains an internal buffer and displays all tetrahedra stored in this buffer. By default the buffer contains all tetrahedra. You may easily visualize a subset of all tetrahedra using a 3D selection box or by drawing contours in the 3D viewer.

Similar to the *Surface Editor*, there is a *Grid Editor* which can be invoked by selecting the green icon of the tetrahedral grid and clicking on the pencil icon (first from the right in the title bar) in the Properties Area. The editor aims to select tetrahedra with respect to different quality of measures, e.g., aspect ratio, dihedral angles at tetrahedron edges, solid angles at tetrahedron vertices, and edge length. The editor contains several modifiers that can be applied for improving mesh quality.

5.3   Advanced Surface and Grid Generation

Extracting a geometric model from 3D images can be used for:

- Visualization of the boundaries of relevant features.
- Measurements based on extracted geometry.
- Numerical simulation of physical properties.

Avizo is used for geometry modeling from 3D images in a wide range of areas: industrial inspection and reverse engineering, digital rock physics, characterization of materials and design such as fuel cells, concrete, catalysts, metals, composites, carbon nanotubes, etc.
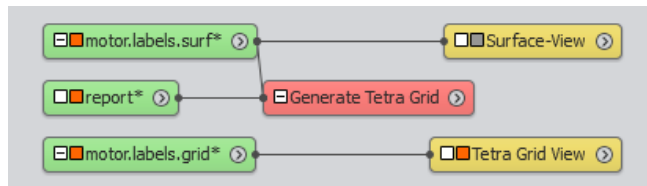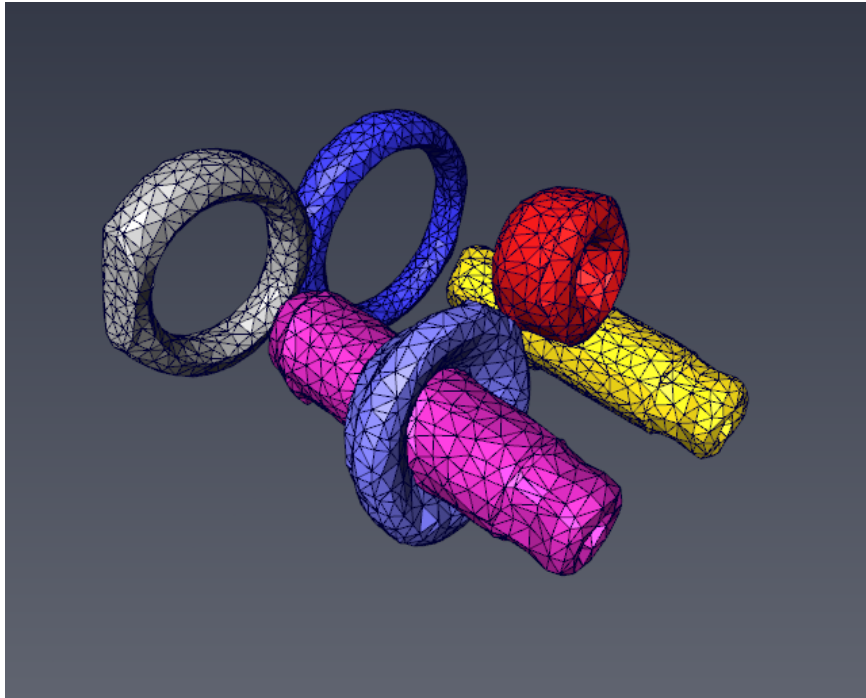
Figure 5.3: Volumetric representation of engine as tetrahedral grid

In this tutorial, you will learn how to:

- Extract 3D surfaces and grids with image-driven accuracy and consistency for single or multi-materials.
- Simplify and refine meshing for manageable mesh size and controlled mesh quality.
- Assign boundary conditions for simulation and export the resulting data.

This section has the following parts:

- *Getting started: a workflow from 3D images to surfaces and grids*

- *Adjusting segmentation for geometry extraction*
- *Generating surfaces with controlled smoothing*
- *Using the Simplification Editor*
- *Using the Surface Editor*
- *Remeshing and exporting surfaces*
- *Generating tetrahedral grid*
- *Assigning boundary conditions and exporting data*

You may skip the last two steps if you are only interested in surface extraction and not in grid generation.

You should be familiar with the basic concepts of Avizo to follow this tutorial. In particular, you should be able to load files, to interact with the 3D viewer, and to connect modules to data modules. All these issues are discussed in Avizo chapter 2 - Getting started.

To apply this tutorial to your data, the image filtering and segmentation steps may be critical: you will find important information about this in Avizo tutorials.

For the purpose of numerical simulation, Avizo can be complemented with Avizo XWind Extension in order to export and import data to standard and commercial software file formats such as Abaqus, ANSYS, CGNS, OpenFOAM, etc. Avizo XWind Extension provides advanced visualization and post-processing of numerical simulation results. See chapter 11 - Avizo XWind Extension User's Guide for more information.

Avizo actually provides support for different numerical simulation approaches: FEA/CFD solvers as illustrated in this tutorial, but also pre-processing for Pore Network Modeling (see chapter 5.5 - Avizo Skeletonization User's Guide), and direct simulation add-on for Avizo (see chapter 12 - Avizo XLabSuite Extension User's Guide) . Avizo XLabSuite Extension does not require geometry pre-processing and provides experiment simulation and effective property tensor calculation for properties such as absolute permeability, molecular diffusivity, electrical resistivity (formation factor) and thermal conductivity.

5.3.1 Getting started: a workflow from 3D images to surfaces and grids

Whether you want to extract a surface for visualization or to build a grid mesh suitable for simulation, you need to start by providing a 3D volume defining the different features, materials or phases. This mask can be built using different segmentation techniques in Avizo, using *Segmentation Editor* tools or segmentation modules, which provide from manual to fully automated algorithms to identify a particular part or material in the 3D data.

In Avizo, the segmentation process results in what is called a *label image*, each label identifying a particular material or phase in the data (see Figure 5.4). This process can be particularly important to capture accurately the required boundaries.

This tutorial assumes that the segmentation process has already been performed, as presented for

instance in tutorial chapter 4.6 - Advanced Segmentation.
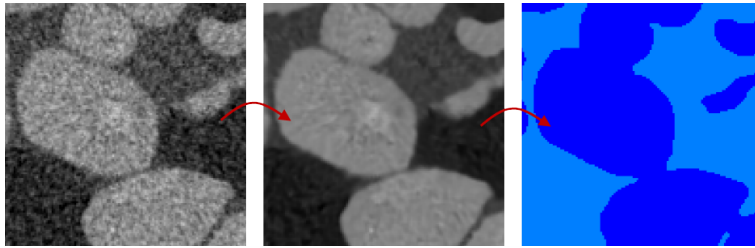
Figure 5.4: Filter and segmentation

Note: Two label images are available for this tutorial:

- `SandPack128.labels.am` is the data set from which are obtained all the pictures in this tutorial,
- `SandPack50.labels.am` is a smaller data set, extracted from the previous one, that you can use to complete the tutorial steps in a shorter time.

In order to build a 3D mesh made of 3D elements or cells, you need first to build 3D surfaces representing the boundaries of the volumes you want to mesh. In order to do this, you can use the *Generate Surface* module onto the label image. However, you may first want to clean up the label image by removing unwanted or useless features and by smoothing noise and voxel aliasing, in order to get an accurate, yet not unnecessarily complex surface.

5.3.2   Adjusting segmentation for geometry extraction

Some pre-processing can be done on the label image before going through the mesh generation process.

- With Open Data in File menu, load `SandPack128.labels.am` from the `data/sandpack` subdirectory in the Avizo installation directory. Remove the *Ortho Slice* that was created at data loading.
- In the label image property area, invoke the *Segmentation Editor* in order to access the tools necessary to clean up the labels (see Figure 5.5).

The *Label Filters* tools, available in the *Segmentation* menu of the menu bar, provide means to modify the current labeling.

There may be small islands in the label image, that are not meaningful for the material visualization or for the simulation to come.

- In the top-right viewer, use the cursor to move the XY slice to position 107.
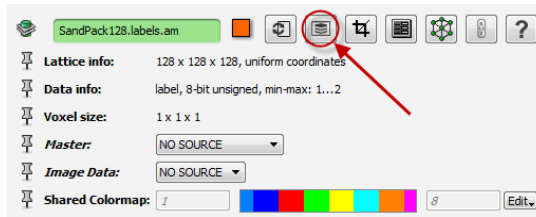
Figure 5.5: The *Segmentation Editor* button.

Looking at this slice, you can observe a small island that needs to be cleaned in order to generate a suitable surface. If you move the slider back and forth, you will see that the island is part of a small bubble and not of a large material. The *Segmentation Editor* provides a tool to detect and remove automatically such disconnected regions.

- In the *Segmentation* menu, select *Remove islands...* (see Figure 5.6).
- Select the *3D volume* option in order to remove the small bubbles.
- Press on *Highlight all islands* to get a preview of the bubbles that will be removed using the *Apply* button. You can see a preview of all islands to be removed in the top-left 3D viewer colored in red.
- Press *Apply*. The islands are then removed (see Figure 5.7).
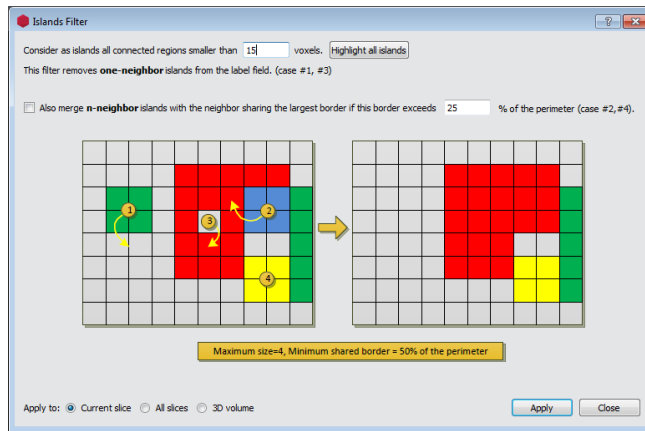
Figure 5.6: Islands filter dialog.

Notice that if you select *Current slice* or *All slices*, the size is defined as the area of contour in a 2D slice and you may remove thin material that may have a large and meaningful volume, possibly connected in 3D.

Advanced Surface and Grid Generation 217
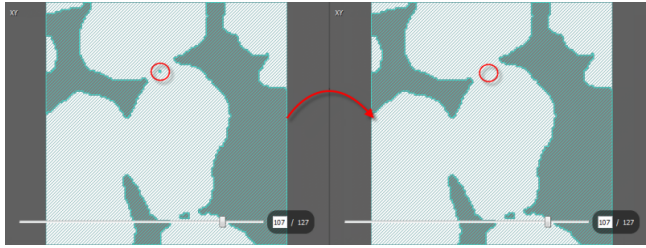
Figure 5.7: Removing islands

In order to correct contour roughness due to voxel aliasing, the *Smooth labels* tool can be used. This smoothing process can be iterated until the desired level of smoothness is reached. Smoothing slightly changes the labeling, but also assigns probability weights to voxels that can be used later on by *Generate Surface* for generating smoother surfaces.

- In the *Segmentation* menu, select *Smooth labels...* (see Figure 5.8).
- Select the *3D volume* mode.
- Press *Apply*.
- Switch back to *Project View*.
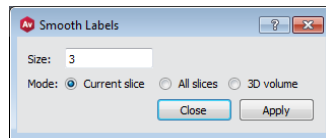
Figure 5.8: Smooth labels dialog

Note: Smoothing can alternatively be performed during the surface generation in the *Generate Surface* module as discussed in the next section.

Tip: The smoothing and islands removing tools will work along one direction unless "3D volume" option is selected in the tool's dialog box. The direction will then be the one corresponding to the currently selected view in the 3 orthogonal views of the *Segmentation Editor*. In some cases, you may want to repeat the smoothing or removing islands process in all of the 3 directions, instead of applying the tools to "3D volume" at once.

Tip: The *Segmentation Editor* is especially appropriate for fine control of the segmentation with visual feedback. For some workflows, you can also consider filtering the label image by using modules such *Label Analysis* and *Analysis Filter*, *Filter By Measure*, or *Axis Connectivity* (binary label image).

5.3.3 Generating surfaces with controlled smoothing

Generate Surface creates a 3D Avizo surface representing the boundaries of each material or phase. The *Generate Surface* module allows for generating a surface after applying or not smoothing operations. Depending on the shapes or features to be meshed, you may not want to apply smoothing or you may want to apply lighter smoothing, or simply use smoothing weights coming from the *Smooth label* tool of the *Segmentation Editor* (see Figure 5.9). All these options are available in the *Smoothing Type* port.
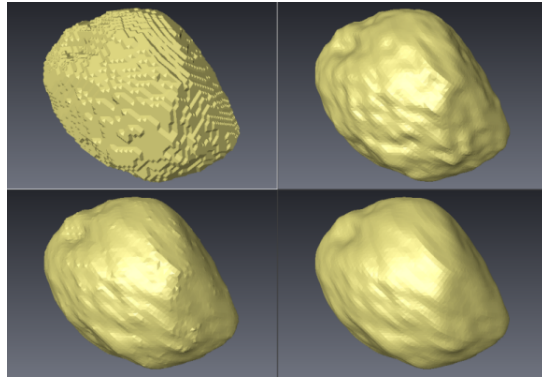
Figure 5.9: Results of the different kind of smoothing on a segmented grain: *None* (top left), *Existing Weights* (top right), *Constrained Smoothing* (low left) and *Unconstrained Smoothing* (low right).

Note: If you have a thin material, a strong smoothing can actually delete the material. Look at the *Generate Surface* documentation for a full description of all parameters.

- Connect a *Generate Surface* module to the label image (see Figure 5.10).
- Check that the *Smoothing Type* is set to *Existing Weights*. This option is only available if the label image contains probability weights information, which was created when applying smoothing in the *Segmentation Editor* in the previous steps.
- In the *Border* port, check the *Fit To Edges* option. This will cause the resulting surface to sharply fit to the edges of its bounding box.
- Press *Apply*.
- Attach a *Surface View* display module to the resulting surface (see Figure 5.11).

You now have a 3D surface object that consists of three patches corresponding to the number of materials identified in the label image (including the Exterior). The number of triangles resulting from the *Generate Surface* module may be very large and not suitable for visualization or simulation. The triangles generated through this first step may also not be suitable for simulation, considering the triangles aspect ratio for instance.

Figure 5.10: *Generate Surface* module parameters.
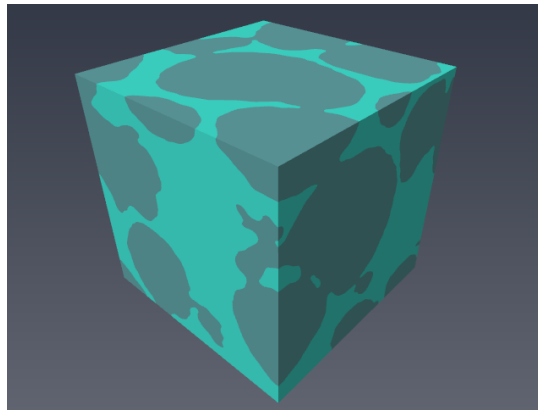
Figure 5.11: Surface generated from the two phases label image.

For the purpose of meshing the 3D volume from the generated surface, some quality checks for the surface are available with the *Generate Tetra Grid* module. This is useful to see if some improvements are required to perform the grid generation instead of directly running a grid generation that would fail or result in a bad quality grid.

- Connect a *Generate Tetra Grid* module to the surface (see Figure 5.12).
- Press on the *Check* button in port *Action*.
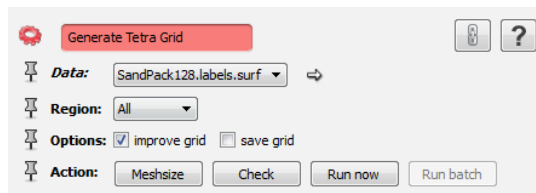
Figure 5.12: *Generate Tetra Grid* module.

A report opens in the *Tables* panel, containing quality information about the two materials patches. You can observe that the largest triangle aspect ratio is highly critical, i.e., superior to 30.

You can correct this by remeshing the surface so you have simulation quality triangles. First you will simplify the number of triangles through a decimation or surface simplification process.

5.3.4 Using the Simplification Editor

The *Simplification Editor* is available in the Property window of the surface object to be simplified (see Figure 5.13).
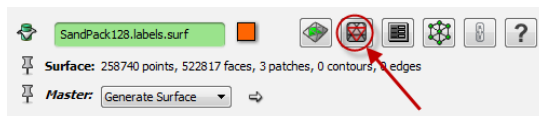
Figure 5.13: *Simplification Editor* button.

Note: The *Simplification Editor* will do in-place simplification, so the exact initial surface will be lost. As a consequence it is recommanded to duplicate the surface prior to simplification. A shortcut for this is to select the surface data object in the project view and press Ctrl-D. You may also want to use later on the shortcut F2 to rename a surface data for better reflecting its content.

- Open the *Simplification Editor*.

Look at the *Simplification Editor* documentation for a full description of all parameters. Default parameters will simplify the surface and generate large triangles in flat areas while generating smaller ones in areas of high curvature, so details are preserved. This is suitable to speed up visualization of a large surface, or to export a lightweight triangular surface file (see Figure 5.14).

For simulation purpose, most of the time, one wants balanced triangles, so instead of using the default option of decimating to a given number of faces, you can use the *max dist* field in the *Simplify* port to tell the editor what edge maximum length you are trying to achieve.

- Set the *max dist* to 2.
- Press *Simplify now* (see Figure 5.15).
- Close the *Simplification Editor*.

Tip: A good estimate of *max dist* can be to initialize this port to roughly 1% of the minimum dimension of the volume. You can use *Local Axes* to display the actual dimension of your volume.

Note: The resulting number of triangles should be targeted to be around twice the number of triangles targeted for simulation because in a next step (*the remeshing step*), this number will be reduced by half. So if a suitable surface for simulation is containing around 100,000 triangles, you need after this simplification step to have a surface twice as complex so around 200,000 triangles.
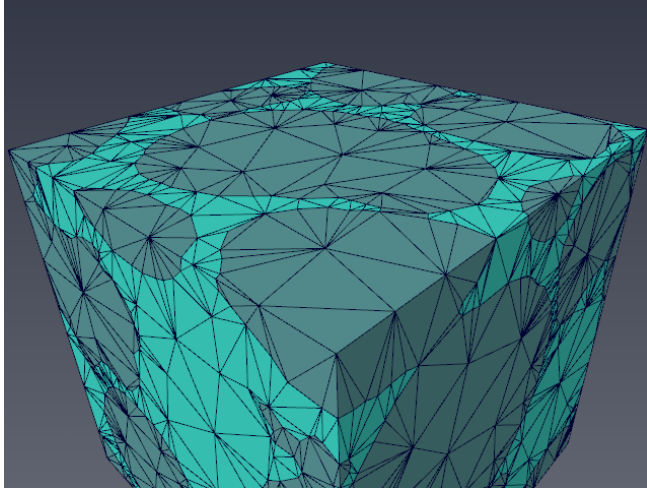
Figure 5.14: Rough simplification of the surface, displayed using Surface View's "outlined" draw style.
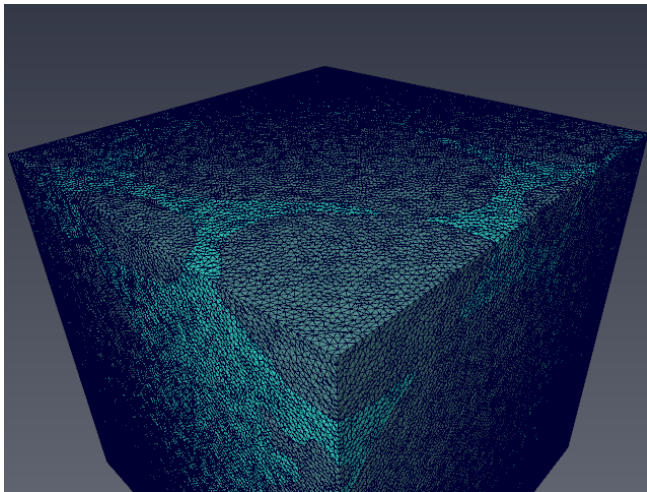
Figure 5.15: Simplified surface.

Notice that a too aggressive simplification can result in intersecting triangles.

At this point, you can perform quality checks again thanks to the *Generate Tetra Grid* module.

Now you have a simplified surface and you can use the *Surface Editor* to check for possible surface anomalies (such as intersections or bad aspect ratio) and to enhance triangle meshing.

5.3.5 Using the Surface Editor

The *Surface Editor* is available in the Property window of the surface object to be checked or cleaned (see Figure 5.16).

Note: Like the *Simplification Editor*, the *Surface Editor* will do in-place editing and the exact initial surface will be lost. While a number of surface editing operations can be undone, it is a safe practice to duplicate the initial surface prior to editing it (select surface data object in the project view and press Ctrl-D).
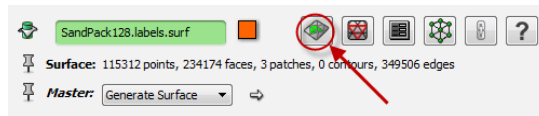
Figure 5.16: *Surface Editor* button.

- Open the *Surface Editor*.

When the *Surface Editor* is activated, a tool bar and a new menu *Surface* are available (see Figure 5.17).
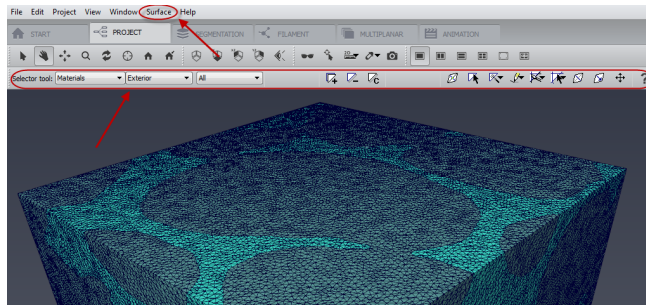
Figure 5.17: *Surface Editor* menu.

- Open the *Surface* menu and in the *Tests* submenu, select *Intersection test*. You can alternatively pick the *Intersection test* in *Selector tool* of the tool bar (see Figure 5.18).

The result of the test is displayed on top of the 3D viewer. In the current case, there is no intersection, so you can skip to the next test. However, be aware that intersections can be fixed automatically using the *Fix intersections...* tool in the *Edit* submenu (see Figure 5.19) or manually by using the *Surface Editor* tools in tool bar.

- In the *Tests* submenu, select *Aspect ratio*.
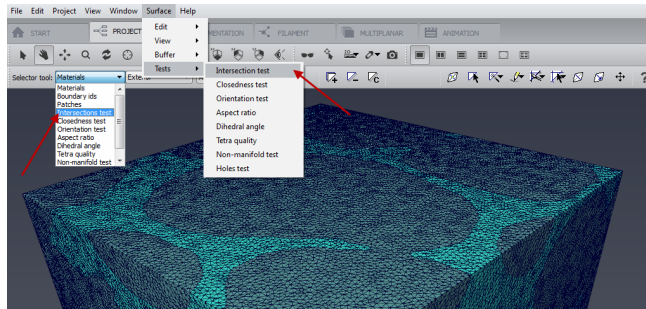
Figure 5.18: *Surface Editor* intersection testing submenu.
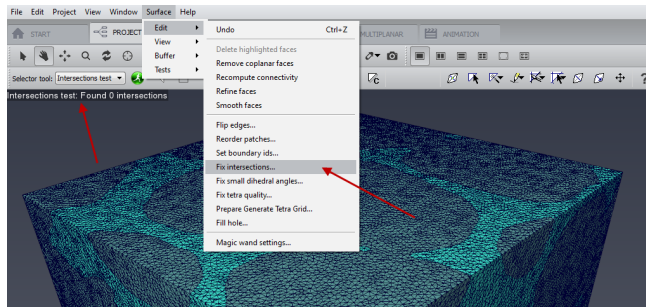
Figure 5.19: Intersection test result and intersections fixing submenu.

The triangle with the highest aspect ratio is displayed in the 3D viewer and the value of the aspect ratio is given in the top left corner (see Figure 5.20). It is possible to observe the next highest aspect ratio triangle by pressing on the right-pointing arrow in the *Surface editor* tool bar.

- In the *Edit* submenu, select *Prepare Generate Tetra Grid...*.
- Keep the default values for lower bound and attempted tetrahedron quality and press *Fix*.
- Close the dialog when finished.

Prepare Generate Tetra Grid combines the *Flip edges...*, *Fix small dihedral angles...*, and *Fix tetra quality...* tools.

If you run the *Aspect ratio* test again on the surface, you notice that the highest triangle aspect ratio has a much more acceptable value (below 30).

At this point, you can perform quality checks again thanks to the *Generate Tetra Grid* module and see also that all tests results are more acceptable.
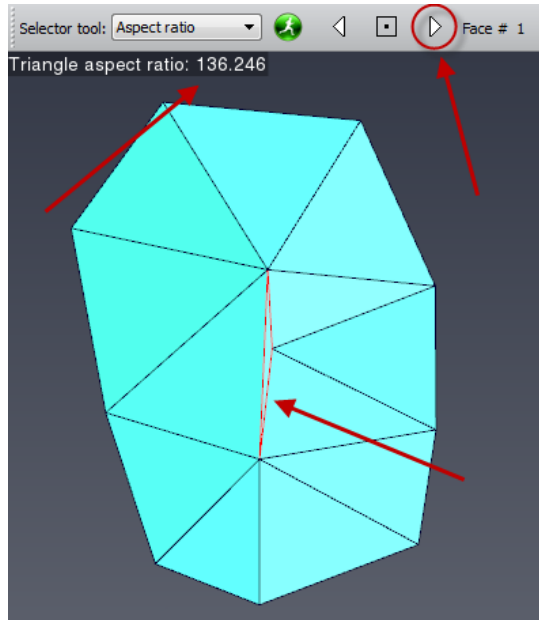
Figure 5.20: Testing the aspect ratio of triangles. A triangle with bad aspect ratio is highlighted.

If the surface you are editing happens to have intersections or too high triangle aspect ratio or any other issue revealed by some test on it, you may need to interact with the surface to correct it. To do so, you have to use the *Tests* menu and to select the test corresponding to your issue. The first problematic triangle, will be displayed, highlighted in red, among its neighbors.

For example, on the surface considered for this tutorial, the triangle with the highest aspect ratio can be improved:

- If not already done, run the *Aspect ratio* test.
- Switch to interaction mode (press [Esc]).
- Select the *Translate Vertices* button in the *Surface Editor* toolbar (or press T key) (see Figure 5.21).
- Click on the vertices of the triangle you want to move and use the dragger to move it (see Figure 5.22). You can pick the square part or rod part of the dragger to translate along a plane or an axis, which can be swapped by pressing once or twice the Ctrl key. Press Ctrl-Z to undo.

Another way to improve the quality of triangles is to collapse one edge of the triangle using the *Contract Edges* tool:

- Run the *Aspect ratio* test to get the next bad triangle.

Advanced Surface and Grid Generation **225**

Figure 5.21: *Translate Vertex* tool of *Surface Editor*.
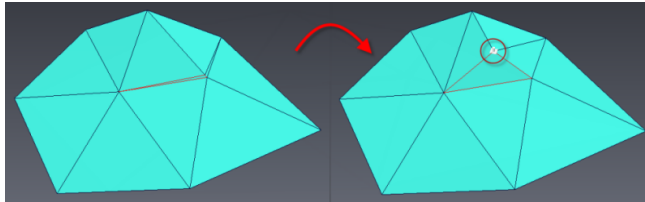
Figure 5.22: Editing a triangle with the *Translate Vertex* tool to improve its quality.

- Switch to interaction mode (press [Esc]).
- Select the *Contract Edges* button in the *Surface Editor* toolbar (or press shortcut O key) (see Figure 5.23).
- Click on the edge of the triangle you want to remove (see Figure 5.24).

Figure 5.23: *Contract Edges* tool of *Surface Editor*.
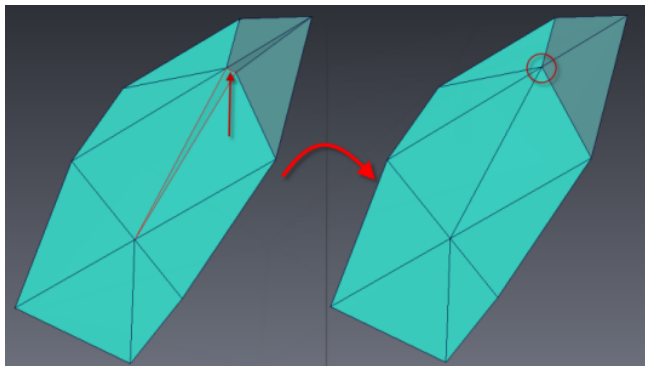
Figure 5.24: Editing a triangle with the *Contract Edges* tool to improve its quality.

At this step, you may want to re-run the *Prepare Generate Tetra Grid* tool.

You can ask now Avizo to remesh the surface using *Remesh Surface* module.

5.3.6 Remeshing and exporting surfaces

- Connect a *Remesh Surface* module to the surface.

In the *Desired Size* port of the module, it is possible to set the targeted number of triangles as a percentage of the original surface number of triangles.

- Keep the percentage value of 50% in the *Desired Size* port.

If the surface is quite simple, the rest of the default parameters can be used, but if the surface is not so simple, for example if it includes multiple patches, tuning the parameters in advanced mode will be useful.

Smoothness is an important parameter in the advanced options. This parameter allows for controlling how sharp you want to keep angles between triangles. This is important if you want to keep sharp edges, for instance, in your surface (see Figure 5.25).
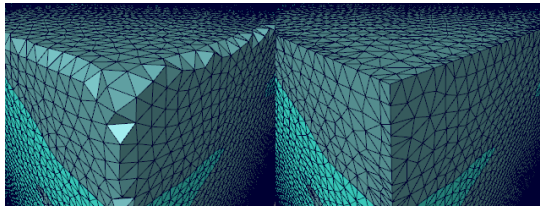
Figure 5.25: Surface remeshed with two different values for smoothness parameter: 0 (left) and 0.6 (right).

- Set the *smoothness* parameter of port *Error Thresholds* to 0.6.

As the current surface includes multiple patches with a lot of common interfaces, in order to minimize intersections near contours, between the generated triangles, the process should be split in two remeshing steps.

- Check the *fix contours* check box in the *Contour Options* port.
- Press *Apply* to run the remesher a first time.

At this step, there are no intersections, but the contours have shorter edge lengths than the rest of the mesh. This might be undesired. The next step will adjust the edge length of the contours.

- Connect a second *Remesh Surface* module to the newly generated surface.
- Set percentage in the *Desired size* port to 100%.
- Set the *smoothness* parameter of port *Error Thresholds* to 0.6.
- Remove the check of the *fix contours* option if it is checked and check the *only around contour* option in the *Zone Options* (Deploy *Zone* port).

- Run the remesher (see Figure 5.26).
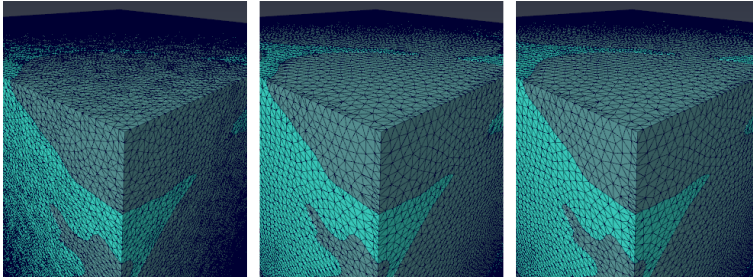
Figure 5.26: From left to right: initial surface, fix contours remeshing, remeshing around contours only.

The triangle based surface can then be exported to various formats including STL, using *Export Data As...* in *File* menu.

At this point, you may want to go through the *Surface Editor* process again to finalize cleaning of the surface, check for intersections and check for aspect ratio.

5.3.7 Generating tetrahedral grid

- Attach a *Generate Tetra Grid* module to the remeshed surface.
- Press on the *Run now* button.
- Press *Continue* in the computational time warning dialog.

The module is generating a report showing some information about the quality of the mesh (see Figure 5.27). The generated tetrahedral grid can be visualized with a *Tetra Grid View* display module.

Figure 5.27: Tetrahedral grid quality report.

By default, all tetrahedra are generated according to the size of the triangles they are going to be "attached" to onto the surface.

- Press on the *Meshsize* button in the *Generate Tetra Grid* module.

If you look at the Meshsize parameters for the two phases (pore space is *Material1* and grains is *Material2*), the value is set to the default value 0, which will trigger automatic sizing of tetrahedra

according to triangles size (see Figure 5.28). You can change this mesh size parameter per material, so the grid generator will try to generate tetrahedra with this edge size while going away from the surface interface (see Figure 5.29). The edge size on the surface interface will always be the size of the edges of the shared triangles.
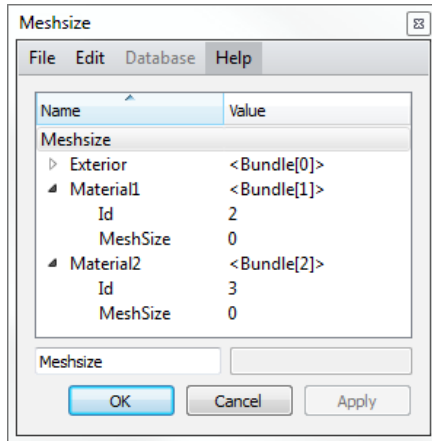
Figure 5.28: Meshsize parameter dialog.
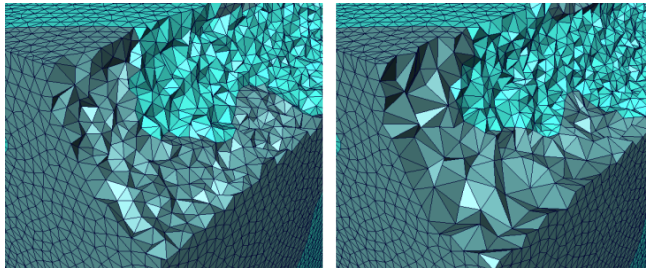
Figure 5.29: Left: Mesh size set to default (0), right: mesh size set to 2 for pore space (*Material1*) and 5 for grains (*Material1*).

Note: If you want to change the size of the tetrahedra on the surface, you can use the *Surface Editor* **before the grid generation** in order to refine or simplify a particular surface patch. You can, for instance, select a material in the *Surface Editor* (that will be highlighted in red) and then refine this patch using the *Refine faces* command in the *Edit* submenu of the *Surface Editor* menu (see Figure 5.30).
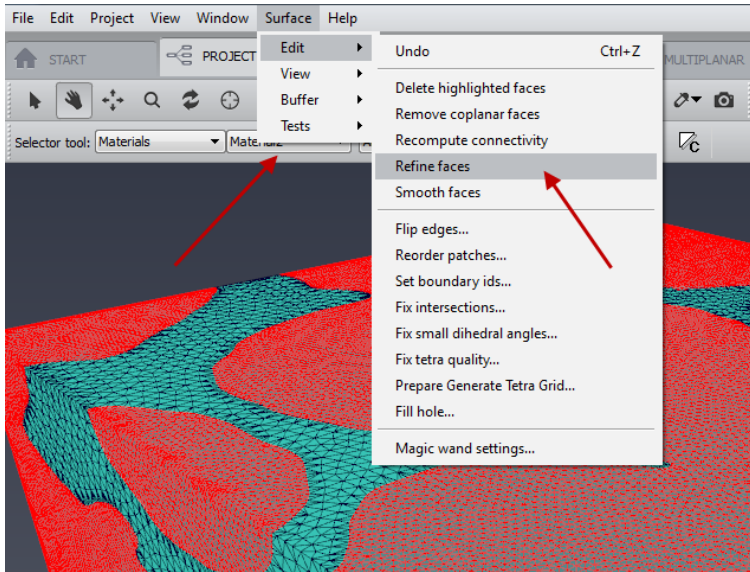
Figure 5.30: Refining the triangles of the grains (*Material1*) phase surface.

5.3.8 Assigning boundary conditions and exporting data

You can use the *Surface Editor* to assign boundary conditions to different patches of the surface, before you actually export the 3D grid. You may want to define inlet and outlet for a flow simulation as well as walls.

- Select the remeshed surface you used to generate the tetrahedral grid.
- Open the *Surface Editor*.
- Switch to interaction mode (press [Esc]).
- Press on the *Magic Wand* tool.

The *Magic Wand* tool can be used in order to select a patch of coplanar triangles. You have to define the crease angle, which is the criteria used to define coplanarity of triangles and will be used by the *Magic Wand* to select neighboring triangles.

- Set the *Degrees* parameter to a value small enough, such as 10 (see Figure 5.31).
- Select a patch by picking any triangle from this particular patch.

You can then assign a particular boundary condition to this patch.

- Open the *Surface* menu, select the *Set boundary ids...* in the *Edit* submenu.
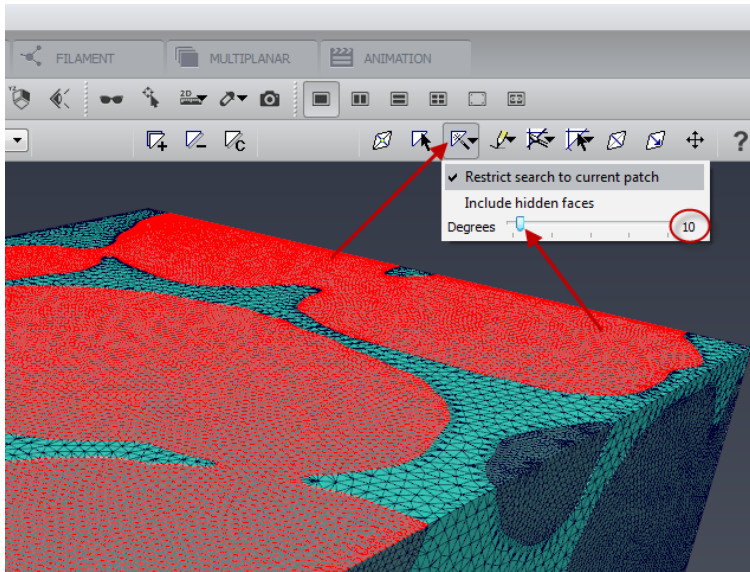
Figure 5.31: Using the *Magic Wand* to select a patch of coplanar triangles.

- In the right column, select the type of boundary condition you want to assign to the patch and then press on *Set*.
- Repeat the operation as many times as needed to have no face left undefined (see Figure 5.32).
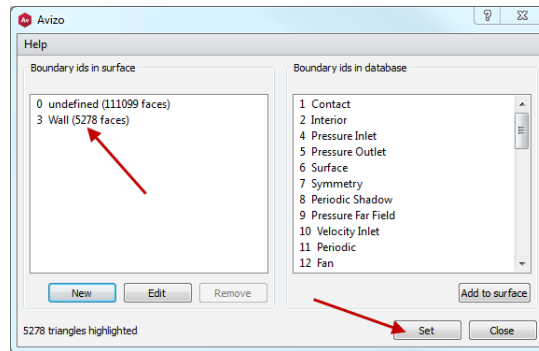
Figure 5.32: Setting the boundary ids.

Using the *Selector* tool from the *Surface Editor*, you can also select different patches on the whole surface and assign, for instance, a Wall boundary condition to all of the grain walls.

The *Surface View* module can be customized to display the surface colored by material but also by type of boundary conditions.

- Quit the *Surface Editor*.
- In the *Selection Mode* port of the *Surface View* module, select *BoundaryID*.
- In the *Colors* port, select *boundary ids* (see Figure 5.33).
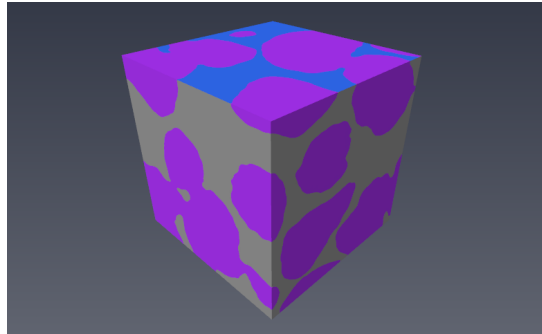
Figure 5.33: Surface colored by boundary ids: for instance here, top pore space faces are velocity inlet, bottom ones are pressure outlet (not visible), side ones are periodic boundaries and grain faces are walls.

Once boundary conditions have been assigned to the surface, you can re-assign them to the 3D grid that was generated previously, so that if saved, these boundary conditions will be exported to the simulation solver.

- Connect an *Assign Boundary Conditions* module to the grid (see Figure 5.34).
- In the *Surface* port, select the remeshed surface on which you just assigned boundary ids.
- Press *Apply*.

Figure 5.34: *Assign Boundary Conditions* module.

A new grid is created, with boundary conditions assigned. It is suitable for simulation and can be exported to FEA/CFD software using the *File / Export Data As...* menu (see Figure 5.35) .

FEA/CFD software format export is available with Avizo XWind Extension.

You can then use Avizo XWind Extension to post-process results of simulation back from the solver
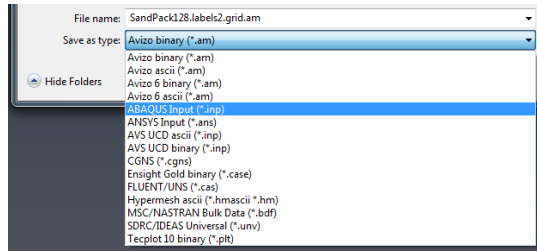
Figure 5.35: Export formats available in the *Export Data As...* menu.

(see chapter 11 - Avizo XWind Extension User's Guide) .

Note: For fluid dynamics in porous media, you can use Avizo XLabSuite Extension modules to assess the absolute permeability of your sample. Avizo XLabSuite Extension directly relies on mask identifying porous phase and solid phase and does not require surface or 3D Mesh generation.

5.4 Visualization and Analysis of 3D Models and Numerical Data with Avizo

The Avizo suite provides advanced support for:

- 3D mesh generation for numerical simulation applications such as flow, stress, or thermal analysis,
- Export of surfaces or 3D meshes to numerical solvers,
- Post-processing of result data coming from solvers,
- Powerful visualization and analysis of scalar, vector, and tensor fields, coming from either simulation or measurements.

Avizo enables workflows such as:

- Exploration, analysis and comparison of data coming from simulation or measurements,
- Modeling from 3D images for Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD), Computer Aided Design (CAD), Rapid Prototyping,
- High quality presentation and communication, from movie generation to remote collaboration, immersive displays or virtual reality (requires *Avizo XScreen Extension*).

Avizo XWind Extension is the software suite that includes the Avizo Lite Edition feature set and all of its extensions required for visualizing, post-processing, analyzing, and presenting results from CAE and CFD simulations.

For extended support for numerical data - import/export, analysis, and visualization - please refer to chapter 11 Avizo XWind Extension User's Guide.

Avizo provides a first level of support summarized in the following sections:

- *Features for surfaces*
- *Supported grids*
- *Features for 3D grids*
- *Scalar fields visualization*
- *Vector fields visualization*
- *Time dependent data visualization*
- *Basic compute modules and tensors*

For convenience, in Avizo Lite Edition, the Avizo XMesh Extension gathers modules providing basic support for visualization of FEA and CFD data.

For a complete overview of features that can complement analysis and presentation of numerical data, please refer to Features overview section and Chapter 11 Avizo XWind Extension User's Guide.

The recommended tutorials for post-processing using Avizo XWind Extension can be found in Chapter 11 Avizo XWind Extension User's Guide.

The on-line version of this chapter contains tutorials for the following topics using Avizo:

- *Air flow around an airfoil* - streamlines and other techniques,
- *Study of a helicopter combustion chamber* - scalar and vector visualization, probing, volume rendering,
- *Flow around a square cylinder* - time dependent data post-processing.

5.4.1   Avizo features for surfaces

Avizo supports visualization of triangular surfaces with attached numerical data (*Surface View*). Surfaces can be directly imported (see *File Formats* in the Reference Guide), or generated from 3D image labels (*Generate Surface*) or point clouds (*Delaunay Triangulation* and *Point Wrap Triangulation*). It is possible to simplify, smooth and edit surfaces, compute surface curvature and surface distances, align and warp surfaces, export surfaces and much more.

Avizo XMesh Extension provides additional features such as surface registration (*Align Surfaces*), quality control (*Triangle Quality*), processing and visualization of vector data over surfaces (*Displace Vertex*, *Magnitude*, *Vectors Slice*, *Stream LIC Surface*, *Line Streaks*), data interpolation (*Interpolate*) and 3D grid generation (*Generate Tetra Grid*).

5.4.2 Supported grids in Avizo XMesh Extension

Avizo supports a wide range of 3D meshes. Display and compute modules may be dedicated to a specific kind of grid. This is specified in the module help.

In Avizo you can work on:

- Regular grids,
- Unstructured tetrahedral grids,
- Unstructured hexahedral grids.

Avizo XWind Extension adds support for

- Unstructured models grids (arbitrary sets of tetrahedron, hexahedron, wedge and pyramid elements).

Note: If an Avizo XWind Extension license is detected at run-time, whenever loading files with format AVS, Fluent, Ideas, Tecplot, or any format supported in Avizo XWind Extension (see *File Formats index* for a complete list), an unstructured model grid is created automatically that can be connected to Avizo XWind Extension modules. This is true even if the file contains only tetrahedra, or only hexahedra or if the mesh is a regular grid.

Regular grids include four types of grids:

- Uniform grids, the simplest form of regular grids because all grid cells are axis-aligned and of equal size;
- Stacked grids, that are composed of a stack of uniform 2D slices with variable slice distance;
- Rectilinear grids, where the grid cells are aligned to the axes, with distance between cells that may vary in each direction;
- Curvilinear grids, each node of which has its position stored as a 3D vector and is numbered one after another.

For more details, please refer to section 13.5.2.2 for regular coordinate types.

Section 13.5.3 for tetrahedral grids and section 13.5.4 for hexahedral grids contain information on these grid properties and on their coordinates or data storage systems.

5.4.3 Avizo XMesh Extension features for 3D grids

5.4.3.1 Grid visualization

Several display modules exist for the visualization of a grid and of its components, such as the mesh, the faces, the boundaries or the nodes. It is also possible to visualize only a part of the grid, depending on the region of interest chosen or on one (or several) material(s) chosen among the different materials the set under study is made of. To that end, the modules *Tetra Grid View* for tetrahedral grids and *Hexa*

Grid View for hexahedral grids provide the user with various powerful display features.

Boundary faces of a tetrahedral grid can be displayed with the previously mentioned modules or with *Grid Boundary* and the boundary conditions can be displayed with *Boundary Conditions*.

Cross sections are available for tetrahedral grids using *Grid Cut.*

5.4.3.2 Grid conversion, transformation and generation

The compute modules *Lattice To Hexa Grid*, *Tetra Grid To Hexa Grid* and *Hexa Grid To Tetra Grid* convert a grid of one type to another type. Conversion of data might be performed at the same time.

Several tools for grid generation and transformation exist for tetrahedral grids, such as *Generate Tetra Grid* (generation of a tetrahedral grid from a 3D triangulated surface), *Merge Tetra* (combination of grids) and *Align Surfaces* (to align triangulated surfaces). You will find an example of grid generation in the tutorial section 5.2 Creating a Tetrahedral Grid from a Triangular Surface.

5.4.3.3 Grid quality

The quality of a mesh is critical for the accuracy of the computation that are done on the grid. Therefore, quality measurement tools such as *Tetra Quality* for tetrahedral grids and *Hexa Quality* for hexahedral grids are very useful. Triangulated surfaces can be tested with *Triangle Quality*.
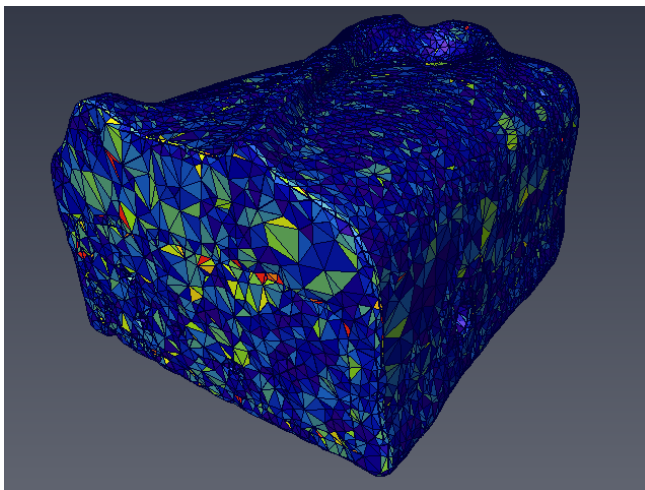
Figure 5.36: Chocolate bar grid representation colored with respect to the diameter ratio of the tetrahedral cells.

5.4.4 Scalar fields visualization

The volumetric visualization tools discussed in section 5.4.3 can also be used to visualize scalar fields on the whole volume since most of these modules have a pseudo-coloring functionality.
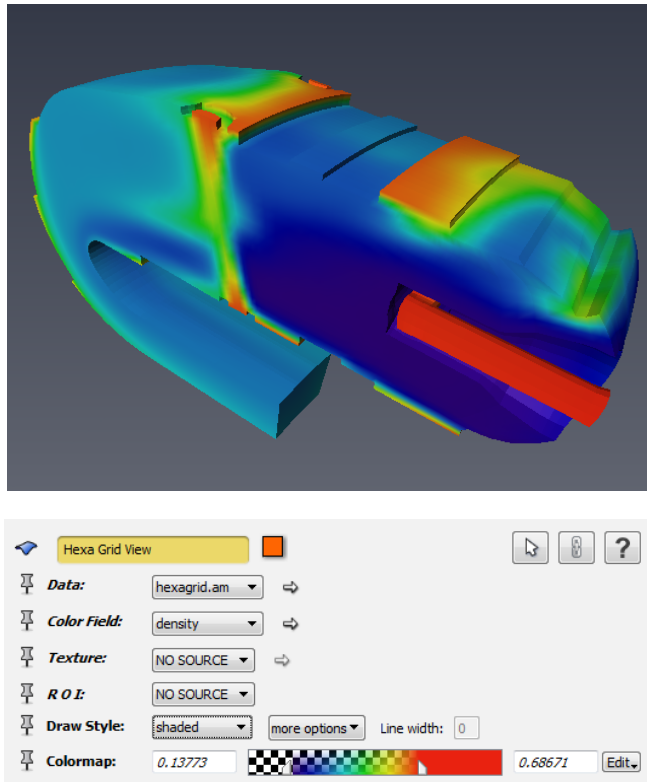
Figure 5.37: Density used as colorfield for the pseudo-coloring of the hexahedral grid of a helicopter combustion chamber section.

Field Cut for tetrahedral grids has the same pseudo-coloring capability for visualizing cross sections of a 3D scalar field.

Isosurfaces (e.g., *Isosurface* for tetrahedra) are powerful display tools to aid in the understanding of physical phenomena and they are available for every kind of grid supported by Avizo XMesh Extension.

5.4.5 Vector fields visualization

The most intuitive way to display a vector field is to plot the vectors as arrows. With the *Vectors Slice* module, you can plot vector fields defined on any grid type supported by Avizo XMesh Extension on a plane cut of the domain.

A volumetric representation is available for vector fields defined on tetrahedral grids with the *Tetra Vectors* module.

The *Particle Plot* module represents vector fields (for all kind of grids supported by Avizo XMesh Extension) by particles (cones) pointing in the direction of the local flow. The particle plot can also be animated to show the flow motion.

Drawing the pathlines of particles is a good way to represent and understand the behavior of a flow, for example, for turbulent flows with many recirculations. To do so, you can use the *Stream Ribbons* module or the *Illuminated Streamlines* module. The choice of the seed points can be done in different ways, for example, using the *Seed Surface* module.

Finally, the *Stream LIC Surface* module visualizes a vector field defined on an arbitrary 3D triangular surface using a line integral convolution (LIC) algorithm which generates a surface texture that reveals the directional structure of the vector field. A similar 2D algorithm is implemented by the *Stream LIC Slice* module.

Please refer to tutorial *Air flow around an airfoil*, especially for *Line Integral Convolution* and *Illuminated Streamlines* techniques applications. This tutorial is available in the online documentation.
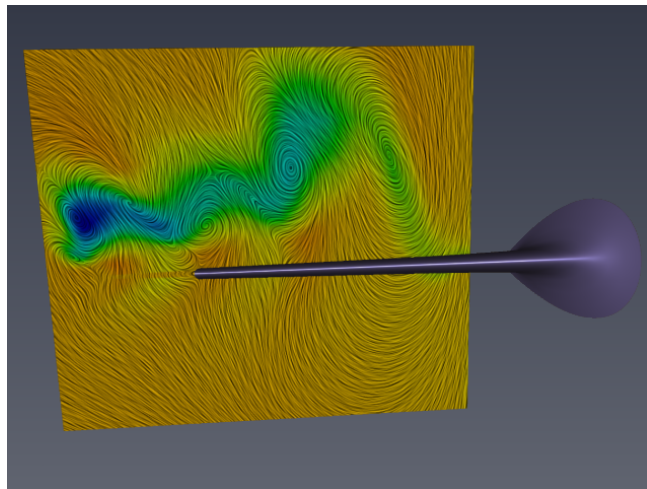
Figure 5.38: Stream LIC Slice display of the air velocity vector field behind an airfoil.

5.4.6 Time dependent data visualization

The visualization features previously mentioned can be efficiently combined with the *Time Series Control* module from Avizo. After loading data via the *Open Time Series Data...* option in *File* menu, one can connect display or compute modules to the data and use *Time Series Control* to obtain and play an animation sequence of, e.g., a flow behavior in time.

This technique is especially appropriate for the *Particle Pathlines* display module. Combined with the *Time Series Control*, it represents a vector field by a set of particles that move in time according to the vector direction and velocity.

Please refer to the *Flow around a square cylinder* tutorial for suggestions on other Avizo XMesh Extension modules to use for time dependent data visualization. This tutorial is available in the online documentation.

5.4.7 Compute modules

Avizo XMesh Extension contains some computational tools, that can be useful in the field of flow and stress analysis for example.

5.4.7.1 Basic computational tools

Coloring a 3D vector field representation or a flow cross section such as a *Stream LIC Slice* with respect to the magnitude of the vectors is a very common representation in physics, very often used for the velocity vector field. To compute the magnitude, the *Magnitude* module can be used on any type of grid supported by Avizo XMesh Extension.

Divergence is an operator used in many physics equations. The *Divergence* module computes the divergence of vector fields defined on uniform grids. The same is true for the gradient of scalar fields or Jacobian matrix of vector fields and can be calculated with the *Gradient* module, on regular grids.

The *Lambda 2* module returns a scalar field that is of interest for the CFD study of turbulent flows as the points where λ_2 is negative mark a vortex core.

5.4.7.2 Tensor data

Tensor generation is possible on a regular grid:

- *Gradient*, as presented before, computes the Jacobian matrix of a vector field;
- *Rate Of Strain Tensor* computes the rate of strain tensor for a displacement vector field.

Avizo XMesh Extension provides some computational and visualization support for symmetric second order tensors.

The *Extract Eigenvalues* module computes the eigenvalues of a symmetric second order tensor. *Eigenvector To Color* creates a color representation of the eigenvectors.

Finally, *Tensor View* displays symmetric second order tensors using tensor glyphs.

5.5 Skeletonization

Avizo XFiber Extension gathers a set of powerful tools for analysis of network or tree-like structures in 3D images such as porous, dentritic, vascular, fractured or fibrous networks.

The 3D image must first be segmented into a label or binary image. Avizo offers for this a rich set of tools, including the *Segmentation Editor*, as well as advanced image processing tools available in Avizo. With skeletonization tools, the labeled regions can then be *thinned* to a 1-voxel thickness skeleton, equidistant to shape boundaries according to a *Distance Map* that can be computed by Avizo. The image skeleton can be converted into a geometrical *Spatial Graph* for further analysis.

In this chapter you will learn how to:

- use the *Auto Skeleton* module,
- display and export results of skeletonization,
- achieve skeletonization step-by-step for detailed control,
- apply skeletonization to large data.

You should be familiar with the basic concepts of Avizo to follow this tutorial. In particular, you should be able to load files, to interact with the 3D viewer, and to connect display modules to data modules. All these issues are discussed in the chapter 2 - Getting Started. For actual use of skeletonization with your data, you may also need to be familiar with image filtering and segmentation (see chapter 3 - Images and Volumes Visualization and Processing in Avizo). Further related tools can also be found in Avizo.

5.5.1 Getting started with Skeletonization: the Auto Skeleton module

The *Auto Skeleton* module extracts from image data the centerline of interconnected regions, such as filamentous structures. The module works on segmented images (label images) as well as on grayscale images, in which case the image is segmented on the fly with a user-defined threshold value. The module basically wraps up a couple of single compute modules that have to be executed in sequence. It first calculates a distance map of the segmented image (*Distance Map For Skeleton*), and then performs a thinning of the label image such that finally a string of connected voxels remains (*Thinner*).

Let's extract the skeleton of porosity network in a fractured rock core sample data set:

- If the *Project View* is not empty, press [CRTL+N] to start a new empty project.
- Choose *File / Open Data...* from the *File* menu and load the file `data/core/coreSample.am` from the Avizo root directory
- Remove the *Ortho Slice* module.

- Attach an *Auto Skeleton* module to the data set by selecting *Image Processing / Skeletonization / Auto Skeleton* in the data context menu.
- Adjust the **Threshold** port of *Auto Skeleton* module to 27. Leave other ports to their default settings.
- Press the Apply button to start the processing.
- You may also attach some *Display* module to coreSample.am such as *Image Ortho Projections*.

Figure 5.39 shows the result of these steps.
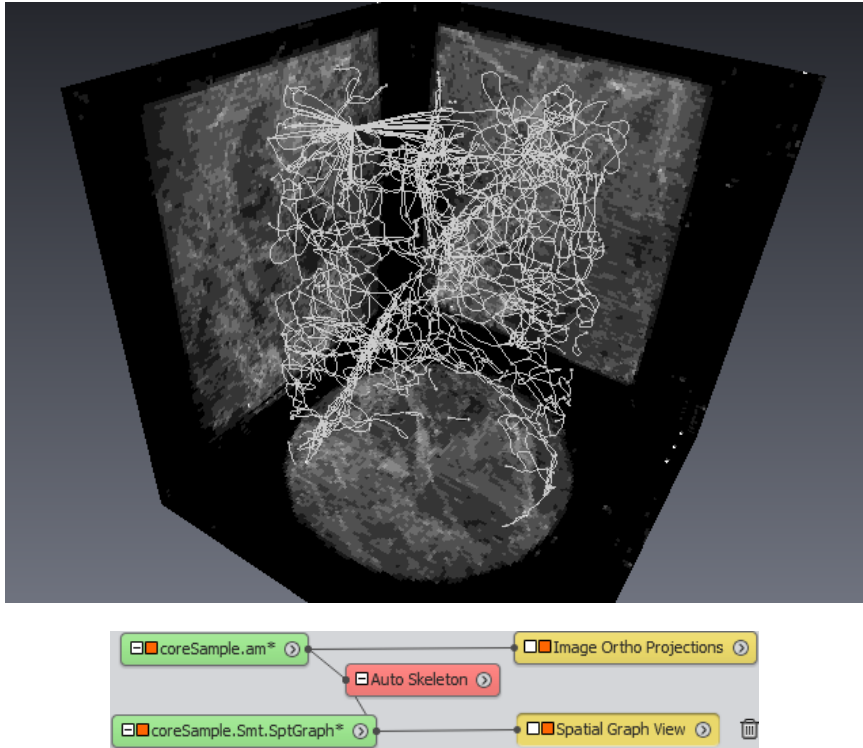
Figure 5.39: Visualization of the *Auto Skeleton* result and associated *Project View*.

The *Auto Skeleton* module automatically converts the voxel skeleton to a *Spatial Graph* object. A *Spatial Graph* consists of nodes and segments where nodes are the branching points and endpoints, and segments are the curved lines connecting the nodes. The three dimensional course of a segment is given by a sequence of points in 3D space. A set of nodes connected by segments is called a graph. A *Spatial Graph* data object can store several graphs. Furthermore, *Spatial Graph* objects can hold scalar values such as the thickness.

The *Spatial Graph* is created by *Auto Skeleton* with a *Trace Lines* module. Lines may appear jagged because they connect centers of voxels. By default, *Auto Skeleton* smoothes the graph lines with a *Smooth Line Set* module. Smoothing can be adjusted or disabled. With the default **Output Options** port setting `Create SpatialView`, a *Spatial Graph View* module is attached to display it.

The distance to the nearest boundary (boundary distance map) is stored at every point in the *Spatial Graph* object as the `thickness` attribute (using *Eval On Lines*).

Note: The `thickness` attribute is computed by the *Auto Skeleton* module as a discrete chamfer distance multiplied (by default) by `1/3` of voxel size, with a minimum of half voxel size. This may be used as an estimate of the local thickness. Optionally, a distance map data can define a *data parameter* `ChamferMapScaleFactor` used to adjust thickness attribute (see details in *Eval On Lines*).

Tip: It may be useful to resample the data to an isotropic voxel size, before using *Auto Skeleton* module (*Resample*). This optional step may improve the result of the distance map and skeletonization process, and may lead to a smoother spatial graph. Depending on the initial voxel aspect ratio, this may, however, increase significantly the data size, unless subsampling is used in some direction.

Note: You may notice some star-shaped sets of connected segments in the spatial graph. This may happen, for example, when the segmentation resulted in a background boundary surrounding a very small area - like a solid region hanging fully surrounded by void. This may be the sign of too much noise in the data which would require cleanup processing, such as a Gaussian filter, median filter, or more advanced filters of Avizo. Such solid islands in segmentation may also be eliminated by using the segmentation editor or morphological tools.

Tip: In the case of a strict tree topology, it may be advantageous to restrict the search to a tree. You may want to use the module *Centerline Tree* to extract a graph with guaranteed tree topology. Note that *Centerline Tree* cannot apply directly to a gray image but only to segmented image data: it requires a label image as input.

5.5.2 Displaying and exporting skeletonization results

5.5.2.1 Visualizing skeleton thickness

It is possible to modify the graph visualization by changing options in the *Spatial Graph View* module (Figure 5.40). For example, it is easy to show the segments as tubes whose diameter depends on the *thickness* defined by the distance map. This is achieved by selecting the checkbox `Tubes` on the **Segment style** port and changing **Tube scale** to `Thickness`. However, be warned that this can be slow depending on the complexity of the spatial graph and the graphics hardware used. In this tutorial, the segments will be displayed with a color depending on their thickness:

- Select *Spatial Graph View* module.
- Set `Nodes` port to off.
- Change **Segment coloring** port from `constant` to `thickness`.
- In the port **Segment colormap**, use the `Edit` button to select `physics.icol`. Then ad-
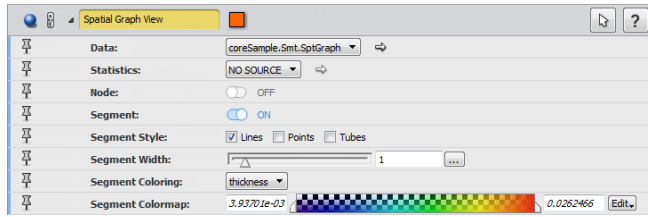
Figure 5.40: Properties of the *Spatial Graph View* module.

just the data range using the `Edit > Adjust range` button, or by right-clicking on the colormap shade and selecting `Adjust range`.If necessary, use the `Zoom > Range` button. See Figure 5.40 to see the complete properties of the module.

Figure 5.41 shows the result of these steps.
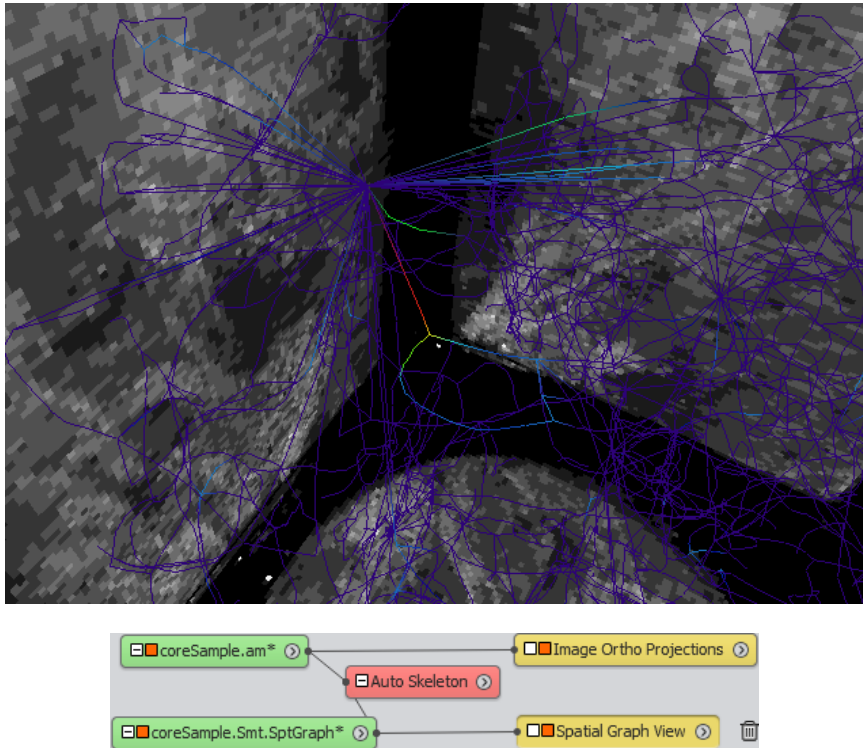
Figure 5.41: Visualization of the *Spatial Graph View* result and associated project view.

5.5.2.2 Exporting the Spatial Graph

You can export *Spatial Graph* data, including connectivity and thickness information, to various formats such as *Avizo Spatial Graph ASCII* format, SWC and MV3D formats.

- For this, select the data module and select *Export Data As...* in *File* menu or data object context menu.

Here is an example output file using Avizo ASCII format:

```
# Avizo 3D ASCII 2.0

define VERTEX 787
define EDGE 1518
define POINT 8567

Parameters {
    ContentType "HxSpatialGraph"
}

VERTEX { float[3] VertexCoordinates } @1
EDGE { int[2] EdgeConnectivity } @2
EDGE { int NumEdgePoints } @3
POINT { float[3] EdgePointCoordinates } @4
POINT { float thickness } @5

# Data section follows
@1
0.637795 0.188976 0.0472443
0.661417 0.212599 0.0472443
<...>
@2
0 0
0 1
<...>
@3
7
4
<...>
@4
0.637795 0.188976 0.0472443
0.635375 0.183291 0.0472443
<...>
```

```
@5
0.00393701
0.00393701
<...>
```

5.5.2.3   Spatial graph statistics

You can get some statistics directly from the *Spatial Graph* data:

- Select the `coreSample.Smt.SptGraph` data module.
- Attach a *Spatial Graph Statistics* module to the data set by selecting *Measure And Analyze / Spatial Graph Statistics* in the data context menu.
- Press the `Apply` button.
- A new data appears in the project view: `coreSample.Smt.statistics`.
- Select `Create a new tab per graph` of the **Options** port.
- Press the `Apply` button again.
- Select `coreSample.Smt.statistics` and click on the `Show` button in the data properties to display a tabbed spreadsheet (see Figure 5.5.2.3). graph summary is displayed into the first tab. Then, graph statistics are summed up into the second tab. One graph corresponds to one tab.

Spatial Graph statistics can be saved in several file formats (CSV, XML, TXT).

5.5.2.4   Using Line Set objects

It can be useful to convert the *Spatial Graph* object to a *Line Set*. Line sets provide additional tools for display, editing, processing or export. For instance, the *Line Set Editor* can be used to interactively identify line or node indexes.

Note: in the following, the *Line Set Editor* is going to be used. It requires Avizo console to be visible, since it outputs information in it. By default, the console is not visible but it can be easily activated by clicking on the *Console* button of the main toolbar.

- Attach a *Convert / Spatial Graph To Line Set* module to the *Spatial Graph* data `coreSample.Smt.SptGraph`.
- Press the `Apply` button to create a *Line Set*.
- Select the created *Line Set*. In the properties panel, click on `Line Set Editor` button (see Figure 5.43).
- Hide the *Spatial Graph View* attached to `coreSample.am`. This step is necessary due to the fact that the lines drawn by this module are overlapping the line drawn by the *Line Set Editor*. It could cause problems for picking lines and points during the next step.

Figure 5.42: Spreadsheet containing the results of *Spatial Graph Statistics*.
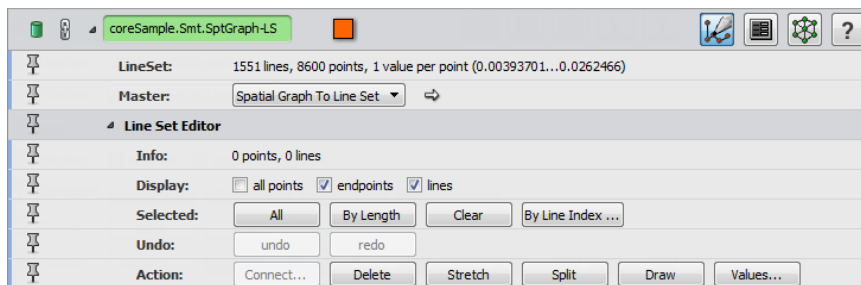
Figure 5.43: Properties of a *Line Set*. The button allowing starting the *Line Set Editor* is highlighted.

- Set the 3D viewer window to interaction mode (press the [ESC] key or click on the arrow button). You can now select segments and points of the line set. Corresponding information is displayed in the console.
- You may also want to see coordinates and image intensity value when picking a point. For this, attach a *Measure And Analyze / Point Probe* module to the image data coreSample.am. Make
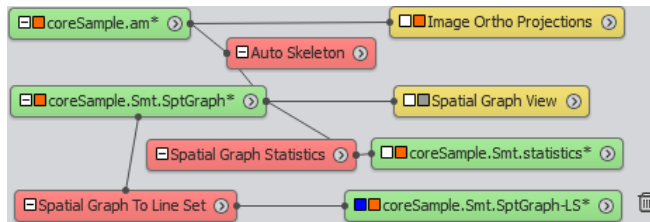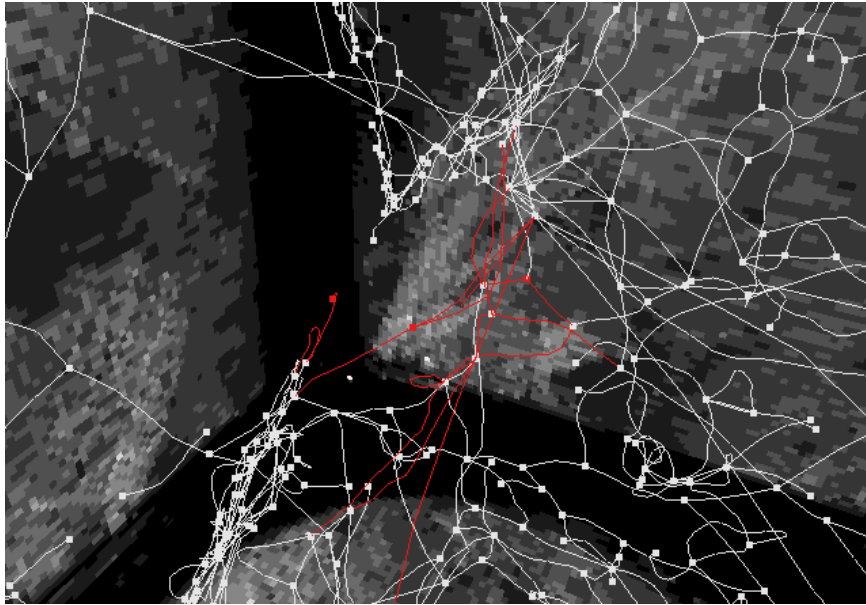
Figure 5.44: Visualization of the *Line Set Editor* result after a few selections and the associated *Project View*. Values appearing are not shown since it depends on the selected points and lines.

sure that the *Point Probe* module remains selected in the *Project View*. Then, when clicking with the middle mouse button on the line set's displayed points and segments, the `Point` coordinates and `Current value` of input field are displayed. You could also attach the *Point Probe* to the distance map data to display the estimated thickness. See the section below on how to display the distance map data object.

During visualization of large data sets, there is often the need to restrict the displayed geometry to a subvolume of the total data set. The *ROI Box* can help for that. You can attach it to any spatial data object. A number of display modules have an input connection called *ROI* that can be attached to the *ROI Box* module to restrict the view.

- Click on the `Line Set Editor` button to deactivate the editor.

- Create a *ROI Box* module by right-clicking on the line set data object and selecting *ROI Box* from the *Display* submenu of the context menu.
- Attach a *Display / Line Set View* module to the line set data object.
- In the *Optional Connections* port of the *Line Set View*, connect the port named *ROI* to the *ROI Box* module. To do this, right-click on the white square on the left side of the *Line Set View* icon and select *ROI*. A blue line is attached to the mouse pointer; after you click on the *ROI Box* icon, the two modules are connected.
- Switch the viewer to interaction mode and click and drag one of the green squares. This will adjust the region of interest and the *Line Set View* will adopt the new restriction immediately.
- Click and drag on the (invisible) faces of the cuboid to move it to another position.

Figure 5.45 shows the results of these few steps.

When working with a small subset of the line set, it is possible to do visualizations that require more graphics power. For example, the lines can be displayed as tubes that reflect the local thickness.

- Choose a rather small part of the line set.
- Select the *Line Set View*.
- Click on the **Shape** drop-down menu and select `Circle` in `Line Shape` group.
- Click on the **Scale Mode** drop-down menu and select `Data 0`.
- Move the **Scale Factor** slider to `2`.

Figure 5.46 shows an example and the parameters of the *Line Set View* module.

In the viewer, the lines are now displayed as tubes. The thickness is scaled with the data associated with the lines.

Note: The data value associated with the lines is the local radius. The *Line Set View* scales by the local diameter. To scale to the physical size, you therefore must use a **Scale Factor** `2`.

It can be useful to color the lines in different ways. In the next example, the line set will be colored by the local z value. First create an *Analytic Scalar Field* to provide the depth (z value), then set line set coloring according to values evaluated from this field.

To create the scalar field:

- Right-click in the *Project View*, then on *Create Object...* and select *Images And Fields / Analytic Scalar Field*.
- Select the newly created icon.
- Type `z` into the **Expr** field. The Range info is updated to `-0.9...0.9`.
- Select the *Line Set View* object,
- Attach its **Scalar field 1** input port in the `Optional Connections` group to the *Analytic Scalar Field* data object.
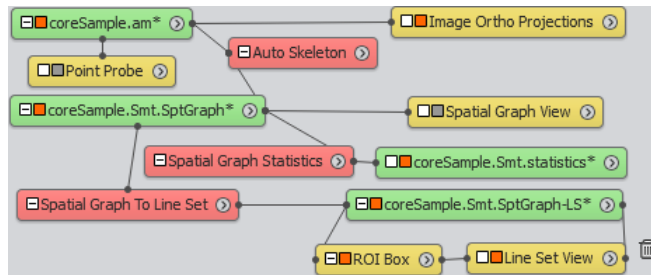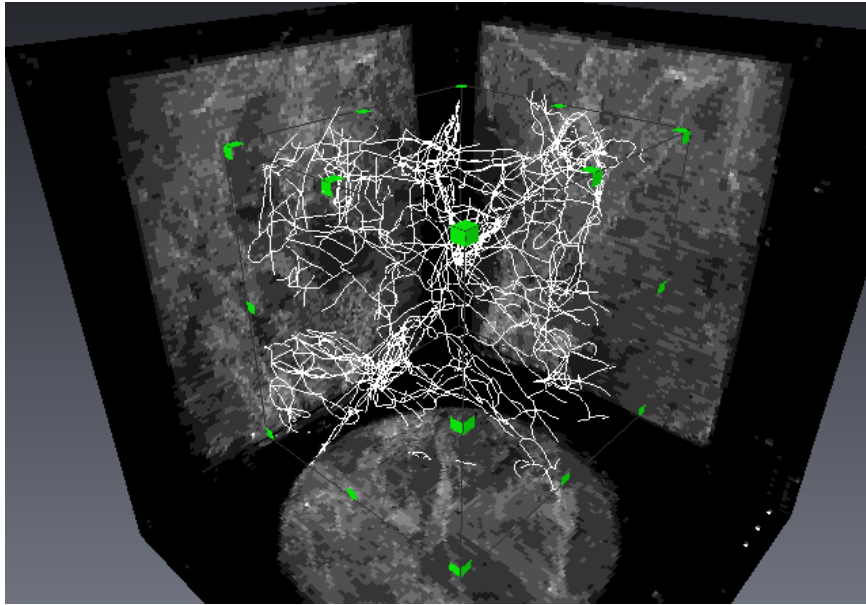
Figure 5.45: Visualization of the *Line Set View* reduced to the *ROI Box* and associated project view.

- Set the **Color Mode** port in the `Coloring` group of *Line Set View* module to `Analytic-Scalar-Field`
- Use the **Colormap** port to change colormap and data range. See Figure 5.47 for more details about the parameters to modify.

Figure 5.48 shows the result and the *Project View*.
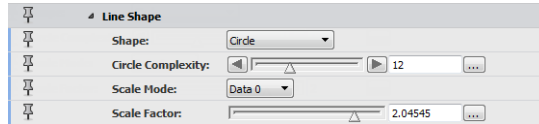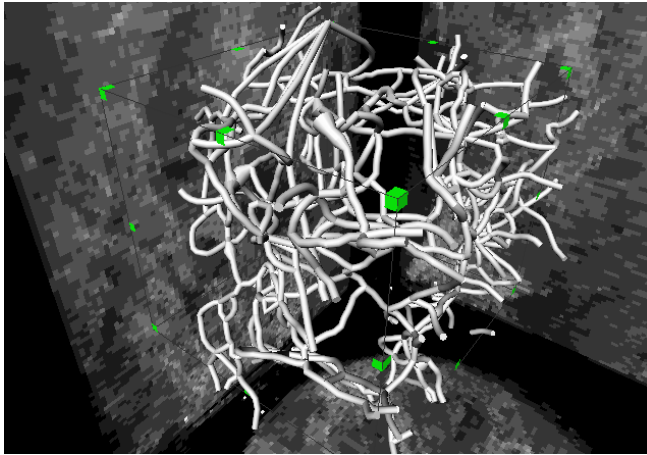
Figure 5.46: Visualization of the *Line Set View* reduced to the *ROI Box* and the parameters of the *Line Set View* module.
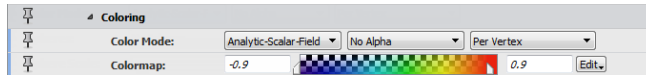
Figure 5.47: Parameters modified in the *Line Set View* module for coloring the tubes.
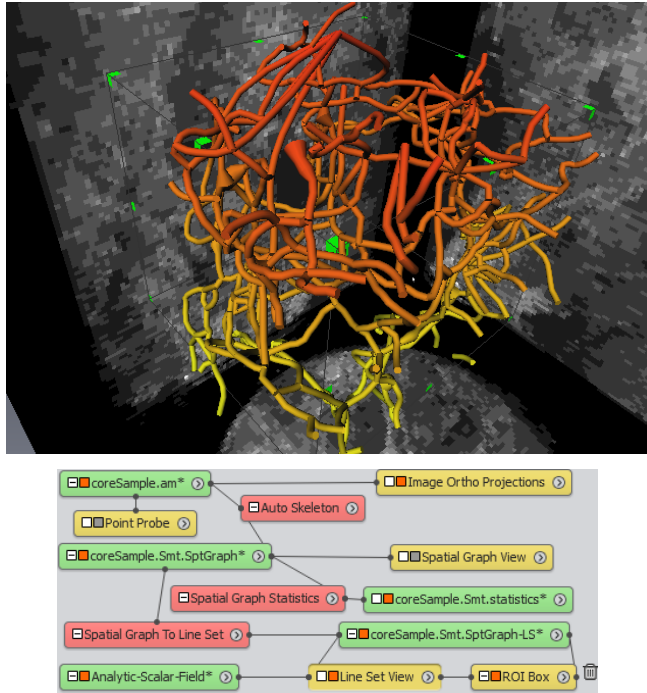
Figure 5.48: Visualization of the colored *Line Set View* and the associated project view.

The *Line Set* object also provides a Tcl script interface to inquire or modify the data. See chapter 9.4 for an introduction to Tcl scripting.

It is possible to convert a *Line Set* back to a *Spatial Graph*.

5.5.2.5 Intermediate data

The *Auto Skeleton* module can expose several intermediate data objects in the module workspace. These intermediate files can be useful for checking, export, or as starting point for specific processing.

- Delete the *Line Set View* module, the `coreSample.Smt.SptGraph-LS` spatial graph (which should also remove the *ROI Box*), the Analytic Scalar Field data created in previous steps, the *Spatial Graph Statistics* module, the *Point Probe* module, the `coreSample.Smt.statistics` data and the *Spatial Graph To Line Set* module.
- Select *Auto Skeleton* module.
- In the Properties panel, **Create Objects** port, check *Distance Map*
- In the Properties panel, **Options** port, uncheck `Show Spatial Graph`

- Press the `Apply` button to process skeletonization again.
- Attach an *Ortho Slice* to the distance map field `coreSample.DistField`. Result may look like Figure 5.49.
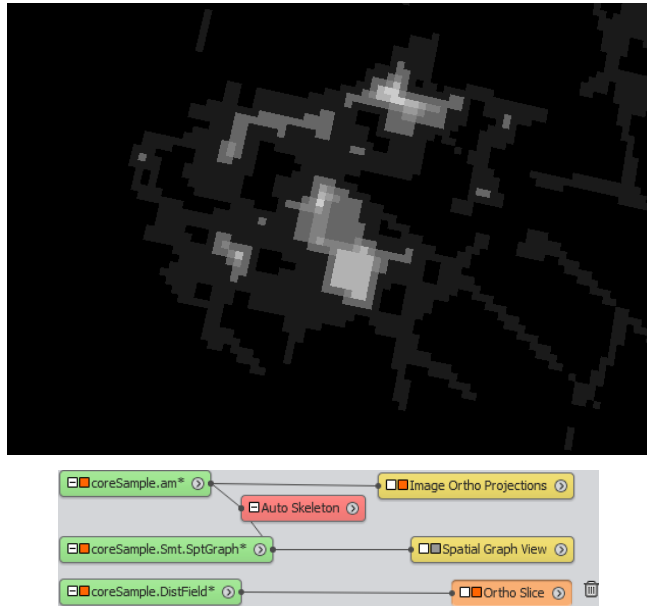
Figure 5.49: Visualization of the colored *Line Set View* and the associated project view.

Note that the distance map `coreSample.DistField` - created by *Auto Skeleton* with a *Distance Map For Skeleton* module - is extended with a `15` voxel border as required by the *Thinner* module in this version when attached to data in memory.

5.5.3 Skeletonization step-by-step

We will now achieve step-by-step skeletonization. This can be useful for advanced users for choosing different distance map settings, or controlling the sensitivity of the *Thinner* module to generate branch points. Let's start by creating a label image. Any segmentation process can be used:

- Remove all objects from the *Project View* (you can right-click in the *Project View* , then select `Remove All Objects`).
- Choose *File / Open Data...* from the *File* menu.
- Load the file `data/core/coreSample.am` from the Avizo root directory.

- Attach a *Multi-Thresholding* module to the data set by selecting *Image Segmentation / Multi-Thresholding* in the data context menu.
- Adjust the **Exterior-Range1** threshold port of *Multi-Thresholding* module to 27.
- Press the `Apply` button to create a label image `coreSample.Labels`.

The *Auto Skeleton* module internally uses a *Image Processing / Distance Maps / Distance Map For Skeleton* which in turn internally creates a *Image Processing / Distance Maps / Distance Map* module with default settings for chamfer distance. In addition, *Distance Map For Skeleton* adds to the data a background border, which is required by the *Thinner* module.

We will do this step-by-step using the *Image Processing / Distance Maps / Distance Map* module. Other distance maps such as those provided by Avizo could be used in a similar way.

- Attach a *Distance Map* module to `coreSample.Labels` by selecting *Image Processing / Distance Maps / Distance Map* in the data context menu.
- Set **Type** port to `Euclid` distance.
- Set **Region** port to `Both (unsigned)`.
- Press the `Apply` button to create a distance map.
- Attach an *Ortho Slice* module to the distance map.
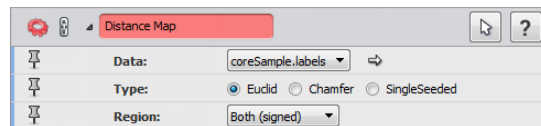
Figure 5.50: Euclidean distance map module parameters.
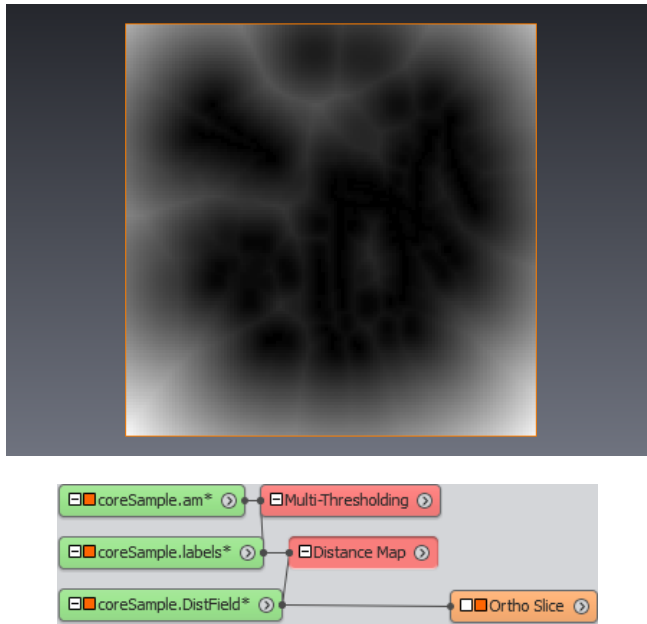
Figure 5.51: Visualization of the distance map and the associated project view.

Euclidean distance may be more accurate, yet much slower to compute than the Chamfer distance. Chamfer distance is generally fine for most applications. In particular, it makes little difference with the data used in this tutorial.

The result of a Euclidean distance map is a scalar field with floating point values. The *Thinner* module expects the distance map input to use positive short integers. An additional `15` voxels border must also be added in the current implementation. The following steps are needed to adapt the distance map data:

- Attach a *Convert Image Type* module to `coreSample.DistField` by selecting *Convert / Convert Image Type* in the data context menu.
- Set **Output Type** port to `16-bit unsigned`.
- Set **Scaling** port `scale` value to `1000`. The scaling used here has no other importance than preserving significant digits of input data for driving the *Thinner* algorithm - still fitting in a positive short integer. The actual thickness value can be retrieved later on from the original distance map.
- Press the `Apply` button to create a distance map data using positive short integers.
- Select the result in the module workspace `coreSample.to-ushort`.
- Click on the `Crop Editor` button in the Properties Area.

- In the *Crop Editor* dialog set **Min index** to −15 −15 −15 and **Max index** to 142 142 142.
- Uncheck **Add mode** port Replicate and set **Pixel value** to 0.
- Press the OK button: this will extend the image with a border of 15 voxels.
- As an alternative to the *Crop Editor*, you could also type in the console the command:
  coreSample.to-ushort crop −15 142 −15 142 −15 142 0

We can now use the *Thinner* module.

Note: The thinning algorithm automatically detects dead end branches of skeleton spatial graphs. A parameter is used to distinguish them from noise on the interface of the considered regions to avoid spurious branches. Its default value is 5, i.e., the branches with a dead end which length is lower than 5 voxels are automatically considered as noise and removed. Setting this to 10, which is a rather large value, leads to only a few branches remaining in the skeleton. The drawback is that you also might miss real endpoints. Generally speaking, it is easier to remove spurious branches directly during thinning that trying to remove them later.

- Attach a *Thinner* module to label by selecting *Image Processing / Skeletonization / Thinner* in the data context menu of coreSample.Labels.
- Attach the **Distmap** input of the *Thinner* module to coreSample.to-ushort.
- You may check **Extended Options** in order to reduce the number of branches. The default **len of ends** value is 2 with *Auto Skeleton* module. Set its value to 5. You can also limit execution time with large complex data by fixing the number of iterations to perform.
- Press the Apply button to create a thinned image data.

Next, we can build a spatial graph from the skeleton and display it with geometrical information:

- Attach a *Trace Lines* module to coreSample.thinned by selecting *Image Processing / Skeletonization / Trace Lines* in the data context menu.
- You may uncheck point cloud in **Options** port as we won't need the point cloud.
- Press the Apply button to create a *Spatial Graph* object.
- Attach an *Eval On Lines* module to *coreSample.Spatial-Graph* by selecting *Compute / Eval On Lines* in the data context menu.
- Attach the **Field** input port of *Eval On Lines* module to coreSample.DistField, the float distance field computed earlier. *Eval On Lines* can map any scalar field on *Spatial Graph*.
- Next you need to type the command "Eval On Lines" setZeroCorrection 0 in the console. Since **ChamferMapScaleFactor** is not defined in distance map data parameters, *Eval On Lines* applies a default 1/3 voxel-size factor to the input value. *Eval On Lines* also forces a minimum value defined by zeroCorrection * voxel size (half voxel size by default).
- Press the Apply button of *Eval On Lines*. The *Eval On Lines* module doesn't create a new data icon. It is more like an editor and changes the connected line set. It adds a data value for every vertex in the line set and calculates the value of the field at the point of the vertex.

- Attach *Spatial Graph Statistics* and *Spatial Graph View* to *Spatial Graph* to display 3D graph and statistics as shown in the "getting started" section.
- In order to improve visualization of the Spatial Graph, click on the module *Spatial Graph View*. Toggle off *Node*. In *Segment*, change the port *Segment Coloring* to *thickness* and in the port *Segment Colormap* click on *Edit* and *Adjust range*.
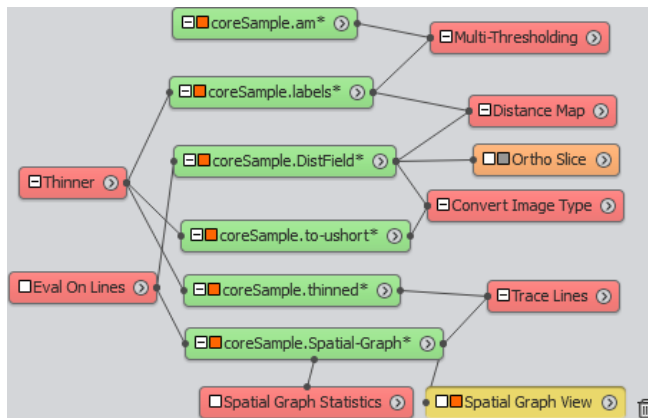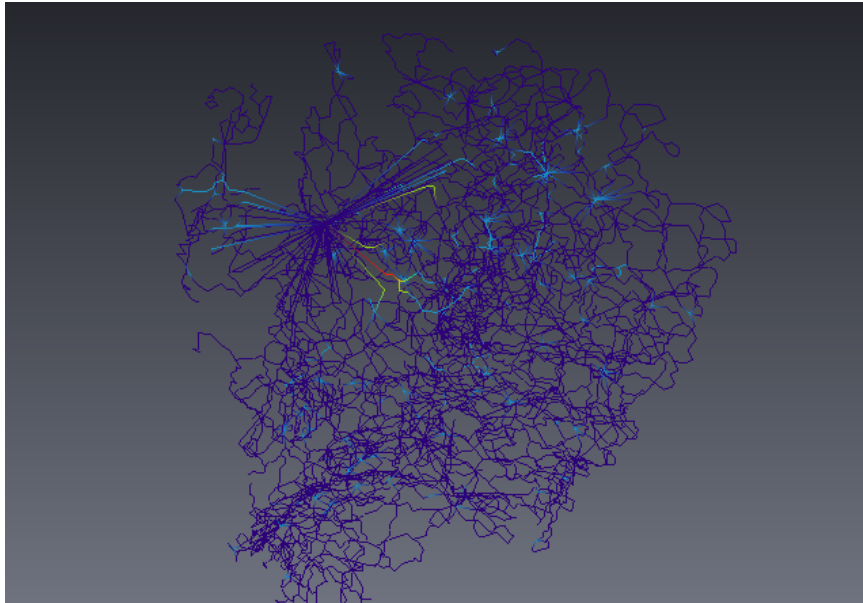
Figure 5.52: Visualization of the skeleton as a spatial graph and the associated project view.

5.5.4 Skeletonization with large data

This section is about achieving skeletonization on very large data that may not fit in memory. For the next tutorial steps, you should have access to a directory to which you're allowed to write intermediate files.

5.5.4.1 Preparation for using Large Disk Data

The modules *Auto Skeleton*, *Distance Map For Skeleton*, *Thinner*, *Trace Lines*, *Eval On Lines* can operate directly on data on disk, allowing processing of data that cannot fit in memory. Another distance map module *Distance Map On Disk Data* is also available for operating on disk data.

These modules currently support reading input and writing image results using the former *Large Disk Data* (LDD) format. For converting image data to LDD, you can use *Convert To Large Data Format*, accessible by right-clicking in project view, then selecting *File / Convert To Large Data Format*. Prior to conversion, you'll need to type `"Convert To Large Data Format"` `forceAmira311Format 1` in the console in order to convert the data to a format compatible with skeletonization modules.

The *Auto Skeleton* module and *Thinner* algorithm on disk data also expect a black border around the data. This border should be at least of size `len of ends` used during thinning (see the section above). If the **BorderWidth** parameter is not specified on the disk data object, a default border is automatically created with size of `len of ends`. The *Auto Skeleton* module expects disk data object to specify a *data parameter BorderWidth*. When using *Thinner* with disk data, you can set the `len of ends` parameter manually in the console, for instance: `Thinner setVar lenOfEnds 10`. This will set the maximum length of branches to `10` voxels before they are detected as unconnected ends.

Avizo data sets may be saved as separate blocks, which can be useful for multiple acquisitions or large data sets split into separate files. Avizo has a special data object (*Mosaic*) for storing links to files on disk, arranging them in 3D, and other operations on these blocks such as alignment and template operations like filtering or resampling.

Note: There is no way to convert the new multiresolution LDA format to LDD format directly. A possible solution is to use *Extract Subvolume* to extract and save separate blocks, and use *Mosaic* to convert these to LDD format.

For now, we will simply use *Mosaic* as a convenience for converting our data to LDD format while adding the border necessary for skeletonization process. For more general information about using large data with Avizo, see section : Working with out-of-core data files (LDA).

- Press [CRTL+N] to start a new empty project.
- Create a *Mosaic* by selecting *Images And Fields / Mosaic* from the *Project >Create Object...* menu of the Avizo main window.

A green icon appears. A *Bounding Box* display module also appears. When you select the green module, you see that it contains no bricks (0 bricks in the **Info** port). The buttons below the **Info** port

are used to add data objects.

- Press the *add files* button.
- In the *Load* file dialog appearing, select the file `data/teddybear/teddybear.info` from Avizo root directory.
- Press *Load*.

The selected file is added to the *Mosaic*. The **Info** port shows the single bricks added. You can visualize the brick outline with a *Mosaic Outline* module.

- Create a *Mosaic Outline* module by right-clicking on the *Mosaic* and selecting *Display / Mosaic Outline* from the context menu.
- Switch off the *Bounding Box* module.
- You may save the data.
- You can check information stored by *Mosaic* object by clicking on its *Data Parameter Editor* button in the Properties panel.
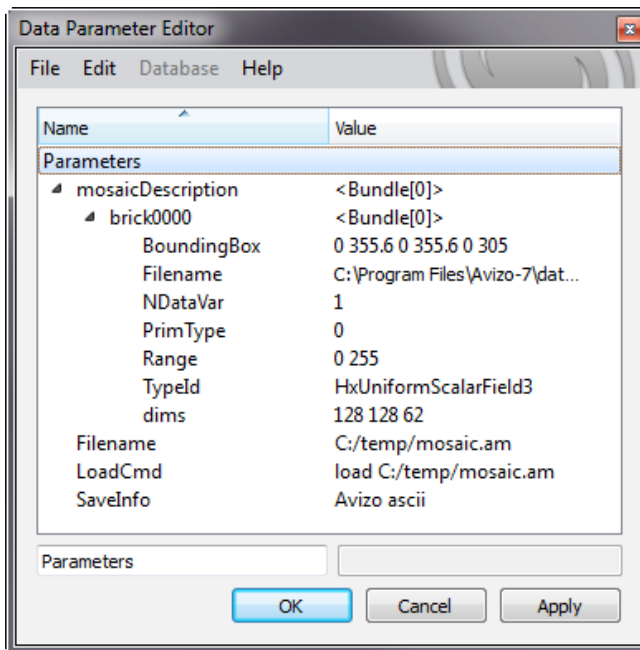
Figure 5.53: Mosaic's parameter editor after saving the data

The next step is to create a new Large Disk Data object and sample the brick onto it. With multiple

bricks, the overlapping regions could be blended with each other. A border can also be added.

Note: The thinning algorithm expects a black border around the data. The border should be at least of size `len of ends` used during thinning (see below). By default, a border of `15` voxels on each side in each dimension. Be sure to check this if you manually set `len of ends`.

- Attach a *Mosaic To Large Disk Data* to the Mosaic by right-clicking on the *Mosaic* and selecting *Convert / Mosaic To Large Disk Data* from the submenu in the context menu.
- The red *Mosaic To Large Disk Data* icon should be selected so the module ports appear in the Properties area.

You can see several options in the Properties Area. The default options are fine for the tutorial, while the border could be set to a lower value.

- You need to set the **Filename** port to a path to output LDD file. Write access is, therefore, needed. If you saved the *Mosaic* data object on disk, a default filename derived from the mosaic file location is displayed in the **Filename** port. You might want to override it.
- Leave the other ports as they are by default and press the `Apply` button.

A new green icon which represents the new *Disk Data* data object will appear in the Project View. After this, the bricks will be loaded one after the other and will be sampled. This may take some time for large data.

- Select the new green icon (named `image`).

Some information about the data stored on disk is displayed in the Properties Area. Next,

- Delete or switch off the *Mosaic Outline* module.
- Connect a *Bounding Box* to the `image` icon.
- You can directly visualize the Large Disk Data `image` by attaching an *Ortho Slice*.

The second box is slightly bigger than the bounding box of the *Mosaic*. This is due to the border added by the *Mosaic To Large Disk Data* module.
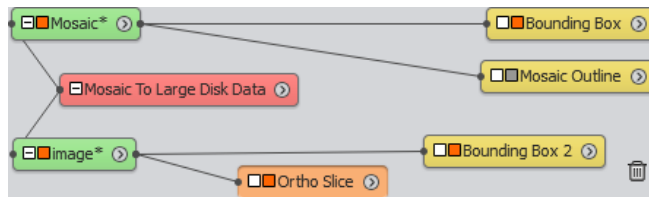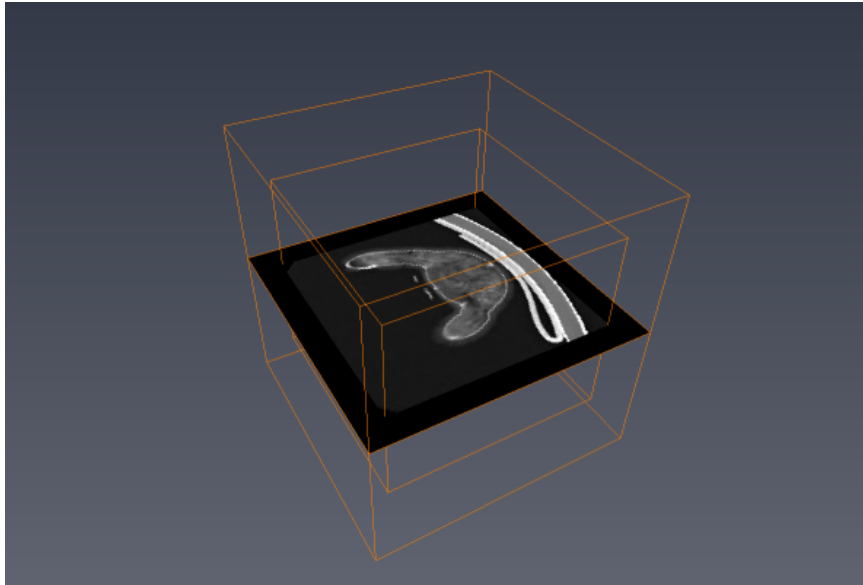
Figure 5.54: Visualization of Large Disk Data and the associated *Project View*

5.5.4.2 Using Large Disk Data for skeletonization

A few display modules are available for visualization of LDD data (note that the LDA disk data format offers more powerful visualization capabilities). Several computation modules are also able to handle the Large Disk Data directly. These include thresholding, computation of a distance map, thinning, extracting a spatial graph set from a voxel skeleton, and computation of the thickness of the lines (evaluating the distance map at the points of the line set). You can, therefore, apply skeletonization directly to large disk data. For other operations, you may need to load a subvolume into the workspace as an in-memory object, or use Avizo modules for on-disk processing.

The next step is to apply a simple thresholding.

- Attach a *Threshold* to the *image* icon by right-clicking on it and selecting *Threshold* from the *Image Segmentation* submenu in the popup menu.

- Set **Threshold** to `140` in the Properties Area.
- Select a filename to which you want to store the result. In the tutorial, we will use the default name `image.labels`.
- Press the `Apply` button.

A new green icon that contains the labels will appear. Connect an *Extract Subvolume* module to this new data, click on the buttons `Max width`, `Max height` and `Max depth` to fit the volume to extract to the complete volume. Finally, click on the `Apply` button. Connect an *Ortho Slice* to the extracted volume to visualize the result of the threshold.

You might want to correct the result of the segmentation procedure manually. This might be useful to fill big vessels or remove uninteresting parts. Avizo has a *Segmentation Editor* to perform this task. Due to the size of the data, you will have to work on subblocks of the whole data set.

In the next step, we'll use *Auto Skeleton* on disk data.

- Attach an *Auto Skeleton* module to `image.labels`
- At this point, you may modify the location of intermediate and result data files created.
- Press the `Apply` button.
- Move and orient the two *Ortho Slice* module in order to obtain Figure 5.55.

You could also achieve step-by-step skeletonization, computing distance maps with either *Distance Map For Skeleton* or *Distance Map On Disk Data module*, using *Thinner*, *Trace Lines* and *Eval On Lines* manually, in a similar way to processing memory data.

Here is an example showing skeletonization, which uses a script object module that can be attached to any label image: "Porosity network reconstructed", located in demo/core/coreSkeleton.hx, that can be accessed from the *Examples* help menu (in Rock Core Sample Analysis demos).
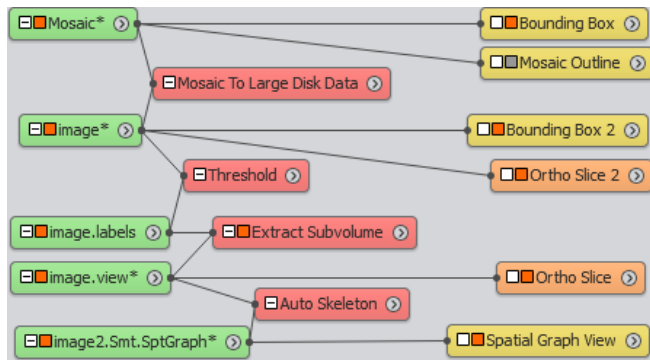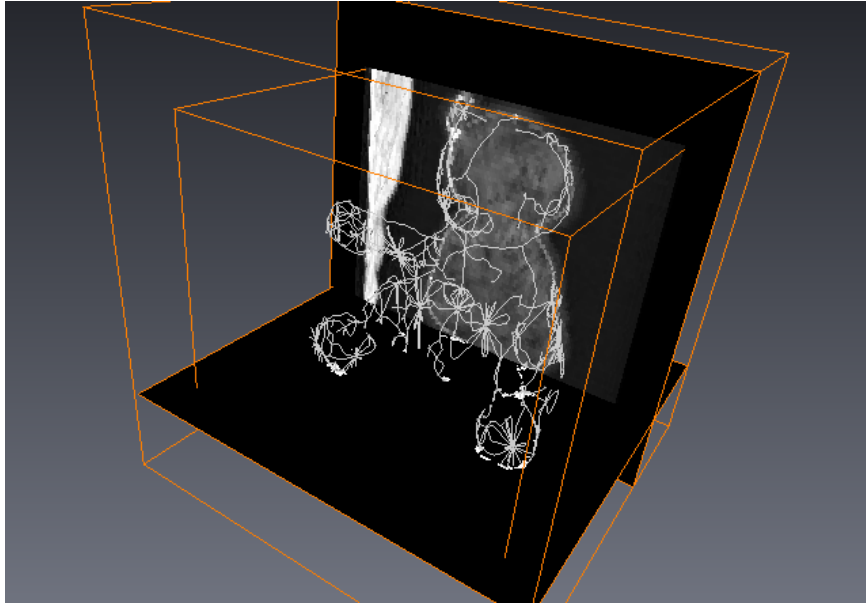
Figure 5.55: Visualization of the skeleton of the LDD and the associated project view.

Chapter 6

Registration, Alignment and Data Fusion

Spatial registration is about aligning or overlaying two or more data sets into the same coordinate system. In registration, typically one of the data sets is taken as the reference, and the other one is transformed - moved and possibly rescaled - until both data sets match. Registered data can be produced by different sensors, at different times, from different object regions, or from different specimens or models. Image registration methods can be manual, automatic, or semi-automatic. Closely related to registration is the task of data fusion, i.e., the simultaneous visualization of registered data sets, or combination into derivative data.

A variety of tools related to registration can be used in Avizo depending on your purpose and on your data - geometric surfaces, 2D image stacks, or 3D volumes. Registration with Avizo is used in a wide range of applications, including:

- Industrial inspection of products with respect to reference models and nominal/actual analysis and reverse engineering,
- Multi-modality image acquisitions such as CT/ MRI (Computed Tomography, Magnetic Resonance Imaging),
- FIB-SEM/μ-CT (Focused Ion Beam-Scanning Electron Microscope, micro-tomography),
- Correlative microscopy,
- 3D image reconstruction from 2D cross sections,
- Imaging of physical experiments or processes - e.g., samples subjected to heat, flooding, compression,
- 3D montage assembly - merging 3D volumes with small overlap.

The following tutorials and examples provide the basics for typical registration tasks.

- *Getting started with spatial data registration using the Transform Editor*
- *Data fusion, comparing and merging data*
- *Registration with landmarks, warping surfaces and images*
- *Registration of 3D image data sets*
- *Registration of 2D image and 3D image data sets*
- *Alignment of 2D images stacks*
- *Alignment and pre-processing of FIB/SEM images stacks using the FIB Stack Wizard*
- *Registration of 3D surfaces*
- *Registration of 3D image and surface, nominal-actual analysis*

You should be familiar with the basic concepts of Avizo to follow these tutorials. In particular, you should be able to load files, to interact with the 3D viewer, and to connect modules to data modules. All of these topics are discussed in Avizo chapter 2 - Getting started. The tutorial section 6.1.1 Using the Transform Editor is recommended as a starting point in most cases. You can then jump to the topic for your specific task. In particular, the 2D Slice Alignment tutorial may be read independently.

These tutorials cover common use cases. For specific requirements or applications that differ from these use cases, you may contact the FEI Hotline for further discussion.

6.1 Getting started with spatial data registration using the Transform Editor

In this section, you will learn how to: change the spatial position of data objects interactively using various manipulators; how to specify numerically a geometric transformation as a combination of translation, rotation, and scaling; how to copy/paste geometric transformations; how to apply geometric transformations to data in order to change actual data coordinates or voxel image axis alignment; and where to find related tools.

This section has the following parts:

- *Using the Transform Editor*
- *Applying Transforms*
- *Numerical input, console and script commands*
- *Transform Manipulators*

6.1.1 Using the Transform Editor

Spatial data visualized within Avizo are placed in a virtual three-dimensional world. That world has a unique coordinate system. Every spatial object in Avizo can be arbitrarily translated relative to the

world origin; likewise it can be rotated with respect to the global axes, and it can be independently scaled (enlarged or shrunk).

Select *View / Global Axes* to display the global coordinate axes. Global axes are centered at the origin of the world coordinates. By default, the x-, y-, and z-axes are drawn in red, green, and blue, respectively. You can also use the viewer's compass to see 3D space orientation. Select *Edit / Preferences / Layout* to change Compass settings.
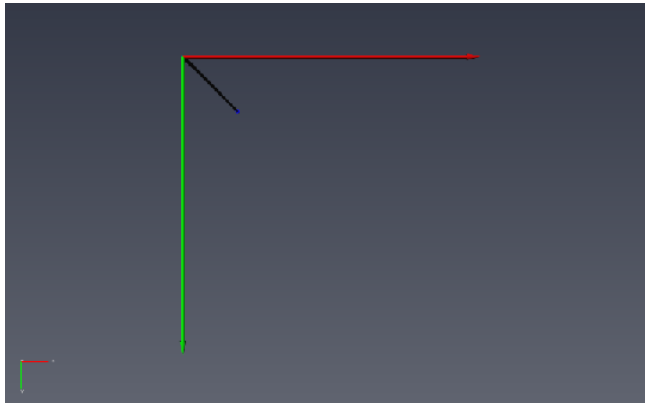
Figure 6.1: 3D axes and compass

Every spatial data object in Avizo has an associated 3D bounding box as well as an optional geometric transformation, which can be defined as a combination of translation, rotation, and scaling (internally represented as a 4x4 homogeneous affine transformation matrix).

Rotating a scene within the viewer in "trackball" mode (hand mouse cursor) does not alter the object position or orientation relative to world coordinates. Rather, it changes the point of view of the camera around the whole scene. In order to change the geometric transformation of an object, i.e., translate, rotate or scale it with respect to other objects or to world coordinates, a *Transform Editor* is available for every spatial data object.

In the following example, we use the *Transform Editor* to manipulate surface data objects. This example would also apply to images, 3D volumes, or regular scalar fields, using modules such as *Ortho Slice* for display.

- Start a new Avizo Project (menu *File / New Project*, or press Ctrl-N).
- With menu *File / Open Data*, load the `chocolate-bar.simplified` surface from the *data / tutorials* subdirectory in the Avizo installation directory.
- Attach a *Surface View* module to `chocolate-bar.simplified` data set (i.e., choose *Surface View* entry from the popup menu of the data set).
- Duplicate the data object. For this, you can select `chocolate-bar.simplified` and

press Ctrl-D or right-click and use data menu *Object / Duplicate Object*. The result is `chocolate-bar2.simplified`.

- Attach a *Surface View* module to the copy `chocolate-bar2.simplified`. You can still only see one engine shape in the viewer for now since the two data sets are overlaid.
- Select `chocolate-bar2.simplified` in the Project View.
- In the *Properties Area*, invoke the *Transform Editor* by clicking on the dragger box button. A manipulator appears around the engine surface in the viewer window: it allows you to change the transformation in 3D. The white square on the left of the data object icon in the Project View is changed to blue, indicating that an editor is active.
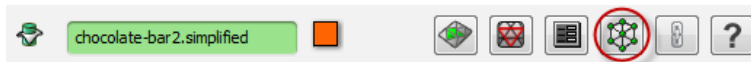
Figure 6.2: Invoking the Transform Editor

- Make sure to switch the viewer to interaction mode: press the arrow button in the upper left corner of the viewer, or toggle between viewing mode and interaction mode using the ESC key.
- Then click and drag a side face of the manipulator: one of the two surfaces that are currently displayed is translated along the face plane. The data object label is then displayed in italics to indicate that the data or its attached information has been modified. A manipulator has several dragger gadgets for controlling the transformation in various ways. Details of how to interact with the manipulator draggers can be found at the end of this *Transform Editor* tutorial.
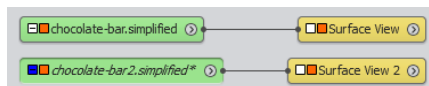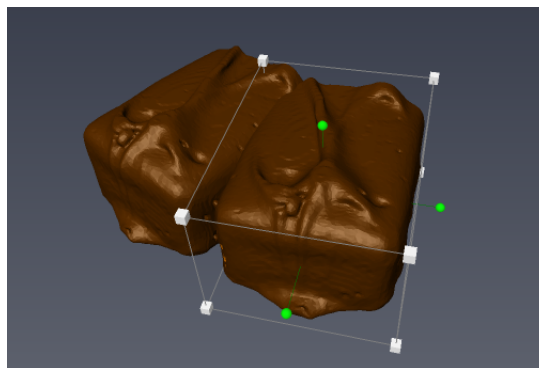
Figure 6.3: An active Transform Manipulator

In the *Properties Area*, the active *Transform Editor* adds several button ports to the data object.
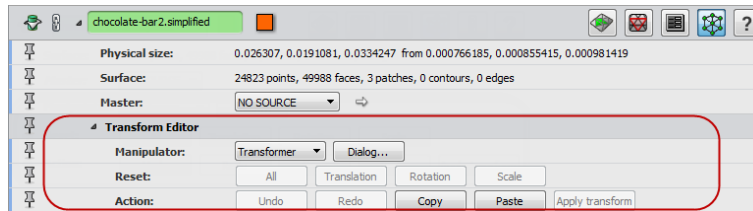
Figure 6.4: Transform Manipulator ports

- The *Manipulator* port lets you choose among several interactive manipulators, or select a Dialog to input numerical values. These are described later in this tutorial.
- The *Reset* button list lets you restore the independent components of a transformation, e.g., canceling the rotation part. Using the *Action* port, it is possible to Undo/Redo the last transformation change.
- You can copy/paste a transformation from one data set to another. For instance:
  - Press the *Copy* button in the *Transform Editor*'s *Action* port of `chocolate-bar2.simplified`.
  - At this point, you can optionally deactivate the *Transform Editor* by clicking on its dragger box button.
  - Select the first loaded data set, `chocolate-bar.simplified`, and activate its *Transform Editor*.
  - Press the *Paste* button in the *Action* port of `chocolate-bar.simplified`: both data sets are again overlaid in the same location, the latest position of `chocolate-bar2.simplified`.

In order to copy transformation from one data set to another, you could alternatively use the module *Copy Transformation* from the object menu *Geometry Transforms*. The *Geometry Transforms* menu contains most of the tools related to registration, alignment, and transforms.

The *Apply Transform* button that you may have noticed in the dialog, commits the transformation. This topic is explained in more detail in the next section.

To facilitate visual control during interactive transformations, you may create an additional viewer, adjust the camera for convenient interaction, and control transformation in the other viewer.

6.1.2 Applying Transforms

At this point, there is an important concept to know about geometric transformations in Avizo. The geometric transformation associated with a data object is taken into account by display modules and

some other modules. However, it does not modify the original coordinates of data unless explicitly requested. That is to say, how the object is positioned in the 3D world is updated with transformations, but the coordinates stored in the data object are not updated. For instance, the bounding box of a 3D volume or the point coordinates of a triangulated surface - sometimes referred to as data "local coordinates" - as stored in memory remain unchanged by the geometric transformation. This geometric transformation simply "presents", when needed, the transformed world coordinates to the attached modules.

It is necessary, in some cases, to actually apply the transformation in order to change the data's initial local coordinates into world coordinates. For instance:

1. Before exporting data to non-Avizo file formats. When saving data to Avizo format and a few other formats, the geometric transform is stored at the same time in the file and can therefore be restored when reloading the data. However, be careful when saving or exporting transformed data sets. Most file formats do not allow you to store geometric transformations. In this case, you must apply the current transformation to the data prior to saving it. Nevertheless, when saving a project, the project file stores transformations for the referred data.

2. When some data processing or manipulation does not support geometry transformation, it requires transformed data coordinates. For instance, the lattice of a transformed volume image or regular scalar field may need to be aligned with global axes or with respect to another lattice. This is useful to perform certain lattice-aligned operations as illustrated below.

Here is a first example showing how to apply a transformation to the surface data that you already used in a previous section:

- If needed, load the supplied `chocolate-bar.simplified` surface data file from the `data/tutorials` directory; attach a *Surface View* module to `chocolate-bar.simplified`; and display the global axes with menu *View / Global Axes*.

- Attach a *Local Axes* module to the data object from the popup menu *Annotate*.

- Use the *Transform Editor* to transform the surface `chocolate-bar.simplified` with at least some rotation. For instance, pick and drag one of the green spherical knobs of the transformer manipulator.

- Press the *Apply Transform* button of the *Transform Editor*. Note that this operation cannot be undone. In case of *vertex set* objects like surfaces, the transformation is applied to all vertices. Old coordinates are replaced by new ones, and the transformation matrix is reset to identity afterwards, i.e., there is no longer any transformation set for the modified surface. After a transformation has been applied to a data set, the transformation can no longer be easily unset.
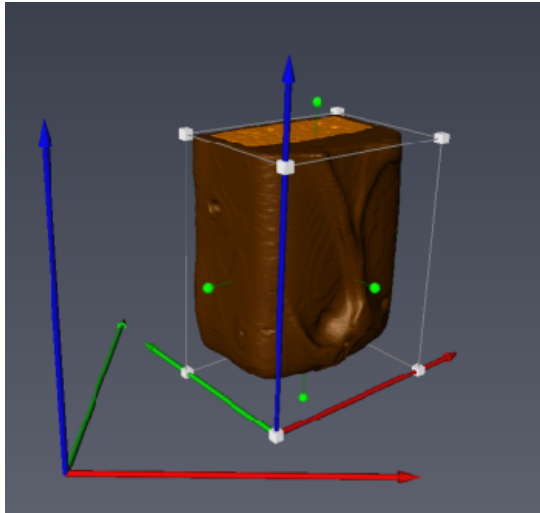
Figure 6.5: Before applying transform to surface
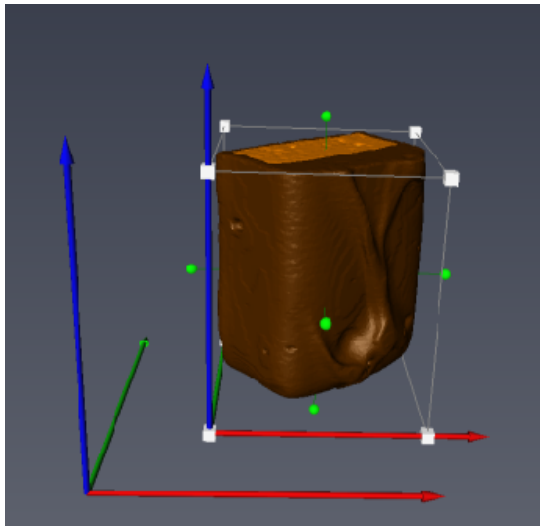
Figure 6.6: After applying the transform

Applying transformation to image

Let's now see how to apply a transformation to a volume data. As in the example above, this will

make the volume data appear to stay at the same location, although without any transformation set. The *Transform Editor*'s *Apply Transform* button is not available for images, regular volumes, or scalar fields. You must instead use the *Resample Transformed Image* module as in the following example:

- Load the image stack file `chocolate-bar.am` located in subdirectory `data/tutorials`.
- Attach modules *Ortho Slice*, *Bounding Box*, and *Annotate / Local Axes* to the data object `chocolate-bar.am`; display the global axes with menu *View / Global Axes*.
- Use the *Transform Editor* to move `chocolate-bar.am` with at least some rotation. When done you may deactivate the *Transform Editor*.
- Attach a compute module *Resample Transformed Image* to `chocolate-bar.am` from the object menu *Geometry Transforms*.
- In the *Properties Area*, set the *Mode* port of *Resample Transform Image* to "extended" in order to make sure that the result will contain all the source data.
- Press the green *Apply* button at the bottom of the *Properties Area*. The compute module produces a new data object `chocolate-bar.transformed` added in the Project View. The input data `chocolate-bar.am` remains unchanged.
- To visualize the result, you can attach to it a *Bounding Box*, *Local Axes*, and *Ortho Slice* modules. The result volume and attached *Ortho Slice* are now aligned along global axes. The input volume data has been resampled over a lattice with new bounding box coordinates.
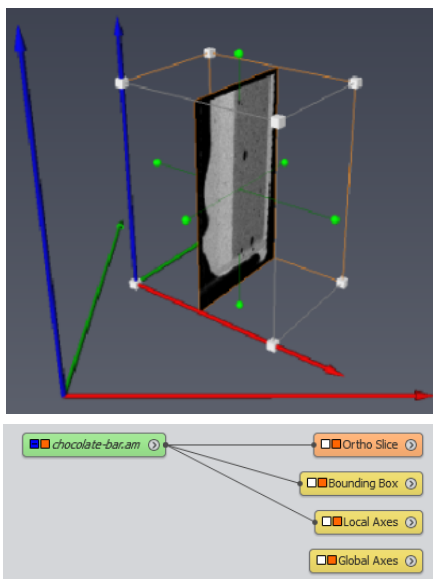
Figure 6.7: Before applying transform to image
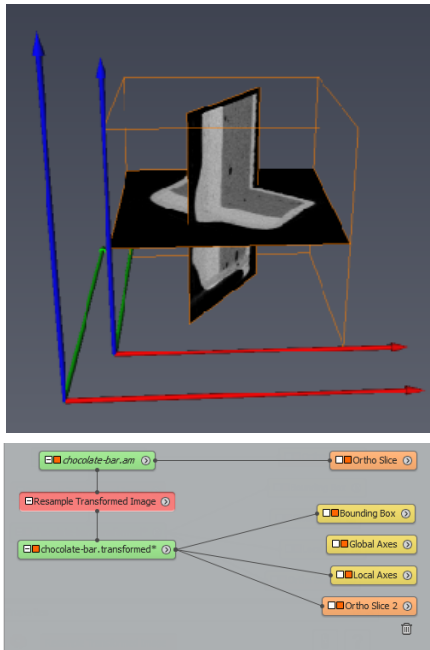
Figure 6.8: After applying Resample Transformed Image

The *Resample Transformed Image* module can be used for sampling a volume data onto a reference lattice connected as input. Another useful feature of this module is to reorient volume data along a given plane by sampling the data on a lattice parallel to this plane. The plane can be set using a *Slice* or *Surface Cross Section* module, by arbitrary rotations, by picking three points, or by a point set to be fitted.

Applying transformations can also be performed by using the Tcl command `applyTransform` in the Avizo console or in a script. Tcl commands relating to transformations are introduced in the next section.

6.1.3 Numerical input, console and script commands

(This section is optional and is not required reading for completing the subsequent registration tutorials.)

For inputting a precise spatial transformation, it is often necessary to access the numerical values of the transformation. The *Transform Editor* also lets you check or enter transformations numerically.

- The *Dialog...* button of the *Transform Editor* pops up the *Transform Editor* dialog.
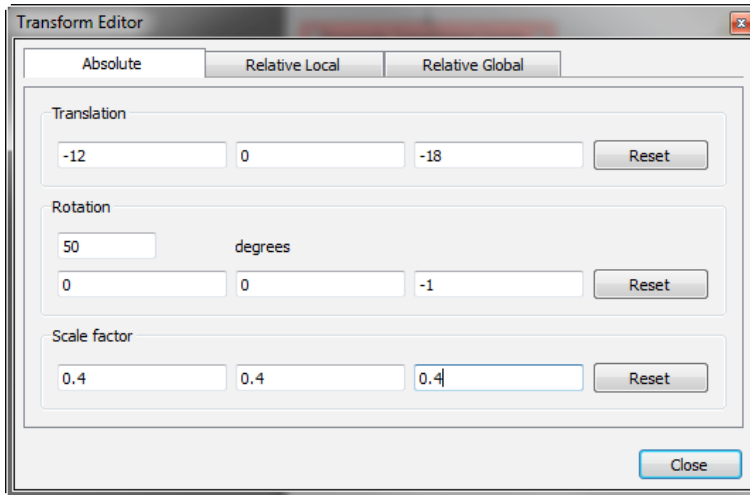
Figure 6.9: Transform Editor Dialog

Advanced users can alternatively retrieve the numerical values of a transformation, by using the Tcl command `getTransform` in Avizo console (see *spatial data* reference for more details on available commands). The transformation is then printed in the console as the list of matrix values. For instance:

```
>"chocolate-bar.simplified" getTransform
0.257115 -0.306418 0 0 0.306418 0.257115 0 0 0 0 0.4 0 -4.30079 23.5849 -6.7025 1
```

Similarly, a transformation which matrix is known can be set by:

```
>"chocolate-bar2.simplified" setTransform 0.257115 -0.306418 0 0 0.306418 0.257115
0 0 0 0 0.4 0 -4.30079 23.5849 -6.7025 1
```

This provides another way to copy/paste transformation between objects. It can be done using a single command as follows:

```
>eval "chocolate-bar2.simplified" setTransform ["chocolate-bar.simplified" getTransform]
```

The components of the transformation can be obtained in human-readable form in the *Transform Editor* dialog, or by using the `getTransform` command with the option `-d`:

```
>"chocolate-bar.simplified" getTransform -d
translation: -12 0 -18
rotation: 0 0 -1 50
scaleFactor: 0.4 0.4 0.4
scaleOrientation: 0 0 1
center: 20.0481 23.4785 18.8292
```

Getting started with spatial data registration using the Transform Editor 272

As explained in the previous section, the transformation is at first set transiently and can be stored by saving the project, but is not a property of the object. To make the transformation permanent, enter:

```
<data> applyTransform
```

and then save the data object. A module is also available for this purpose: *Geometry Transforms/Resample Transformed Image*. For instance, type in the Avizo console:

```
>"chocolate-bar.am" applyTransform
```

The `chocolate-bar.am` data is changed to a resampled volume and the transient transformation is reset to zero. In order to provide more flexibility on the resolution of the output grid and the type of resampling, use the module *Resample Transformed Image* introduced in a previous section.

6.1.4 Transform Manipulators

(This section is optional and is not required reading for completing the subsequent registration tutorials.)

Many manipulators are available in the *Transform Editor*. Interaction with the different manipulators is detailed hereafter.

- The default manipulator is the *Transformer*. It allows translations, rotations, and scaling. It is the most general manipulator for doing an approximate registration.
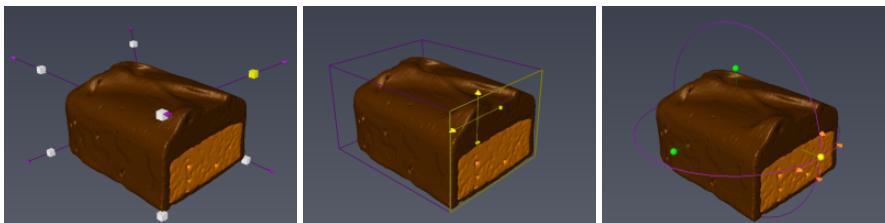
Figure 6.10: Transformer: Click-drag a corner cube to scale (hold Shift key to constrain direction, hold Ctrl key to fix opposite corner or face). Click-drag any face to translate (hold Shift key to constrain direction, hold Ctrl key for perpendicular translation). Click-drag a green ball to rotate one way (hold Shift key for free rotation, hold Ctrl key to change center).

- The *Jack* manipulator is convenient for translations along an axis and uniform scaling.
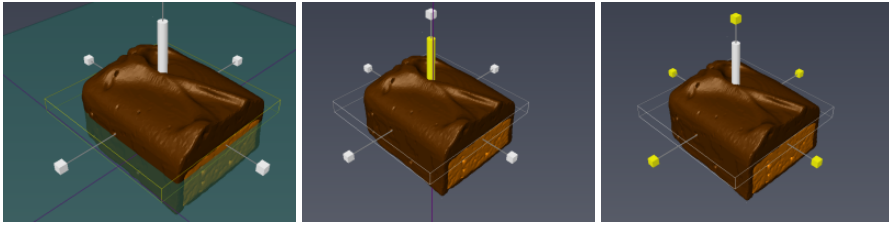
Figure 6.11: Jack: Click-drag rectangle or cylinder rod to translate. Press Ctrl key to change axis. Click-drag cubes to scale. Click-drag axial lines to rotate.

- The *TransformBox* is a simplified version of the *Transformer* that allows translation, rotation and uniform scaling.
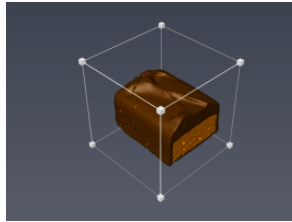
Figure 6.12: TransformBox: Click-drag any small cube to scale. Click-drag any face to translate. Hold Shift to constrain axis. Click-drag any box edge to rotate.

- The *Trackball* and *Centerball* allow rotation.
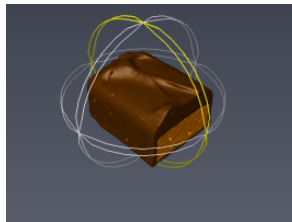
Figure 6.13: Trackball: Click-drag stripes to rotate. Click-drag anywhere to rotate freely. Hold Ctrl key to scale. Hold Shift key for user axis and stripe.
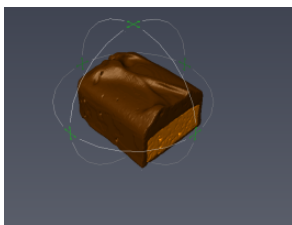
Figure 6.14: Centerball: Click-drag circles to rotate. Click-drag anywhere to rotate freely. Click-drag green arrows to translate rotation center (Hold Shift key to constrain translation axis).

- The *HandleBox* allows translation, uniform scaling and anisotropic scaling. It is the most suitable manipulator for anisotropic scaling operations.
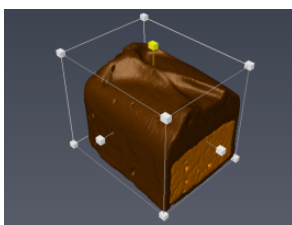
Figure 6.15: HandleBox: Click-drag any face to translate (Hold Shift key to constrain translation axis). Click-drag a small cube to scale.

- The *TabBox* allows translation and anisotropic scaling by box resize. It is used by modules such as *Extract Subvolume* or *ROI Box* (Region Of Interest).
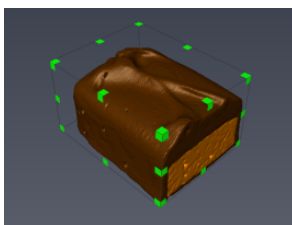
Figure 6.16: TabBox: Click-drag any face to translate. Click-drag green vertex to modify the bounding box.

6.2   Data fusion, comparing and merging data

The task of image fusion is the simultaneous visualization and analysis of two data sets. This tutorial introduces some of the basic tools and techniques available in Avizo to that end, typically used with

registered or aligned data sets. Avizo makes it easy to manipulate multiple data in single viewer or in multiple views, as well as to combine different data sets in the same visualization.

This section has the following parts:

- *Color Wash*
- *Ortho Views*
- *Mapping a 3D volume overlaid on a surface*
- *Side-by-side viewers, synchronized views and objects*
- *More about Data Fusion*

6.2.1   Color Wash

The *Color Wash* module, attached to an *Ortho Slice*, helps you to visualize two arbitrary images or volumes in combination. The representation of one data set is overlaid with another data set, taking into account their locations in space. The rendering of the *Ortho Slice* is modulated so that it also encodes the second data set.

- Load the image stack file `chocolate-bar.am` located in subdirectory *data / tutorials/*.
- Load the file `chocolate-bar.labels.am` from directory *data / tutorials*. The data set `chocolate-bar.labels.am` contains labeling of different regions, obtained by a segmentation of the grayscale image.
- Attach an *Ortho Slice* module to `chocolate-bar.am`.
- Attach a *Color Wash* module to the *Ortho Slice* module: right-click on *Ortho Slice* icon in the Project View and select *Color Wash* in the popup menu.
- Select the yellow icon of the *Color Wash* module.
- Connect the *Data* input port of the *Color Wash* module to the second data object, `chocolate-bar.labels.am`. To do this, you can set the port *Data* in the *Properties Panel*.
- You can change the *Colormap* port of *Color Wash* module to adjust the range. The *Weight Factor* port allows adjustment of the blending between the two data sets. You can also try other *Fusion Method* such as *Magic Lens*, useful for checking data alignment.
- Activate the *Transform Editor* of `chocolate-bar.am` and move the data set. You can also change the *Ortho Slice* orientation or slice number to control the image in a different plane.
- Note that the *Color Wash* data doesn't need to be the same size or resolution as the *Ortho Slice* data - data can be unrelated as far as they overlap spatially. For instance, load the image stack file `coreSample.am` located in subdirectory *data / core /* and attach it as *Color Wash* data instead of `chocolate-bar.labels.am`.
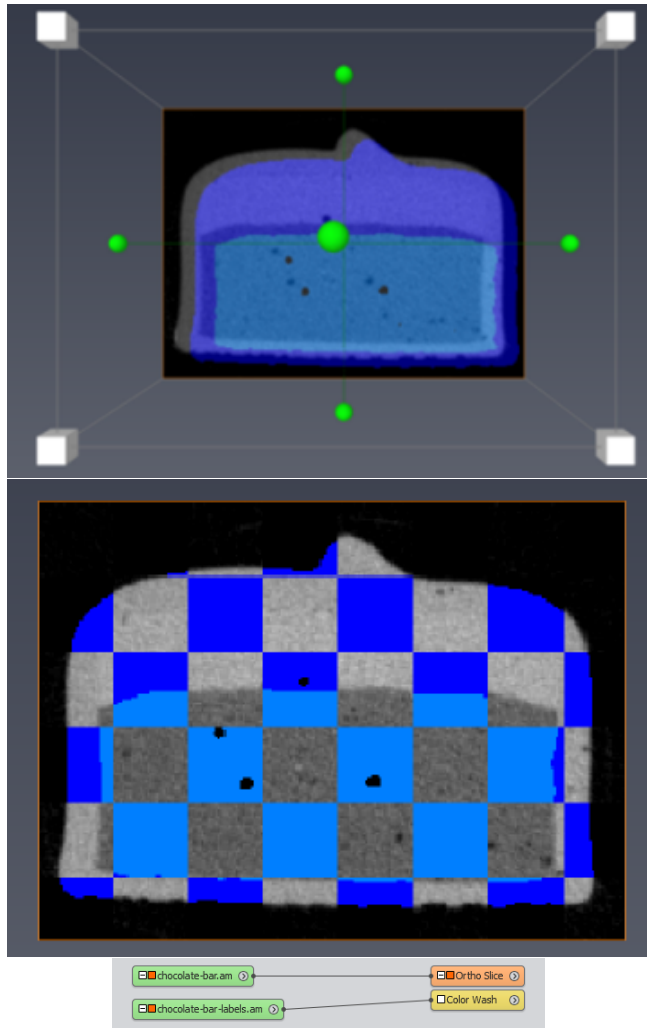
Figure 6.17: Color Wash fusion method: examples with Weighted Sum, Magic Lens

6.2.2 Ortho Views

In image fusion it is sometimes necessary to observe all three orthogonal directions simultaneously. For this we can use the powerful *Ortho Views* module. See also the related tutorial section 3.3 (Getting started with Avizo and 3D images) for learning more about *Ortho Views*.

- Load the image stack file chocolate-bar.am located in subdirectory *data / tutorials /*.

- Duplicate the `chocolate-bar.am` data set in the project View.
- Attach the module *Ortho Views* to `chocolate-bar.am`. The graphics display is then split into four viewers.
- Attach a *Slice overlay* module to the *Ortho Views*: right-click on the yellow *Ortho Views* icon in the Project View and select the entry *Slice*.
- Then connect the *Data* input port of *Slice* module to `chocolate-bar2.am`
- You can change the *Colormap* port to select a semi-transparent colormap such as `volrenGreen` or `physicsVolRend` and adjust the range. Make also sure to set *Slice*'s *transparency* port to "Alpha" (i.e., colormap opacity use it).
- You can use the *Transform Editor* to move `chocolate-bar2.am`.
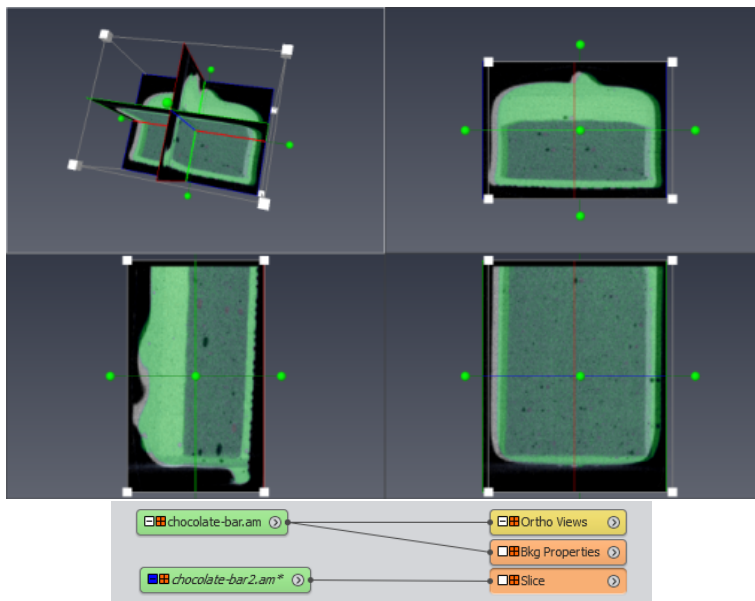
Figure 6.18: Ortho Views display

- You can hide the *Transform Editor* manipulator in some views, while keeping the data visible: toggle the visibility of the data object off - this will make invisible both the slice and the manipulator, then toggle the slice visibility on.
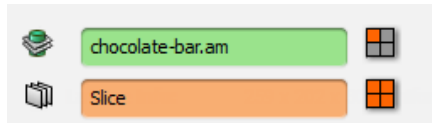
Figure 6.19: Visibility toggles for data and display modules

6.2.3 Mapping a 3D volume overlaid on a surface

The *Surface View* module, used to display surface data, can also display a surface "immersed" in a volume data set (3D scalar field), mapping the volume values as colors over the surface.

- Load the image stack file `chocolate-bar.am` located in subdirectory *data / tutorials /*.
- Load the supplied `chocolate-bar.simplified` surface data file from the *data / tutorials* directory.
- Attach a *Surface View* module to `chocolate-bar.simplified` data set (i.e., choose *Surface View* entry from the popup menu of the data set)
- Connect *Surface View*'s *Colorfield* input port to `chocolate-bar.am` data object. The surface is now colored according to `chocolate-bar.am` data and *colormap* port.
- Choose, for instance, the colormap `physics.icol`. Then right-click on the colorbar and choose *Adjust range* to `chocolate-bar.am`
- Use the *Transform Editor* to move `chocolate-bar.am` relative to the surface.
- Change the colorfield mapping type. With "Per-vertex" mode, the colors are probed at the vertices and interpolated inside the surface triangles, leading to a fast but less precise rendering. Colorfield mapping precision is then limited by vertices density of the surface. The "Per-voxel" mode accurately maps the data texture onto the surface at the expense of memory consumption and lower performance. It may be useful then to simplify the surface by using the *Surface Simplification Editor*.
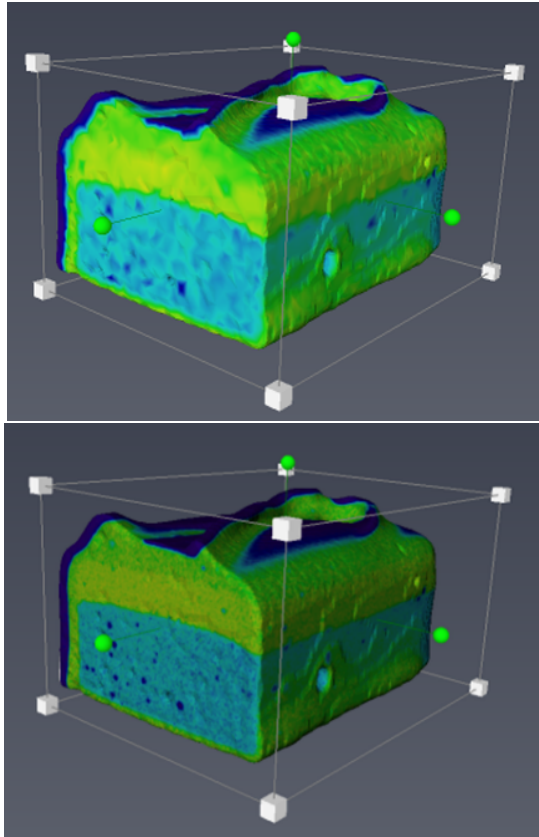
Figure 6.20: Surface View colorfield mapping: per-vertex vs per-voxel

6.2.4 Side-by-side viewers, synchronized views and objects

It is often convenient to visualize different data sets side by side in a synchronized way.

- Load the files `chocolate-bar.am` from and `chocolate-bar.labels.am` from directory *data / tutorials*. The data set `chocolate-bar.labels.am` contains labeling of different regions, obtained by a segmentation of the grayscale image.
- Attach an *Ortho Slice* module to `chocolate-bar.am` and to `chocolate-bar.labels.am`.

Figure 6.21: Ortho Slice modules connected to the data

- Right-click on the *Colormap* port of module *Ortho Slice 2* attached to `chocolate-bar.labels.am` to select colormap `labels.am`.
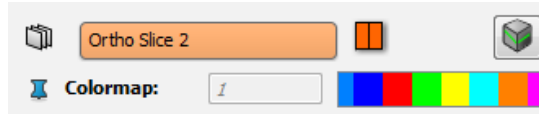
Figure 6.22: View of the Colormap port with the labels.am connected

For now only one *Ortho Slice* is visible since the data sets overlap.

- In the viewer toolbar, toggle on the Two-Viewers (vertical) button, and make sure that the viewer button "Link Object Visibility" is off.
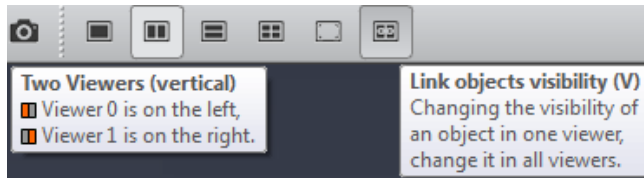
Figure 6.23: View of the Viewer toolbar

- Toggle the visibility of *Ortho Slices* to make `chocolate-bar.am` visible, for instance, only in the left viewer and `chocolate-bar.labels.am` visible only in the right viewer.
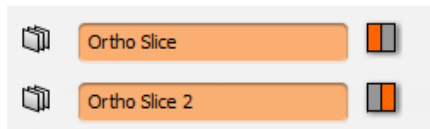
Figure 6.24: Visibility of the two Ortho Slice modules

- Right-click in the left viewer to open its popup menu then select "Link camera to...", then click inside the right viewer to select it.
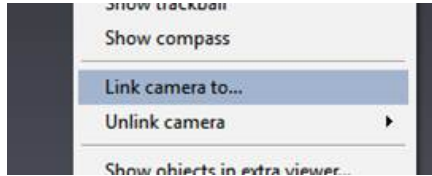
Figure 6.25: Link Camera to... option

- Select the *Ortho Slice* module. In the in *Properties Area* activate the *Connection Editor* (as shown in picture below). Then, click on the chain-link icon that appeared beside the *Slice Number* port of the *Ortho Slice* module, and drag onto the *Ortho Slice 2* module in the Project View. The *Slice Number* ports are now interconnected: changing one will change the other instantaneously.
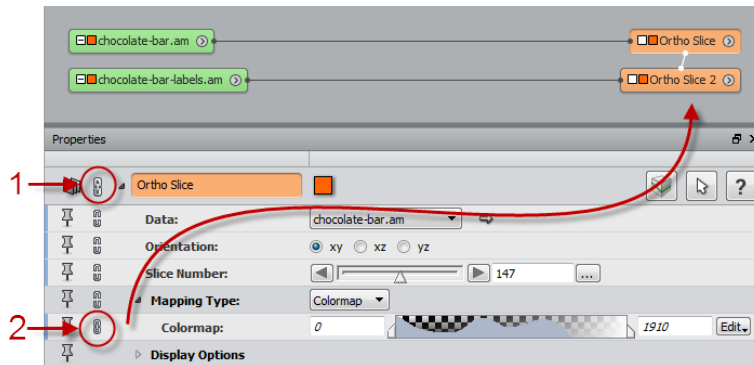
Figure 6.26: Interconnecting ports: (1) activate Connection Editor, (2) drag chain-link from one port to a module or port to be connected

You can then at any time Unlink the viewers in the viewer's popup menu (right-click in viewer). You can also disconnect the ports by right-clicking on the link-chain icon of one of the interconnected ports and selecting "disconnect" in the popup menu. This technique can be useful to compare a data set before and after processing, as shown in the following example:

- Load `chocolate-bar.am`, attach to `chocolate-bar.am` an image filter module *Gaussian Filter*. In the *Properties Area*, set the *Gaussian Filter* to *3D* interpretation, and *Kernel Type* to *Standard*, then press *Apply*.

- Attach an *Ortho Slice* to the result data `chocolate-bar.filtered`, then proceed as above to set up linked viewers.
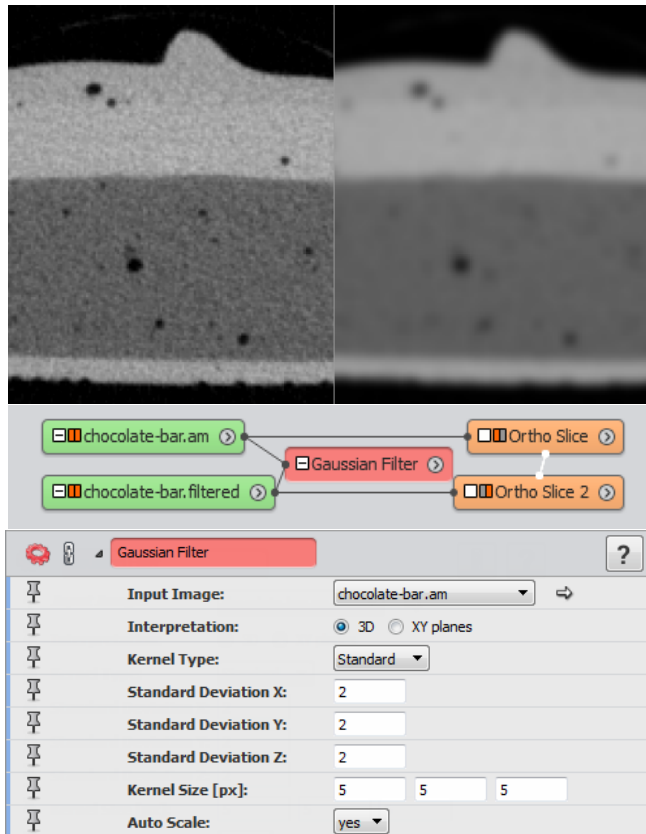
Figure 6.27: Comparing before and after processing

Note: See also modules *Image Processing / Filter Sandbox* and *Slice* to display on-the-fly effects of image filters.

6.2.5 More about Data Fusion

Avizo offers many ways to compare or merge surface or scalar field data. Here are some further hints.

- You can superimpose surfaces using transparency. Here are some useful settings to tradeoff quality and performance. Best results can be achieved with the default options *fancy alpha*

and *sorting* in the *Draw Style* port of *Surface View*. You may want, however, to deactivate one of these options for better performance. You can also change the transparency mode in the main menu *View / Transparency* for different quality/performance tradeoffs. The Sorted Layers options - supported by modern graphics hardware - may give best results to prevent artifacts with complex transparent objects. See *Surface View* and *View Menu* for more details.

- For comparing surfaces, see also *Measure And Analyze / Surface Distance*, *Measure And Analyze / Shortest Edge Distance*, *Compute / Interpolate*, *Compute / Vertex Difference*, *Compute / Arithmetic* (for comparing data attached to surface, or mapping distance map image on surface).

- Landmarks and warping can be used for comparing data. See the related *tutorial*.

- You can superimpose image data using *Color Wash* with *Ortho Slice*, *Ortho Views*, *Height Map Slice* (height field + color field), *Volume Rendering*, *Isosurface*, etc.

- For comparing images, see also the modules *Compare Image*, *Correlation Histogram*, *Arithmetic*, *Subtract Image*, *AND NOT Image*, *Blend With Image*.

- Here is an example for computing difference between images using the *Arithmetic*:

    - Connect the *Arithmetic* module (Compute submenu) to one data set (*Input A*). Connect the second data set as *Input B*.
    - Type "a-b" or "abs(a-b)" in *Expr* field. Press *Apply*. The result as same dimension as *Input A*.

- The powerful module *Compute / Arithmetic* can attach to surface, grid, or image, and can be used to interpolate and map values from one data set to another or to a regular grid.

- The *Merge* module is available for blending images that can be arbitrarily overlapping.

- *Measure / Correlation Histogram* can be used for creating a label image from correlated regions in two images sets, typically after registration.

- *Multi-Channel Field* can group multiple grayscale images of same size for convenient display with *Ortho Slice* or *Volume Rendering*. Multi-channel objects are created automatically when reading some file formats containing multi-channel information. Alternatively, channels can be manually attached to a multi-channel object created via the *Project >Create Object...* menu (category *Images And Fields*).

6.3 Registration with landmarks, warping surfaces and image

Landmark-based registration computes a geometric transformation between two corresponding point sets. A rigid or affine transformation can be computed from these landmarks for aligning the model to the reference. Landmark tools also allow for computation of warping deformations of 3D surface or images to best fit the corresponding landmarks.

Registration with landmarks is most often used as an interactive alignment method since landmarks can be edited interactively in Avizo. However, landmarks can also be imported or produced by other means, such as feature extraction from images after segmentation. This can be used for automatic

registration.

This section has the following parts:

- *Creating landmark sets*
- *Registration with Rigid Transformation*
- *Warping surfaces*
- *Warping volumes (3D images)*
- *Retrieving and copying registration transformation*

To follow this tutorial, you should know how to load files, how to interact with the 3D viewer, how to use the two-viewer layout and the viewer visibility toggles.

Registration with landmarks in this tutorial applies to 3D images and triangulated surfaces. For aligning 2D images or slices see tutorial in section 6.6 (Alignment of 2D images stacks). See also section 6.1.2 (Applying Transforms) on how to align a volume with an arbitrary plane or Slice defined by picking or point sets.

6.3.1 Creating landmark sets

The data sets we will be working with in this tutorial are the labeled motor surface (`motor.labels.surf`) and the simplified labeled motor surface (`motor.simplified`). All steps can also be applied on images, using, for instance, *Isosurface* as the display module.

Let's open and display two surface data sets.

- Load `motor.part1.simplified.surf` and `motor.part2.simplified.surf` in the *data / registration* directory.
- Attach a *Surface View* module to each surface data.
- You can change the color or draw style of one *Surface View* module to better distinguish the two surfaces. For instance, set the *Surface View*'s *Colors* port to "constant", then use the *Colormap* port *Edit* menu or double click on the colored box to change the color using the *Color* Dialog.
- Open two viewers and display each of the surfaces in one viewer (see tutorial in section 6.2.4 (Side-by-side viewers)).
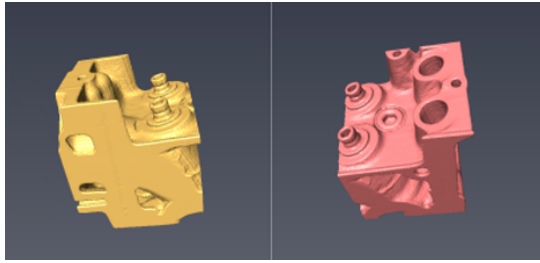
Figure 6.28: Surfaces before registration

Now, let us create a *landmark set* object.

- From the *Project >Create Object...* menu, select *Points And Lines / Landmarks (2 sets)* in order to create an empty landmark object. A new green icon will show up in the Project View. Since we are going to match two objects by means of corresponding landmarks we had to select the landmark object containing two sets of landmarks (Landmarks (2 sets)).
- Select object *Landmarks-2-sets* in the Project View.
- Set *Image Set1* port of *Landmarks-2-sets* to `motor.part1.simplified.surf`, and set *Image Set2* to `motor.part2.simplified.surf`.
- Setting *Image Set* input ports may not be mandatory for the next steps in this tutorial, but this would be required for the *Landmark* object to take into account the transformations set, for instance, if you would move the surfaces with the *Transform Editor*.

Let's now set up for editing landmarks.

- Launch the *Landmark Editor* by clicking on the *Landmark Editor* button in the *Properties Area*.
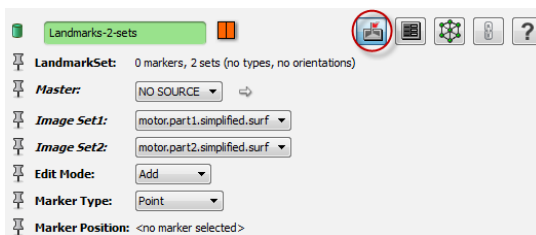
Figure 6.29: Landmark editor

When the editor is started, a *Landmark View* module is automatically created and connected to the *Landmarks* data object. As indicated on the info line, two empty landmark sets are available now. We use the editor to define some markers in both objects.

- Right-click on the *Landmarks* object and add a second *Landmark View* module to it.
- In the first *Landmark View* module properties, set *Point Set* port to *Point Set 1*.
- In the second *Landmark View* module properties, set *Point Set* port to *Point Set 2*.
- Set the viewer toggles of the two *Landmark View* modules so that the first landmark set will be displayed in the left viewer, and the second landmark set will be displayed in the right viewer.
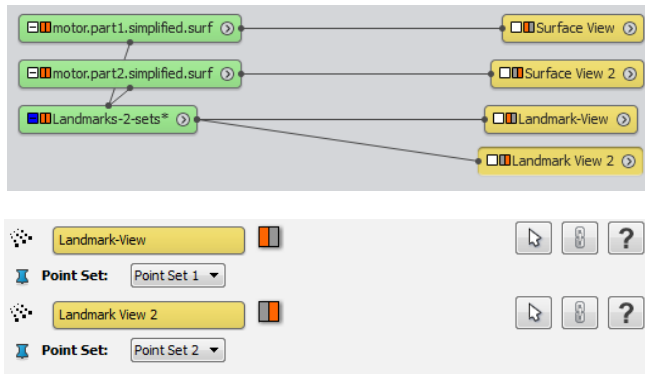
Now, we are ready to define and set corresponding landmarks.

- Make sure that the *Landmarks* object is selected in the Project View, and check in the *Properties Area* that *Edit Mode* is set to *Add*. Note that landmarks can be edited in the viewer windows only if the *Landmarks* object is selected.
- Click on the viewer toolbar arrow button or press the ESC key to turn on picking interaction mode (arrow mouse cursor). Click on a particular point of surface1. This point must be in the common region between surface1 and surface2. Click on the corresponding point in surface2. A pair of points is created. Repeat this step to create several pairs of landmarks (at least 3, 8 pairs are probably sufficient). You may want to change the view of the objects to set landmarks on parts that are initially hidden.
- If you want to change the position of an existing landmark set, select *Edit Mode: Move*. Click on the landmark to be moved, and then just click on the desired position.
- A pair of landmarks can be removed by choosing *Edit Mode: Remove* and by clicking on one of the landmarks (blue or yellow).

Note: When editing landmarks, make sure that no transparent object, such as a *Transform Editor* manipulator, is intercepting the mouse click instead of the expected visible surface. You may not immediately notice that the landmark has been added at the wrong location.
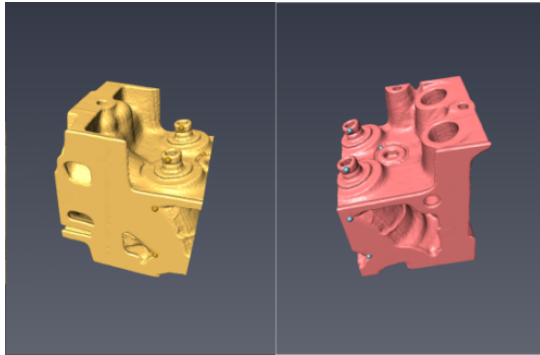
Figure 6.30: Landmark on surfaces

Once landmarks have been created, the next step is to transform the two objects into each other. The *Landmarks* object can calculate a rigid or linear transformation that best fits one landmark set to another (you will see later below how to retrieve it), or you can attach a compute module that can transform a surface or image data object into a new one according to the landmark sets.

6.3.2   Registration with Rigid Transformation

The module *Landmark Surface Warp* can be used to create a transformed copy of an input object, either with a rigid transformation or with warping.

Let's use it for registration of `motor.part2.simplified.surf`, corresponding to the 2nd landmark set, with a rigid transformation towards the 1st landmark set and `motor.part2.simplified.surf`.

- Attach a *Compute / Landmark Surface Warp* module to the *Landmarks-2-sets* object created in previous steps.
- In its *Properties area*, set the *Surface* data port to `motor.part2.simplified.surf`, choose the *Direction 2>1* and *Rigid* Method with *uniform scale*.
- Press *Apply*.
- Attach *Surface View 2* to `motor.part2.simplified.Warped`.
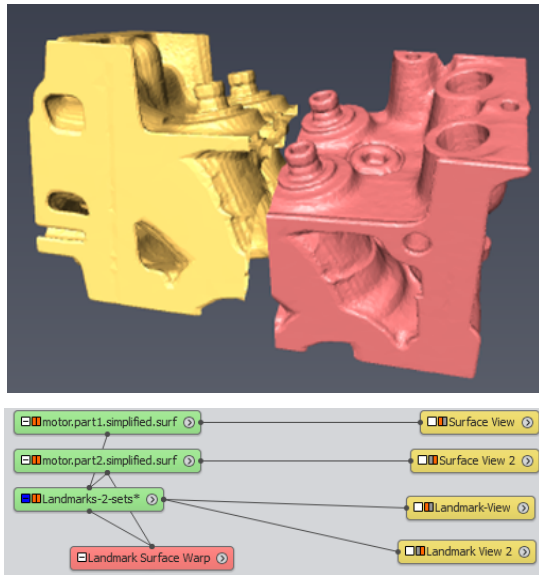
Figure 6.31: Before Landmark Surface Warp
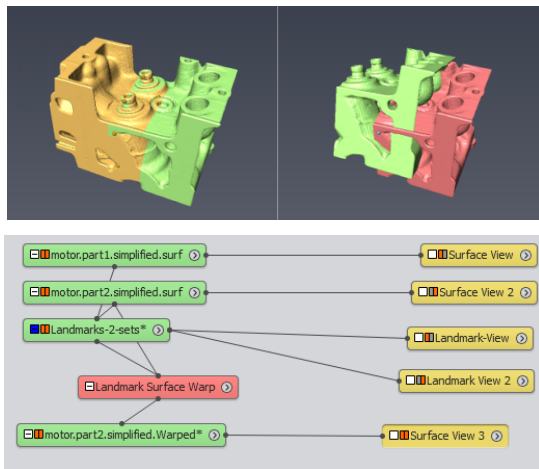
Figure 6.32: After Landmark Surface Warp (Rigid)

The *Landmark Surface Warp* module creates a new surface copy from `motor.simplified.surf`, with coordinates already transformed. You can verify with the *Transform Editor* that no geometric

transformation is set for `motor.part2.simplified.surf`. Therefore, it cannot be used to retrieve the registration transform. The method for doing this, without creating a copy of data, is given in a later section.

In some cases, a rigid transformation (translation and rotation) is not enough, for instance, if there is a difference in scale.

- Try setting *Transformation type* to *rigid+scale*, or *affine* (different scale along x, y, z, possibly with shear). Press the *Apply* button, or toggle on *auto-refresh* for automatic apply upon port change. Depending on the landmarks that were set, you may observe small differences in the result.

The next section is about even more severe, non-linear, transformations.

6.3.3 Warping surfaces

In some cases, it is useful to deform an object for the best fit with the landmarks. For instance, you may want to compare shape variations relative to some fixed anchor points, or warp a 3D image to fit a predefined shape.

If a deformation is required to obtain a better fit to the landmark sets, we can use another transformation method of the *Landmark Surface Warp* module.

- Attach a *Landmark Surface Warp* module to the *Landmarks set* (see Registration via Rigid Transformation section).
- Choose *Bookstein* method, press *Apply*, and visualize the result.
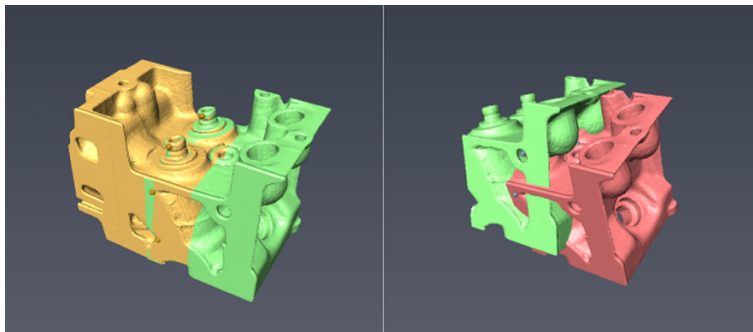
Figure 6.33: After Landmark Surface Warp

Depending on the landmarks you place, the surface may have been slightly deformed to best fit the transformation from landmark set 2 to landmark set 1. You can try different methods and options.

You can then try warping the image data as described in the next section.

6.3.4 Warping volumes (3D images)

You will now transform a volume according to the transformation from landmark set 1 to landmark set 2.

- Remove the *Landmark Surface Warp* module and the previous result `motor.part2.simplified2.Warped`.
- Switch back to single-viewer mode: click on left viewer to select it (a white frame surrounds the viewer area), then press the *Single view* toolbar button.
- Load `motor.am` located in *data / tutorials*.
- Attach a *Bounding Box* module and a *Volume Rendering* module to `motor.am`. You can see it is overlapped by `motor.part1.simplified.surf`.
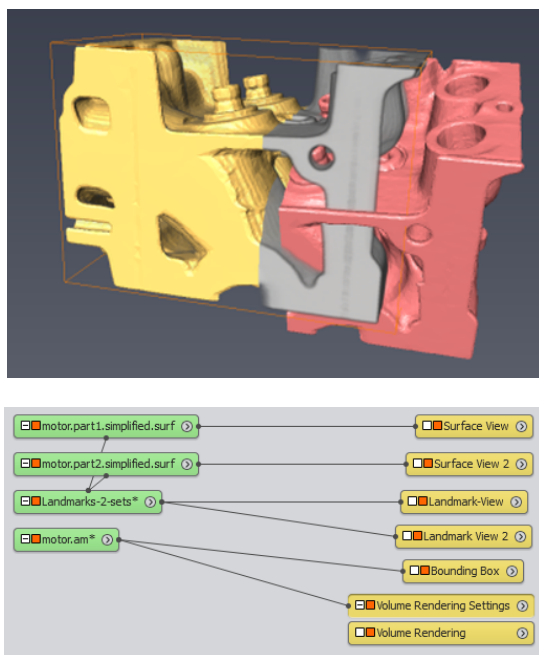
Figure 6.34: Setting up for Landmark Image Warp

- Turn off the viewer visibility toggle for `motor.part1.simplified.surf` and *Landmark View*.
- Attach a *Landmark Image Warp* module to *Landmarks-2-sets* created in previous steps.

- In *Landmark Image Warp* properties, set *Image Data* port to `motor.am`.
- Options should be set as *Direction 1>2* and *Bookstein* method. Press *Apply*.
- Visualize the result by attaching the *Volume Rendering* module to the result `motor.Warped`.
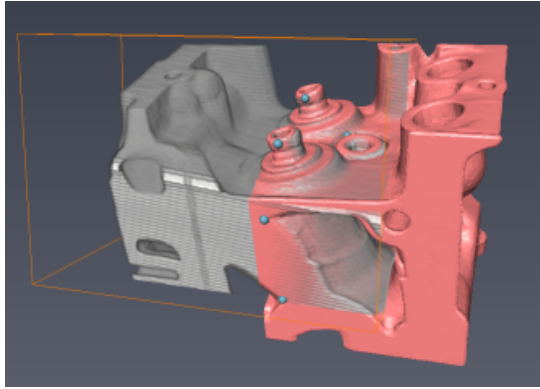
Figure 6.35: After Landmark Image Warp

You can see that volume has been moved and warped following the transformation from landmarks set 1 to landmarks set 2. However, the result volume looks cropped within the extent of the initial input volume `motor.am`. By default, the result of *Landmark Image Warp* is sampled over the same lattice as input. To overcome this, you could extend the input volume using the *Crop Editor*. Or you could attach another image to the *Master* port of *Landmark Image Warp* as a reference for bounding box and resolution, overlapping the expected destination area. An *Arithmetic* module can be used to prepare such reference (main window's menu *Project >Create Object...*, select *Images And Fields / Arithmetic*, then choose regular result type, resolution and volume box location).

- You may then try different methods and options. See *Landmark Image Warp* reference for more detail.

6.3.5 Retrieving and copying registration transformation using Tcl command

It can be useful to set the computed rigid or linear transformation without creating a transformed copy of a data set.

You can retrieve the transformation by using Tcl commands in the console or in a script. In the Avizo console, type:

```
>"Landmarks-2-sets" computeRigidTransform
0.939504 -0.0555508 -0.338004 0 0.0390105 0.997694 -0.0555385 0 0.34031
0.0389928 0.939505 0 -0.0165833 0.212631 0.110712 1
```

This computes a rigid transformation that moves the points of the first set as close as possible onto the points of the second set (the sum of the squared distances between corresponding points is minimized). The result is returned as a 4x4 transformation matrix, which, for example, can be used to transform some other data object using the setTransform command. For instance, type:

```
>eval "motor.part1.simplified.surf" setTransform
["Landmarks-2-sets" computeRigidTransform]
```

By default, `computeRigidTransform` transforms the 1st landmark set into the 2nd one. You can specify which sets and order to use in the command. The command `computeLinearTransform` is available to compute transformations with scaling. See reference *Data Type: Landmarks* for more detail.

6.4   Registration of 3D image data sets

The *Transform Editor* and *Landmarks* tools introduced in previous tutorials can be used for image registration. This section focuses on automatic optimized registration based on image comparison. In this tutorial, you will learn how to register 3D images, wholly or partially overlapping, obtained with the same or different acquisition modality.

This section has the following parts:

- *Getting started with Register Images module*
- *Register Images guidelines*
- *Using the Image Registration Wizard - example with partially overlapping images*
- *More about the Register Images module*

To follow this tutorial, you should know how to load files, interact with the 3D viewer, use modules and *Transform Editor* basics.

6.4.1   Getting started with Register Images module

The *Register Images* module is mainly used to refine that process. It provides automatic registration via optimization of a quality function of the matching between a transformed model image and a reference image.

For success and best efficiency with the automatic registration optimization, the two 3D data volumes should already be positioned fairly close to their optimized alignment. Otherwise, processing could take too much time because of the larger parameter space to be searched, despite the optimized hierarchical algorithm used in *Register Images*. Therefore, the method for image registration generally proceeds in two steps:

1. Pre-alignment with manual or automatic approximate registration.
2. Automatic refined registration.

The following example shows step by step how to quickly register two 3D images of the same object. A later section in this tutorial will show you an even simpler workflow using the convenience script module *Image Registration Wizard*.

- Load the image stack file `chocolate-bar.am` located in subdirectory *data / tutorials*.
- Load the label image `chocolate-bar.labels.am` located in subdirectory *data / tutorials*. This file contains labeling of different regions, obtained by a segmentation process

Let's now set up some convenient visualization for the data. You could simply attach *Ortho Slices* and *Bounding Box* modules to each data object. You may prefer to observe the data simultaneously in all three directions as described in the tutorial on section 6.2.2 (Data Fusion, Ortho Views).

- Attach 3 *Ortho Slices* to `chocolate-bar.am`, with different orientation (xy, xz and yz).
- Attach a *Colorwash* module to each *Ortho Slice*. Set the *Data* input to `chocolate-bar.labels.am`, and the *Fusion Method* to *Weighted Sum*
- Activate the *Transform Editor* for `chocolate-bar.labels.am` and arbitrarily change the position and orientation for this image as shown in figure below. You can then deactivate the *Transform Editor*, and attach a *Bounding Box* module to `chocolate-bar.labels.am`.
- Attach a module *Geometry Transforms / Register Images* to `chocolate-bar.labels.am`, which is, therefore, considered as the model to be transformed.
- Attach the *Reference* input port of *Register Images* to `chocolate-bar.am`.
- Make sure that the *Register Images* module is selected in the Project View in order to display its control ports in the *Properties Area*.
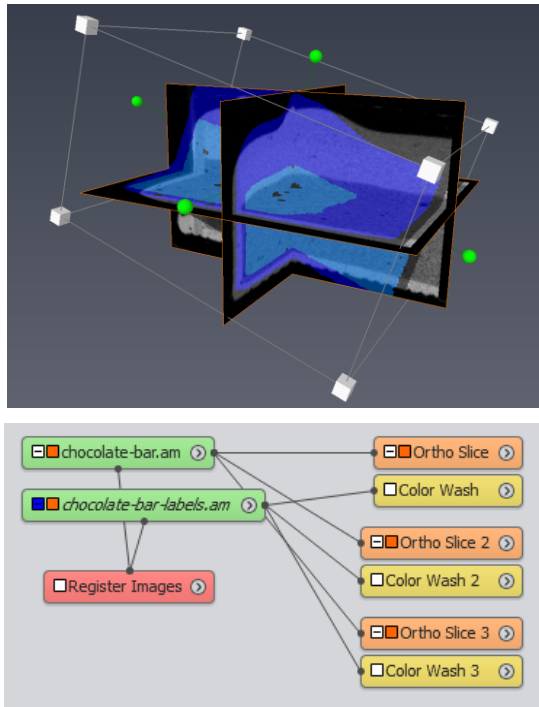
Figure 6.36: Set up Ortho Slices display for Register Images

You are now ready to experiment with registration following the steps below. The *Register Images* ports *Metric*, *Transform*, and *Advanced* options basically control the matching quality measure used, the degrees of freedom for the transform, and other advanced options. They will be detailed later. Let's focus for now on the *Action* port, exposing two pre-alignment methods and the actual optimized registration.
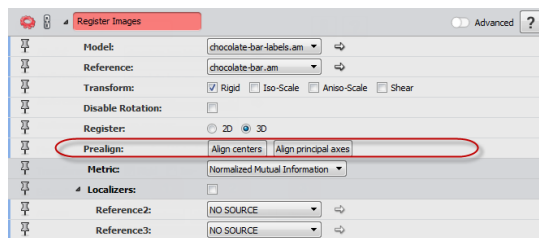
Figure 6.37: Register Images ports

- Press first the *Align Centers* button. The centers of gravity of both data sets are computed considering voxel values as weights. The model's transform is then changed to a translation aligning both centers of gravity. The two images then become close, yet you can notice some remaining shift: this is due to the gravity centers not being located at the same relative position in both data sets.
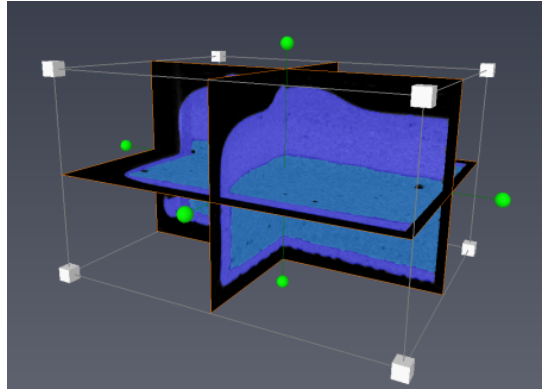
Figure 6.38: After Align centers

- Then press the *Align principal axes* button. The centers of gravity and moment of inertia of both data sets are computed, again taking voxel values as mass density. The corresponding principal axes are used to change the model transform. The best of 24 possible alignments of the principal axes is determined according to the image-matching quality measure (*Metric* port). The two images are then nearly matching again, but there is some orientation drift, in addition to translation shift: the principal axes of both data sets do not align exactly because of the different spatial distributions of voxel intensities.
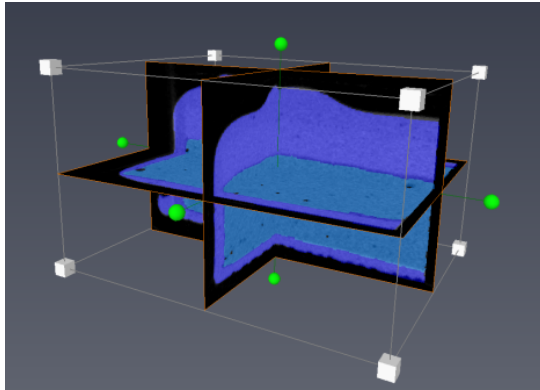
Figure 6.39: After Align principal axes

- Last, press the *Apply* button. This starts the actual iterative registration optimization, which may take a few seconds depending on your hardware. You can then see a best-fit matching of the label image segmentation to the grayscale image.
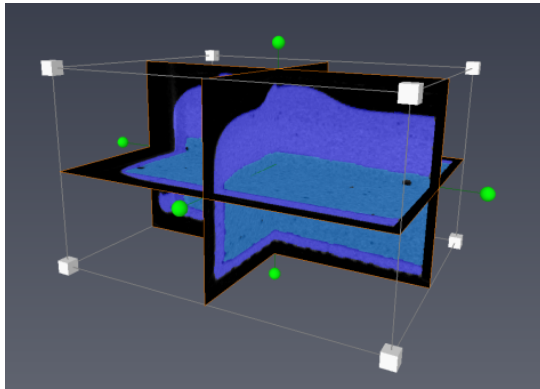
Figure 6.40: After the registration

Depending on the data sets characteristics and location, orientation, scale, you might be able to perform a successful refined registration directly by clicking on the *Apply* button, without pre-alignment. However, the registration may happen to fail to find the optimal position as illustrated below.

- Activate the *Transform Editor* for `chocolate-bar-labels.am`, and then change both its orientation by 90 degrees and its scaling by a factor of 2 (by *Dialog*, or by dragging green spherical knobs and white cubic knobs).
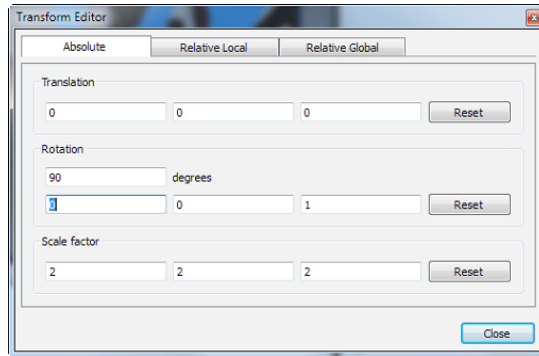
Figure 6.41: Transform rotation and scale values

- Then select the *Register Images* Module. In the *Properties Panel*, check *Iso-* in the *Transform* port, to enable search of scaling in addition to *Rigid* transform.
- Press the *Apply* button. The registration clearly fails (with extended parameters left to default values). The position was not close enough in order to guarantee the expected optimal solution. The search stopped as it could not find a way to improve further.
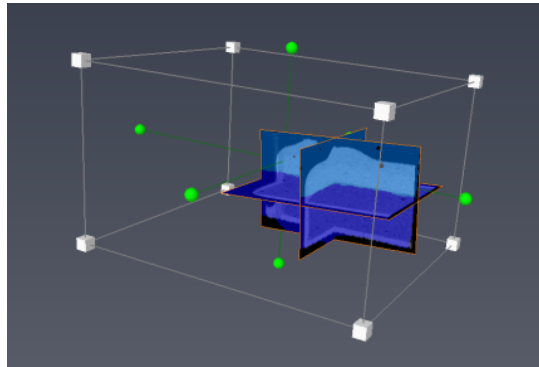
Figure 6.42: Registration failed

- Press the *Align principal axes* button, then the *Apply* button. The registration succeeds.

6.4.2   Register Images guidelines

Before using the *Apply* button, it is often necessary to first move the data sets closer together by some other means, pre-aligning, for instance, by using the *Transform Editor*, by setting an approxi-

mate transform with parameters known in advance, or by using *Landmarks* as described in a previous tutorial.

The easiest way for fully automatic registration, applicable in many cases, is simply to pre-align the data sets using *Align Centers* and/or *Align principal axes* before applying the registration. This may not always work, though, depending on the data sets, which can then require special handling.

1. When differences between the model and reference data set are such that the voxels moment of inertia can't match enough anymore. E.g., some added components, particles or parts changed the model's principal axis significantly.

2. When the model and the reference have some symmetry in voxel intensity distributions that makes the moment of inertia ambiguous. E.g., a cylindrical core sample without significant mass asymmetry.

Here are further guidelines for using the *Register Images* module. More details are given in a later section of this tutorial.

1. Make sure your data sets have identifiable references for pre-alignment, or keep track of information about the approximate location of the model relative to the reference. In some cases, especially if you want fully automated registration, you may need to consider adding physical markers, radio-opaque paint, or fiduciaries to your samples before acquiring images.

2. Use as few degrees of freedom as possible, if only to shorten computation. That is, prefer to set the *Transform* port to *Rigid* rather than *Rigid+Iso*, etc. You can also restrict the registration search to a 2D plane if appropriate (see below advanced options). Whenever possible, set the correct voxel sizes for reference and models (see *Crop Editor*), then set the *Transform* port to "Rigid" instead of searching also for an isotropic scaling.

3. You can select a range of voxel intensities to be considered in registration (see advanced options in a later section). *Register Images* provides robust image matching metrics with respect to different modalities and voxel intensity ranges. However, it can be very useful to ignore the high and/or low intensity voxels of data set components that would otherwise affect the registration.

4. You may need to tune metrics and advanced parameters, depending, for instance, on the data set modality, voxel intensity distribution, dimensions, and aspect ratio. More on this available in a later section.

5. Use reduced data when necessary. *Register Images* uses an efficient hierarchical algorithm. However, in some cases, you can accelerate processing by subsampling the input data without losing significant precision. Also, you can reserve registration for the most relevant subset or derivative of your data set, e.g., cropped, resampled, filtered, masked or segmented, instead of processing the full original data. You can then copy the transformation from such registered "proxy" model to the original data. As a special example, the next section shows how to register

data sets that are partly overlapping.

6.4.3 Using the Image Registration Wizard - example with partially overlapping images

It is sometimes necessary to assemble 3D images taken separately. Unless a careful acquisition recorded the accurate image locations, you may need to take advantage of image overlapping to register precisely the images. The overlapping regions must still contain enough information for successful registration. The *Register Images* module gives most flexibility for addressing this. Here is the method outlined:

1. Extract overlapping subvolumes from the volumes to assemble
2. Pre-align using approximate known locations, or aligning centers, or aligning principal axes
3. Optimize registration
4. Copy the resulting transformation to initial full model
5. Merge the resulting images

For convenience, you can use the *Image Registration Wizard* that helps to register 3D volumes. It allows you to select the image subvolumes to be registered and manages for you the registration process. (Advanced users familiar with scripting may look at the *Image Registration Wizard* as an example of a workflow-driven script module.)

- Load the image stack files `chocolate-bar.part1-reference.am` and `chocolate-bar.part2-model.am` located in subdirectory *data / registration/*.
- Attach a *Display / Isosurface* module to each of the loaded data sets. Change the color of the model *Isosurface* to distinguish it.
- Attach *Geometry Transforms / Image Registration Wizard* to `chocolate-bar.part2-model.am`.
- In *Image Registration Wizard* properties, set the *Reference* to `chocolate-bar.part1-reference.am`.
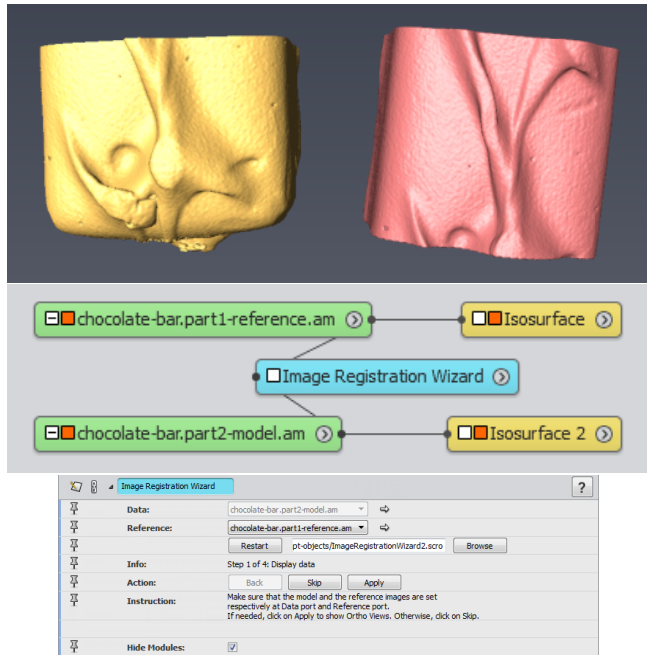
Figure 6.43: Setting up Image Registration Wizard

- Since you have already set up the data display, you can press the *Skip* button. Otherwise, if you press *Apply*, the wizard will set up *Ortho Views* for you.

In the next two steps, you will specify overlapping subvolumes from the model and the reference volumes, with optional subsampling. The subvolumes should correspond approximately to the common region between the model and the reference. Subsampling may be useful for large data.

Note that if you leave the default options, the full volumes will be used directly. It is then possible to use the *Image Registration Wizard* to register fully overlapping volumes.

You can also register images that are not fully loaded in memory, stored on disk using the LDA file format. When attached to such an "out-of-core" data set, the subsampling rate is set to 4 by default.

- Select approximately the overlapping region of `chocolate-bar.part2-model.am`, either using the blue-corner tab box in 3D, or the *Box min* and *size* ports in the wizard's properties.
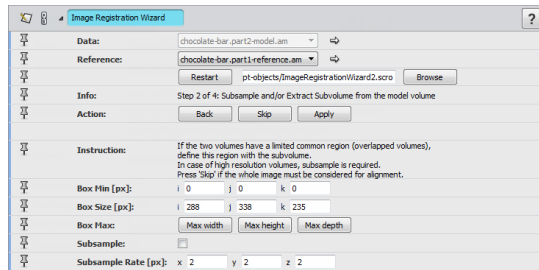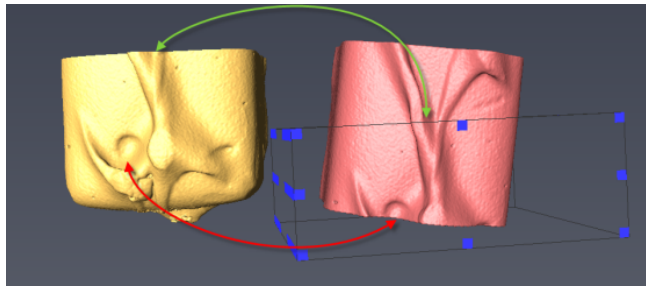
Figure 6.44: Defining model subvolume

To better visualize the subvolume location with axis-aligned display, you may find it convenient to use the viewer's orthographic projection mode.

- For instance, in the viewer toolbar, click on the *Perspective / Orthographic* button, then click on the YZ button (or press X key), then Ctrl-Shift+click on the *Rotate* button for 90 degrees rotation. Click again the *Perspective/Orthographic* button when you want to go back to perspective display. Note: The *Ortho Views* module is another useful display alternative.
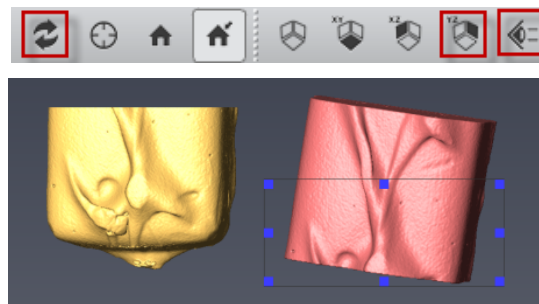
Figure 6.45: Setting up orthographic view

- Press *Apply* to proceed to the next step.
- Select approximately the overlapping region of `chocolate-bar.part1-reference.am`.
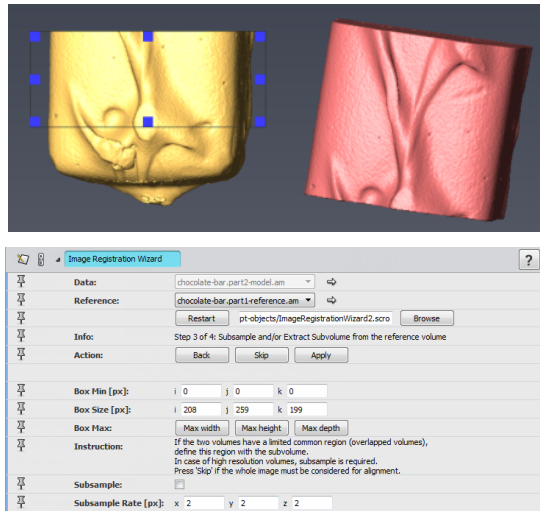
Figure 6.46: Defining reference subvolume

- Press *Apply* to proceed to the next step.

You are now ready to proceed to registration. Some of the *Register Images* options are exposed (further explanation of the *Register Images* options is provided in the next section).

1. Image comparison measure
2. Intensity range to be considered for registration (available when using *Mutual Information* metric)
3. Degree of freedom: rotation and translation only; uniform scaling; variable scaling in x, y, z; shear
4. Type of pre-alignment done before registration optimization. "None" assumes that the model is already approximately aligned with the reference. For instance, you might set the model in the correct initial position by using the Transform Editor.
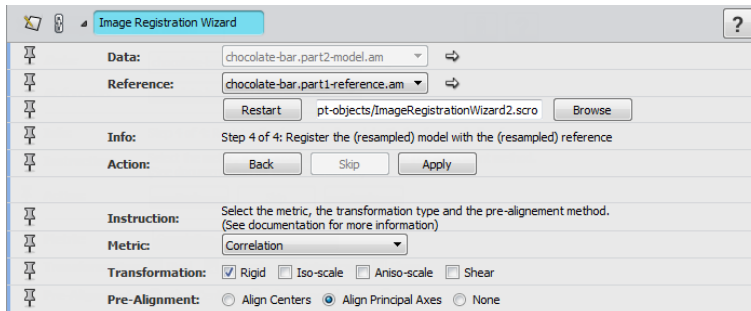
Figure 6.47: Image Registration Wizard - registration settings

Since the overlapping regions we have selected are suitable for principal axes pre-alignment, you can simply leave the default options.

- Press *Apply* to finally proceed to registration.

You should obtain a good alignment of `chocolate-bar.part1-reference.am` and `chocolate-bar.part2-model.am`.
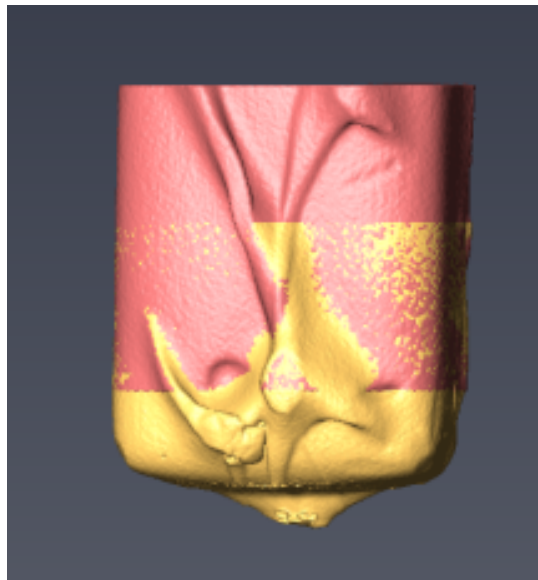
Figure 6.48: Image Registration Wizard - final result

- You can go back to previous steps with the wizard to experiment different options.
- Once you obtain a satisfactory result, you can remove the *Image Registration Wizard*. The ancillary modules and temporary data objects will be removed at the same time.

As a final additional step, you can now *merge* the registered volumes into a single data set.

- Attach a *Compute / Volume Operations / Merge* module to `chocolate-bar.part1-reference.am`, and then attach `chocolate-bar.part2-model.am` as Lattice1 additional input.
- Press the *Apply* button.
- You can attach a *Volume Rendering* module to the result `Merged-chocolate-bar.part1-reference.am`. Then toggle off the visibility for *Isosurface* modules attached to model and reference parts.
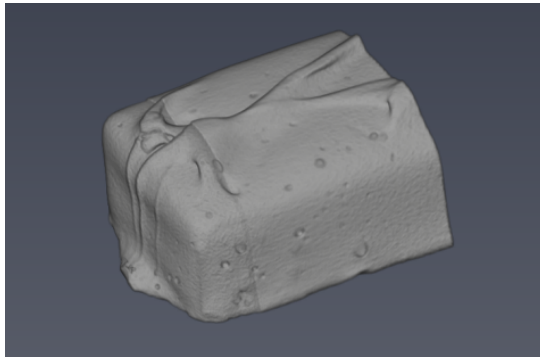
Figure 6.49: Merge result

6.4.4   More about the Register Images module

This section gives more details about the key options and advanced parameters that may be useful to set when using the *Register Images* module:

1. Degrees of freedom
2. Image comparison metrics
3. Optimization control

Let's first outline how the iterative optimization of image registration works. The algorithm considers a model data set to be transformed, and a reference data set to which the model is registered.

1. The first step is to resample internally both data sets, using an adjustable resampling rate.

2. A measure of similarity between images is calculated depending on the selected metric.

3. The model transformation is modified with a variable incremental step, depending on the degrees of freedom and the optimization strategy trying to maximize similarity.

4. The process is repeated then hierarchically to higher resolutions.

Degrees of freedom

The number of transformation parameters can be selected. Four different transformations are available:

- Rigid transformation : translation and rotation
- Isotropic scaling
- Anisotropic scaling
- Shearing
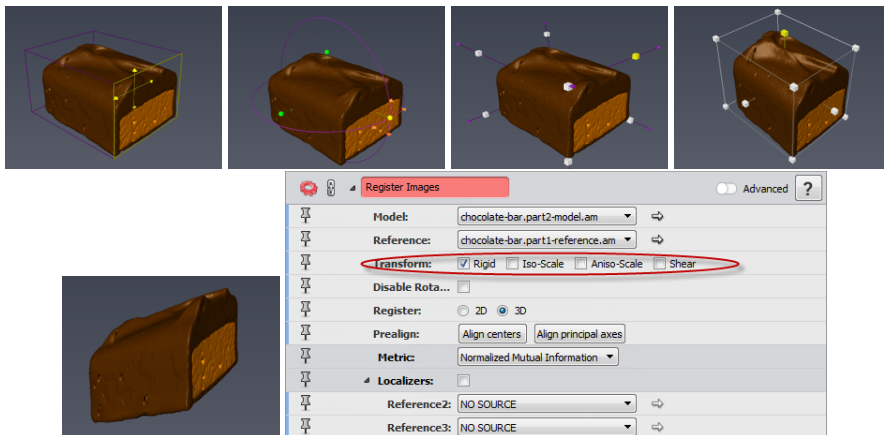
Figure 6.50: Degrees of freedom: rigid (translation+rotation), uniform scaling, anisotropic scaling, shearing

2D constrained registration

In addition, you can restrict the search to transformations within the same plane. For this, switch to advanced mode and toggle the *Register 2D* mode.
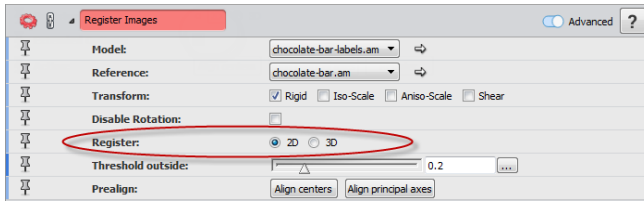
Figure 6.51: Registration restricted to 2D

Note that if the model or reference input is a 2D image (1 slice only in z direction), the *Register* port is automatically set to 2D, which constrains the search space to one plane.

You can work around this by resampling the 2D slice on a slab with at least 2 voxels depth as dimension, keeping the bounding box size to 1 voxel. Then a 3D degree of rotational freedom can be applied. This can be done with the *Crop Editor*, as follows: Note the bounding box extent in z direction, set *Image crop max z* index to 1 or more, leaving the *Add* mode: replicate option enabled. Make sure that the bounding box extent is set to the same extent as before (i.e., the voxel size in z direction).
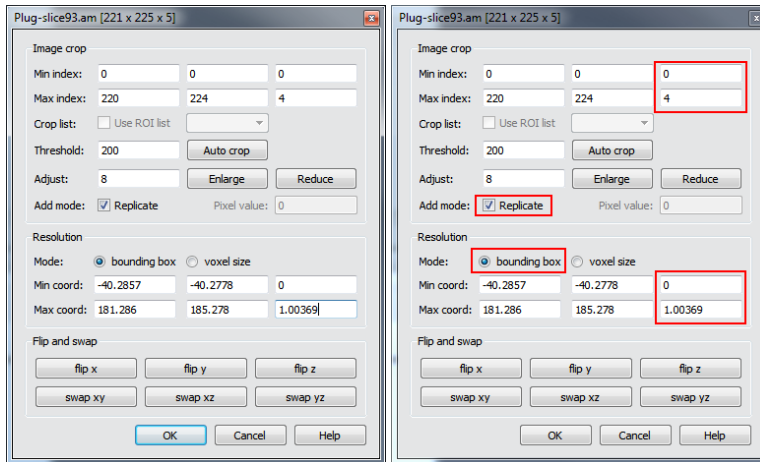
Figure 6.52: From 2D image to slab

Similarity metrics

The *Metric* port specifies the similarity measure between the two data sets.
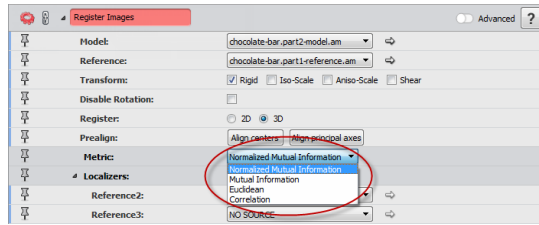
Figure 6.53: Register Images metrics

- *(Normalized) Mutual Information* uses model and reference histograms to compute an energy/entropy. The goal is to minimize the entropy, so as to maximize the mutual information between the two data sets. This is the most generic and robust metric. It is recommended (especially the normalized version) when images come from different modalities (e.g., CT/MRI, ECT/XCT). Also note that the histogram range can then be specified to define the relevant voxel values to be considered for registration. You can use this to ignore high or low intensity voxels.
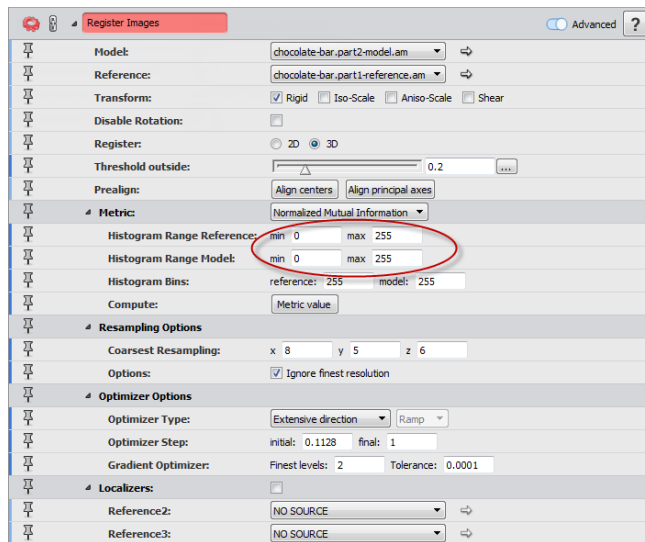
Figure 6.54: Register Images histogram range

- *Euclidean Distance*: computes the distance between the gray values of the two images. It is well suited for data sets with similar histograms (or similar signal response). For instance, images acquired using the same modality and intensity calibrations, or overlapping parts of the same image

Registration of 3D image data sets   308

- *Correlation*: computes a correlation value between the model and the reference, suited for data sets whose histograms (or signal response) are similar via a linear transformation, i.e., images acquired using the same modality but possibly not the same Intensity Range Partitioning.
- *Label Difference*, for labeled images, measures difference between two connected labels.

Optimizer control

You may need to adjust optimization parameters in order to balance improved accuracy, search robustness, and processing time.
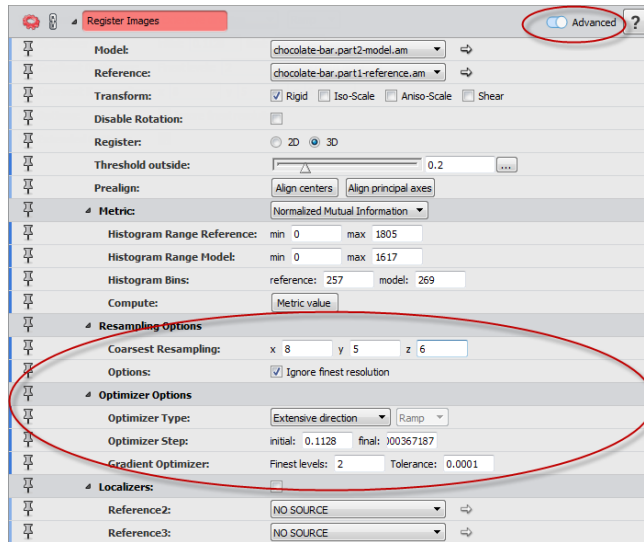
Figure 6.55: Register Images optimizer options

- *Optimizer Step*: set the initial and final value for the step width applied in the optimization. The greater the initial value, the larger will be the transformation tested. The smaller the final value, the finer will be the transformation tested. Default values are based on the data bounding box and voxel size (see reference help for detail). If the pre-alignment is precise enough, you can reduce the initial value closer to the voxel size. When the bounding box aspect ratio is rather flat and images tend to flee away during registration, you may need to reduce the initial step in order to keep the images overlapping longer.
- *Coarsest Resampling*: set the coarsest level of resolution. Data sets will be resampled many times until the coarsest resampling rate is reached. For the coarsest search, these values can be increased.
- *Ignore finest resolution*: if this box is checked (default), the highest level of resolution will be left out. Often, full resolution is not needed to register images, which can dramatically save processing time.

Two different optimization strategies are applied, depending on the resampling level. At the finest level, the *Quasi Newton* optimizer is applied. In the other levels, the optimizer can be chosen between:

- The *Extensive Direction* or *Best Neighbor*, well suited for coarse resolution levels;
- *Quasi Newton* or *Line Search*, suited for finest resolution, or when information overlap is reduced due, for instance, to flat aspect ratio;
- *Conjugated Gradient*.
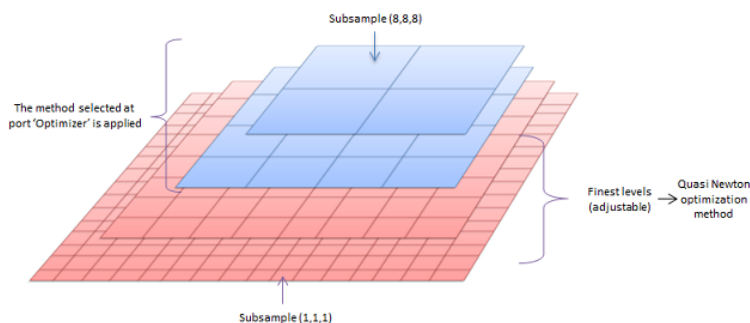
Figure 6.56: Resampling with a coarsest resampling rate of (8, 8, 8)

Getting similarity measure with Tcl in console or script

At the end of a registration, a distance factor can be retrieved by Tcl command in the console or in a script, providing a similarity measure:

```
>"Register Images" getLastImageDistance
0.300090
```

This allows creation of custom search scripts for specific purposes.

6.5   Registration of 2D image and 3D image data sets

In some case, it is necessary to combine information from a 2D acquisition - e.g., SEM, QEMSCAN, thin-section light micrograph - with a 3D volume such as a micro-tomography image.

It is possible to register a 2D image with respect to 3D image. However, the limited amount of information in particular in the slice thickness direction can make it difficult and can require more processing.

In this section, you will see an example of a strategy for solving such problem. For more details, background information, and further guidelines, please refer to the previous tutorial section 6.4 on 3D image registration.

6.5.1 Guidelines

The method for 2D-to-3D image registration proceeds in two stages, as indicated in a previous section:

1. Pre-alignment with manual or automatic approximate registration. It is preferable for the approximate location and orientation of the 2D slice to be known in advance. Landmark-based registration from some identifiable or automatically segmented markers can also be an option. Last, a systematic search can be employed as shown in the next section, possibly using subsampled data.

2. Refined registration. The registration settings may need to be adjusted to take into account the limited information overlap within the 2D slice.

6.5.2 Pre-alignment: Searching for a 2D slice (needle) in a 3D volume (haystack)

Let's start with an example of searching for an extracted slice within an image stack. A possible strategy is to register the 2D slice at different locations within the 3D stack, and use the best similarity score to retrieve the best slice location. We will now use a script module that manages for us the search process.

- Open the following files in the in *data / core* directory: the 3D image file `Plug.am`, the 2D image file `Plug-slice93.am`, and the script file `SearchSlice.scro`.
- For convenient display, you may split the viewer using the *Two viewer* toolbar button. Then attach a *Bounding Box* module to the 3D image, and an *Ortho Slice* module to each image. Set the visibility of the 3D image to the left viewer, and display the 2D image in the right viewer.

- Select the module *SearchSlice.scro*.
- In the *Properties Area*, set the *Data* port to `Plug.am` and set the *Slice* port to `Plug-slice93.am`. You can set the *Metric* to *Correlation* in this case since the modality is identical.
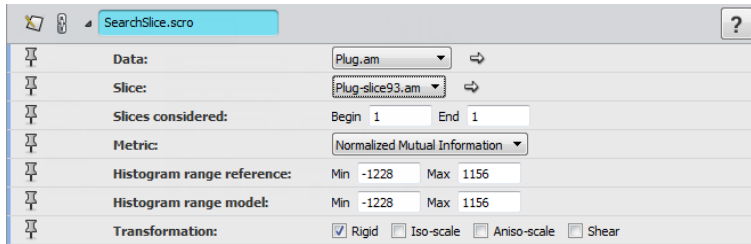
Figure 6.57: Search script - options

- For quicker trial, you can restrict the slices considered for search to 90-95.
- Press the *Apply* button at the bottom of the *Properties Area*.

It may take a few seconds for the script to computes image similarity for best registration at different z-position in the 3D image and store the results in a *Spreadsheet* object. For each selected slice position in z, different initial orientations can be used for registration.

In this example, the script extract all slices between the search window defined at *Slices considered* port. At each extraction step, the extracted slice, named `Plug.view`, is registered with `Plug-slice93.am` using the *Register Images* module. For each slice, four initial positions are considered, consisting in 90 degrees rotation about the z-axis. Four values are obtained, which correspond to the distances between the extracted slice and the reference slice, measured with the transformation computed at the end of the registration and the metric chosen. The minimal and the maximal distances, or similarity values, are saved in the Spreadsheet for each extracted slices.

- Select the *Spreadsheet* object.
- In the *Properties Area*, click on the *Spreadsheet: Show* button.

You can check the best score, corresponding to the higher maximal distance value, in the spreadsheet. It corresponds to the slice with the best similarity value. Note that the slices are numbered starting from one by this script, unlike the *Ortho Slice* module slice, whose numbers start from zero.

Registration of 2D image and 3D image data sets   312

Figure 6.58: Search script - result spreadsheet

Advanced users familiar with scripting may extend this script for specific purposes. See chapter 9.4 (Scripting Guide) for guidance on how to extend scripts.

6.5.3 Refined alignment

The approximate registration is done: you know approximately the slice position. Now, an accurate registration can be computed in order to compute the right transformation in the XY plane, and possibly adjust for some tilt. The steps for this example are the following:

- Consider only `Plug.am` and `Plug-slice93.am`.
- Click on `Plug-slice93.am`. and click on the *Transform Editor*.
- Translate the image to the 94th slice position (numbered 93) on the core sample. You can use the *Transform Editor Dialog:* change the z-value of translation to 94, since the voxel size in z is approximately 1.
- Resample the 2D slice as a slab as indicated in tutorial section 6.4.4 (Registration of 3D images, paragraph '2D constrained registration').

Plug-slice93.am [221 x 225 x 5]

Image crop

Min index: 0 0 0
Max index: 220 224 4
Crop list: ☐ Use ROI list
Threshold: -986 Auto crop
Adjust: 8 Enlarge Reduce
Add mode: ☑ Replicate Pixel value: -1228

Resolution

Mode: ◉ bounding box ○ voxel size
Min coord [mm]: -40.2857 -40.2778 0
Max coord [mm]: 181.286 185.278 1.00369

Flip and swap

flip x | flip y | flip z
swap xy | swap xz | swap yz

OK Cancel Help
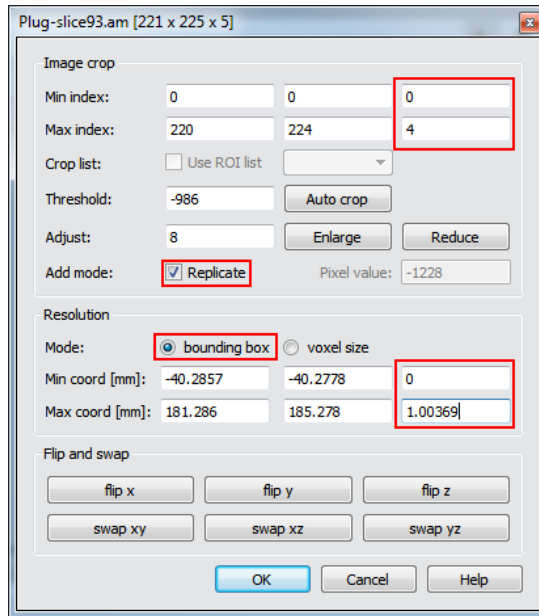
Figure 6.59: Crop Editor settings

- Attach a *Register Images* module to `Plug-slice93.am`. Set *Reference* port to `Plug.am`. Select an appropriate *metric* (*Correlation* or *Euclidean*), and in extended options, change the initial value of the *optimizer step* to 2 (close to the voxel size) since the image is already fairly well aligned in the z-axis.
- You can set at first *Register mode* to *2D*, which should be sufficient in that case, then press *Apply*.
- If you want to try *3D* mode so that slight image tilt will be considered, you may need to set *Optimizer* to *Quasi Newton* or *Linear Search*, and reduce *search step*.

6.6 Alignment of 2D images stacks

Many microscopy techniques require the specimen to be physically cut or captured into slices and then images are taken from each cross section separately. Often the images will be misaligned relative to each other. The images have to be aligned in order to turn the initial image stack into a correct 3-dimensional model of the specimen.

Beside the general registration module Register Images described in a previous tutorial, Avizo provides the *Align Slices* module for alignment of serial sections. *Align Slices* can be used, for instance, with histological slices, or milled or polished material surface layers, images at different focal planes, captured with cameras, light, confocal or electron microscopes, etc.

Note that we do not address here the specific case of alignment of projection images collected by variable tilt tomography, which is a prerequisite to tomographic reconstruction. While Avizo algorithms could be used for such alignment to some extent, this is beyond the scope of this tutorial.

Align Slices is used for aligning 2D slices of a 3D image stack. The module *Register Images*, which can be also used to register 3D or 2D images, is described in a separate section. Alignment with *Align Slices* module can be manual, automatic, or semi-automatic. The following topics will be discussed:

- *Basic manual alignment*
- *Automatic alignment*
- *Alignment via landmarks*
- *Optimizing the quality function*
- *Resampling the input data*
- *Other alignment options and guidelines*

6.6.1 Basic manual alignment

In this tutorial, we want to align 10 microscopic cross sections of a leaf showing a stomatal pore. The images are located in the *data* directory in the subdirectory *align*. Each slice is stored as a separate JPEG image. The file `leaf.info` defines a 3D image stack consisting of the 10 individual slices. It is a simple ASCII file as described in the stacked slices file format section.

Note that this example data set is a stack of color images (RGBA). See also the section "More about align slices" at the end of this tutorial for issues to be considered when using grayscale images.

- Load the file *data / align / leaf.info*.
- Create an *Align Slices* module by choosing *Geometry Transforms / Align Slices* from the popup menu over the `leaf.info` icon.
- Press the *Edit* button of *Align Slices*.
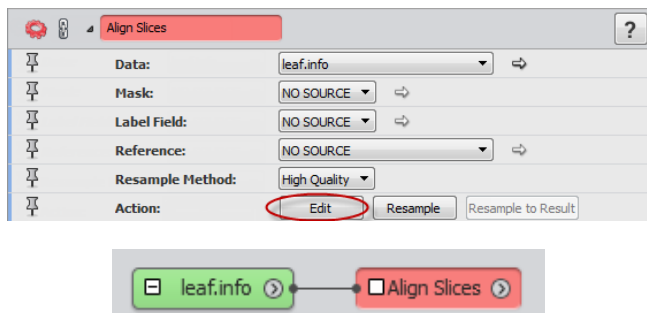
Figure 6.60: Setting up Align Slices

The slice aligner window opens in place of the 3D viewer, allowing you to interactively align the slices of the 3D image stack. To facilitate this task, usually two consecutive slices are displayed simultaneously. One of the two slices is editable, i.e., it can be translated and rotated using the mouse. By default, the upper slice is editable. This is indicated in the tool bar of the align window (the *upper slice* button is selected).
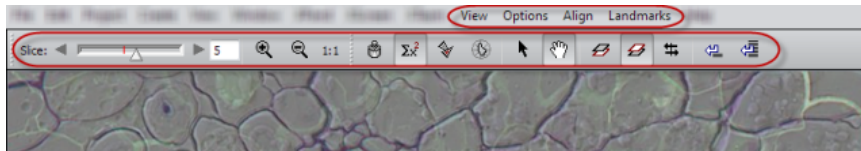
Figure 6.61: Align Slices menu and toolbar

- If necessary, press the *zoom out* button to allow the entire slice to be visible in the viewer.

Figure 6.62: Zoom out button

- Translate the upper slice by moving the mouse with the left mouse button pressed down.
- Rotate the upper slice by moving the mouse with the left mouse button and the Ctrl key pressed down. Alternatively, slices can be rotated using the middle mouse button.
- Make the lower slice editable by selecting the *lower slice* tool button. Translate and rotate the lower slice.
- Hold down key number 1. While this key is hold down, only the lower slice is displayed.
- Hold down key number 2. While this key is hold down, only the upper slice is displayed.
- Pressing key number 1 and 2 also changes the editable slice. Note how the slice tool buttons change their state.

Other pairs of slices can be selected using the slider in the upper left part of the align window. Note that the number displayed in the text field at the right side of the slider always refers to the editable slice. The next, or the previous pair of slices, can also be selected using the space bar or the backspace key, respectively. The cursors keys are used to translate the current slice by one pixel in each direction.

- Browse through all slices using the space bar and the backspace key. Translate and rotate some slices in an arbitrary way.
- Translate all slices at once by moving the mouse with the left mouse button and the Shift key pressed down.
- Rotate all slices simultaneously by moving the mouse with the left mouse button and the Shift and Ctrl key pressed down. Moving all slices simultaneously can be useful in order to move the

region of interest into the center of the image.

6.6.2 Automatic alignment

Besides manual alignment, four automatic alignment options are supported: alignment using a gravity centers and principal axes transformation, automatic optimization of a quality function, edge detection-based alignment, and alignment via best fit of user-defined landmarks. The principal axes method and the edge detection method are only suitable for images showing an object that is clearly separated from the background. The optimization method requires that the images are already roughly aligned.

Depending on acquisition modality and quality, slices can have moderate drift that can be corrected easily automatically. In some cases, one has to correct a few abnormal slices to enable automatic alignment. A general workflow could be outlined in two steps:

1. Pre-alignment with manual or automatic method
2. Automatic alignment optimization

Often such a pre-alignment can be achieved using the landmarks method.

6.6.3 Alignment via landmarks

Alignment via landmarks first requires you to interactively define the positions of the landmarks. This can be done in *landmark edit* mode.

- Activate *landmark edit* mode by pressing the *landmark alignment mode* button.

Figure 6.63: Landmark alignment mode button

In *landmark edit* mode only one slice is displayed instead of two. Two default landmarks are defined in every slice.

- Once you have activated the *landmark edit* mode (arrow mouse cursor), click on one of the default landmarks. The landmark gets selected and is drawn with a red border.
- Click somewhere into the image in order to reposition the selected landmark.
- Click somewhere into the image while no landmark is selected. This causes the next landmark to be selected automatically.
- Click at the same position again in order to reposition the next landmark.

The double click method makes it very easy to define landmark positions. Of course, additional landmarks can be defined as well. Landmarks can also be deleted, but the minimum number of landmarks is two.

- Choose *Add* from the *Landmarks* menu.
- Click anywhere into the image in order to specify the position of the new landmark.
- Select the yellow landmark by clicking on it.
- Choose *Remove* from the *Landmarks* menu in order to delete the selected landmark again.

Two landmarks should be visible now, a red one, and a yellow one. Next, let us move these landmarks to some reasonable positions so that we can perform an alignment.

- Select slice number 0.
- Place the landmarks as shown in the Figure 6.64. Make use of the double-click method.
- In all other slices, place the landmarks at the same positions.

Figure 6.64: Setting landmarks with Align Slices

Once all landmarks have been set, we can align the slices. It is possible to align only the current pair of slices, or to align all slices at once. Note that all alignment actions, as well as landmark movements,

can be undone by pressing Ctrl-Z.

- Switch back to *transform* mode by pressing the *manual alignment* button. Two slices should be displayed again.
- Align the current pair of slices by pressing the *align current slice pair* button.
- Align all slices by pressing the corresponding button.
- Move and rotate the whole object into the center of the image using the mouse with the Shift key hold down.

Figure 6.65: (1) Manual alignment button; (2) Align current slice pair; (3) Align all slices

In most slices the alignment now should be quite good. However, looking at the pairs 3-4 and 4-5 (displayed in the lower left corner of the align window) you'll notice that there is something wrong. In fact, slice number 4 has been accidentally inverted when taking the microscopic images. Fortunately, this error can be compensated for in Avizo.

- Select slice pair 3-4 and make sure that the upper slice, i.e., slice number 4, is editable.
- Invert the upper slice by pressing the *mirror editable slice* button.
- Realign the current pair of slices by pressing the corresponding button.
- Select slice pair 4-5 and realign this pair of slices as well.

Figure 6.66: (1) Mirror editable slice button; (2) Align current slice pair

Alternatively, you could have aligned all slices from scratch by pressing the first button from the right.

6.6.4 Optimizing the least-squares quality function

Once all slices are roughly aligned we can further improve the alignment using the automatic optimization method. The quality of the current alignment is displayed in the status bar of the align window. This is a number between 0 and 100, where 100 indicates a perfect match. The quality function is computed from the squared gray value differences of the two slices. The optimization method tries to maximize the quality function. Since only local maxima are found, it is required that the slices be reasonably well aligned in advance.

- Click on the slice in the viewer. The quality of the alignment is displayed in the status bar at the bottom of the window. Remember the current quality measure.
- Activate the *optimization* mode by pressing the *least-squares alignment mode* button. Remember the current quality measure.
- Align the current pair of slices by pressing the corresponding button. Observe how the quality is improved.

Figure 6.67: (1) Least-squares alignment button; (2) Align current slice pair

Automatic alignment is an iterative process. It may take quite a long time depending on the resolution of the images and of the quality of the pre-alignment. You can interrupt automatic alignment at any time using the *Stop* button.

- Automatically align all slices by pressing the first button from the right.

6.6.5 Resampling the input data into result

If you are satisfied with the alignment, you can resample the input data set in order to create a new aligned 3D image suitable for further visualization or processing. This is done using the *Resample* button of the *Align Slices* module.

- Press the *Resample* button of the *Align Slices* module. The images are resampled using an accurate interpolation method. Note that the *Align Slices*' port *Resample Method* lets you alternatively choose a fast preview mode.
- Before visualizing the result, you could open an extra viewer (menu *View / Layout / Extra Viewer*). For now, simply press the *Close* button of *Align Slices* module in the *Properties Area* to close the slice aligner window and display again the main viewer window. When closing the *Align Slices edit* mode, you can confirm to save the modified transformation information into the data parameter section of the input object (next section shows how this can be used).
- Attach an *Ortho Slice* module to the resulting object leaf.align and verify how the slices are aligned.

By default, the dimensions of the resampled image result are the same as the dimensions of the input image. When *Align Slices editor* is active, you can choose menu *Align / Options* and select *Output* tab, then you can define a different output size, including automatic fit for all slices. See *Align Slices* help for more details.

6.6.6 2D alignment guidelines, more about Align Slices

Completing alignment

Sometimes you may want to improve an alignment later on. In this case, it is a bad idea to align the resampled data set resulting from initial alignment, since this would require a second resampling operation, cumulating interpolation errors. Instead, you could write the transformation data into the original image object and store this object in Avizo format. After reloading the Avizo file, you can attach a new *Align Slices* module and continue with the stored transformations.

- Make sure that the slice aligner window was closed as described in previous steps, so that the slice transformations are recorded in the data parameters of the input object `leaf.info`. Note that you can Choose *Save transformation* from the *Options* menu of the slice aligner at any time while you are modifying slice alignment.
- Delete the *Align Slices* module.
- Save `leaf.info` in Avizo format. For this use menu *File / Save Data As*, then make sure an Avizo native format is selected.
- Reload the saved object `leaf.am`.
- Attach a new *Align Slices* module to `leaf.am` and click the *Edit* button. You can verify that the original alignment is restored.

Using a reference image alignment

In some cases, you might want to reapply the same alignment to a separate image stack with the same dimensions, such as an additional acquisition channel or a segmented image (label image). For instance, you might want to correct the alignment after image segmentation has been performed independently. In order to avoid segmenting the newly resampled image from scratch, you can apply the same transformations to the label image using a reference image.

- Delete any existing *Align Slices* module.
- Load the file *data / align / leaf-unaligned.labels* into Avizo.
- Attach a new *Align Slices* module to the label image.

The guard cells of the stomatal pore are marked in the label image. Segmentation has been performed before the images were aligned. Now we want to apply the same transformation defined for the image data to the labels.

- Connect port *Reference* of *Align Slices* to `leaf.am`. This is done by activating the popup menu over the small rectangular area at the left side of the *Align Slices* icon. Observe how the transformations are applied to the label image.
- Export an aligned label image by pressing the *Resample* button.

Note about color image segmentation: The image volume used in this tutorial is an RGBA color field. You can use *Interactive Thresholding* for color image segmentation. However, other tools such as the

Segmentation Editor only support grayscale images. Therefore, you must convert the color field into a scalar field using *Convert Image Type* or into separate channels using *Channel Works* before you can invoke such segmentation tools for the resampled labels.

Grayscale images

When attaching *Align Slices* module to grayscale image, a *Intensity Range* port is available to select the range of intensity used for display and for computing alignment. It is important to make sure that the selected range contains the intensity values you want to be considered in alignment. By default, the data window is determined automatically by image histogram (see tutorial section 3.4 on Intensity Range Partitioning for more information).

Selecting data used for alignment

In some cases, automatic alignment can be distorted by anomalies that cannot be tracked consistently across all slices, such as bright spots or artifacts appearing on individual slices. Here are some hints to solve this:

- When using grayscale image input, selecting an appropriate data window is a way to filter out unwanted high or low pixel values.
- *Align Slices* can use a label image connected to its *Mask* input port to restrict the region considered for automatic alignment. You can easily produce such a mask by using, for instance, the *Segmentation Editor* or the *Volume Edit* module.
- In complex cases, you could perform alignment based on any subset or derivative of initial image stack (e.g., cropped, filtered, or segmented), and then reapply the alignment to the original data.

Tuning alignment options

Here are some guidelines for tuning automatic alignment. For accessing *Align Slices* options, select the menu *Align / Options* when *Align Slices edit* mode is active.

- If alignment of a full stack fails, you may save time by trying to identify the first slices that fail and improving alignment for those slices at first.
- In the *General* tab, uncheck the *Allow Rotation* checkbox to disable rotation in the alignment.
- In the *Least Squares* tab, you can increase the *Max number of iterations* (e.g., to 5000) if the alignment process stops before correct match.
- In the *Least Squares* tab, you can reduce the Resample size *scale factor* if a slice 'flees' out of the canvas. This may happen if there is not enough information or overlapping for meaningful quality measure at the coarse resolution that is used to speed up the first steps of the process.
- You can fix the reference slice used for alignment across the whole stack with menu *Options / Fix Reference*. This could be used in combination with an input Mask image to select a limited image area as a fixed reference.

More about alignment data parameters

The slice alignment transformations are stored in a *data parameter* bundle *AlignTransform* and there-

fore can be written into the file header of the aligned stack. Such data parameters can be viewed with the *Data Parameter Editor*.

For example,

```
slice044 6 4 -16.3025 -1
```

indicates that slice ♯ 44 was translated 6 voxels in X, 4 voxels in Y, and rotated -16.3025 degrees about Z. The -1 means that the slice was not mirrored (If 1, it would mean that the slice has been mirrored).
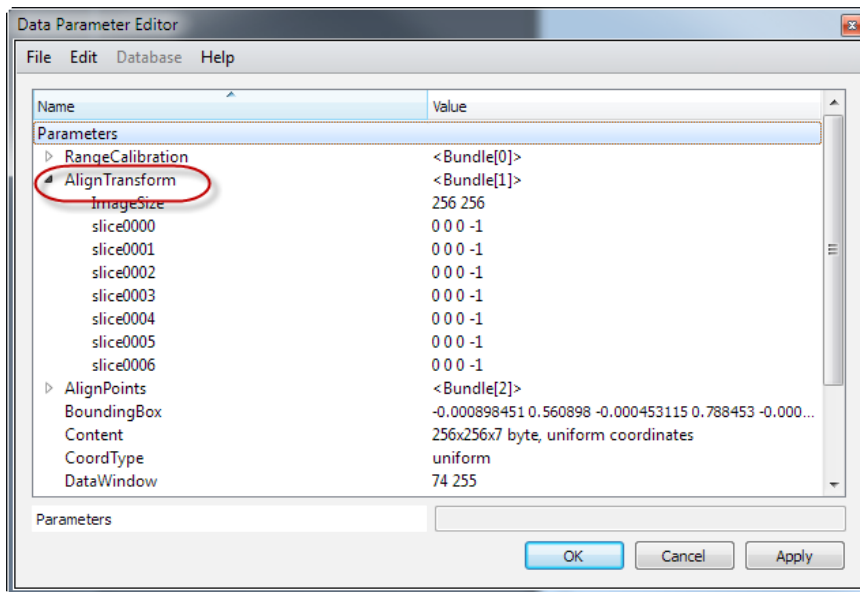
Figure 6.68: Alignment data parameters

6.7 Alignment and pre-processing of FIB/SEM images stacks using the FIB Stack Wizard

The capture of image stacks using FIB/SEM devices (Focused Ion Beam / Scanning Electron Microscope) may require alignment and other pre-processing such as foreshortening, shear, and shading correction. A script module *FIB Stack Wizard* is available to assist this workflow.

FIB/SEM is a powerful technique for imaging samples in 3D at the nanometer scale, optionally with compositional information. A FIB/SEM DualBeam device combines a Focused Ion Beam for milling serial cross sections in the sample, and a Scanning Electron Microscope for imaging the milled 2D

sections. A variety of detectors (SE, low kV BSE, EDX, EBSD, etc.) can be used for analyzing the sample's materials and structure.
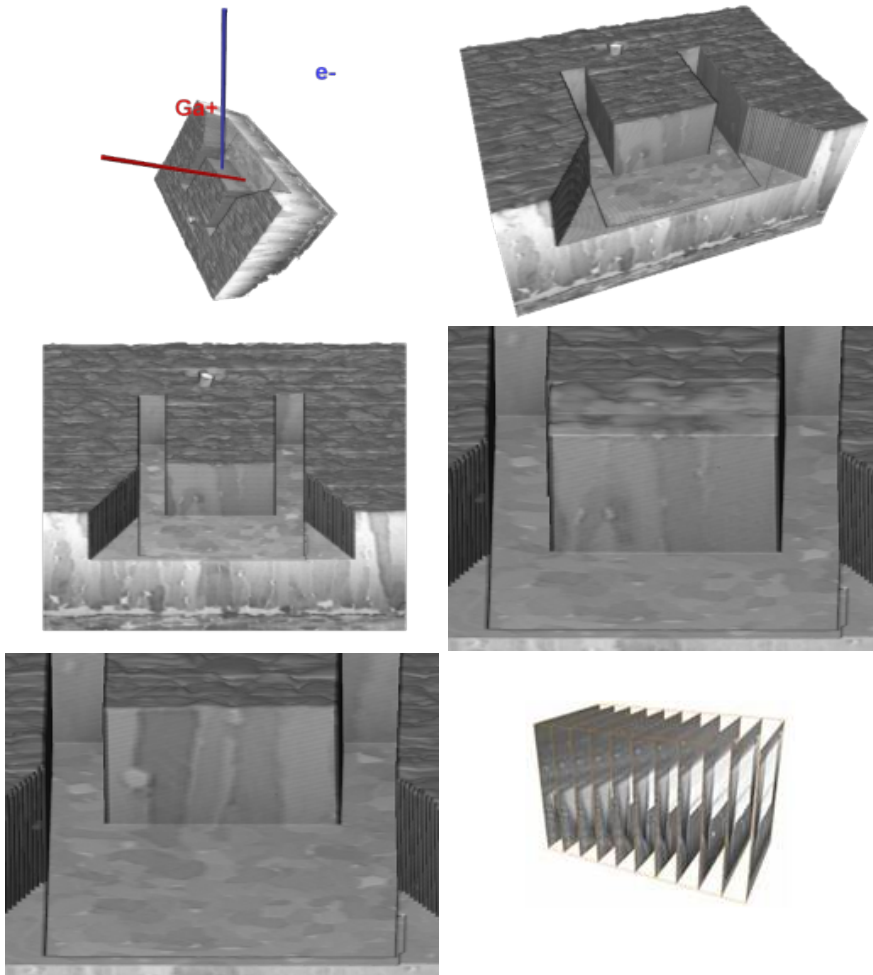
Figure 6.69: FIB/SEM principle: overviews, SEM views, serial sections

After such a FIB serial sectioning acquisition, a 3D model can be reconstructed from the 2D images. Depending on the input data and purpose, some pre-processing including alignment and other steps may be required, in particular to ensure the accuracy of the 3D model for further quantitative analysis.

The angle between the ion beam and the electron beam is typically 52 degrees. Because of angle between milled surface and imaging axis is not 90 degrees, the raw captured images shows a geometrical

artifact compared to real sections:

- Apparent vertical shift upward that grows with each image in the stack
- Foreshortening: vertical size (y axis relative to capture) appears compressed.

You may get images already corrected for these effects, but in some cases it may be necessary to compensate them with downward shift, i.e., stack shearing and stretching.

Another unwanted artifact is lateral drift that may occur, especially during long acquisitions, yet even shorter ones can show some jittering drift between images, due to environment vibration, for instance.

In addition to geometrical artifacts, some further image processing may be needed to correct for noise or non-uniform illumination (sometimes described as shadowing).

Avizo provides a number of tools for processing and analyzing FIB/SEM image, including a convenience script module that steers you along the most common processing steps: the *FIB Stack Wizard*.

In this tutorial, you will learn in particular how to use the FIB Stack Wizard and other Avizo tools for:

- Importing and calibrating voxel size
- Geometry correction and cropping
- Shading correction
- Image filters

Advanced users familiar with scripting may look at the FIB Stack Wizard as an example of a workflow-driven script module.

6.7.1 Images import, voxel size and foreshortening correction

The data set used in this tutorial is derived from images of a sample of molybdenum disilicide (MoSi2), a material used in furnace heating elements, by courtesy of C. Kong, University of New South Wales, Australia. The original images (2048x1768 x 300 slices) have been modified for the purposes of this tutorial.
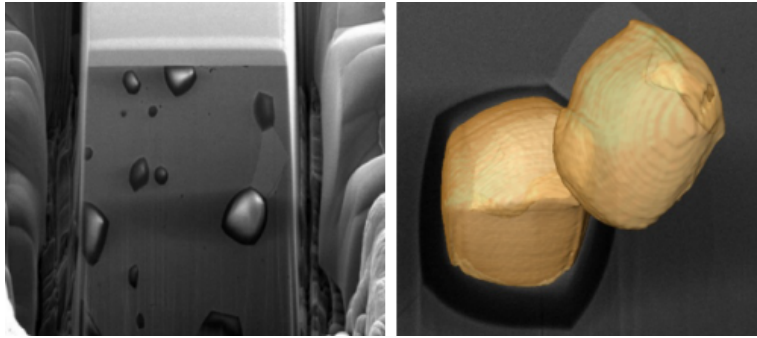
Figure 6.70: MoSi2 sample - courtesy C. Kong, University of New South Wales

- With menu *File / Open Data*, load the data file `MoSi2-shear-corrected.am` located in subdirectory *data / fib* in Avizo installation directory. This image stack was saved as a native Avizo data file. About loading stacks of image files, see the tutorial section 3.1 (How to load image data).

- You can attach one or two *Ortho Slice* modules to `MoSi2-shear-corrected.am` to examine the data, or use the *Ortho Views* module to set up 4-view display.
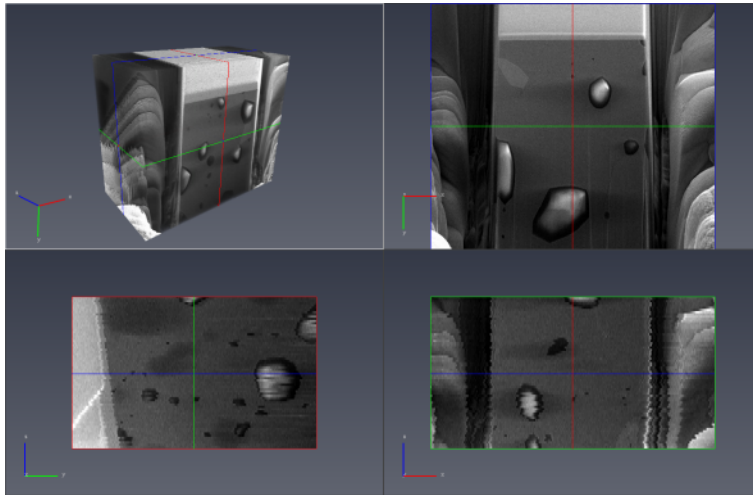
Figure 6.71: Data displayed using Ortho Views template

The full data set extent is 64 micrometers in width and 36 micrometers in depth, i.e., for the resampled images about $0.125\mu m$ pixel size (X-Y field-of-view) and $0.486\mu m$ slice thickness (Z voxel size).

Depending on the input file format, you can specify the pixel size and slice thickness as 'voxel size' when loading the data if information could not be retrieved from the file. You can always check and adjust it afterwards by using the *Crop Editor*.

- Activate the *Crop Editor* for `MoSi2-shear-corrected.am` to see the bounding box extent and corresponding voxel size in the dialog box. You can then deactivate the *Crop Editor*.
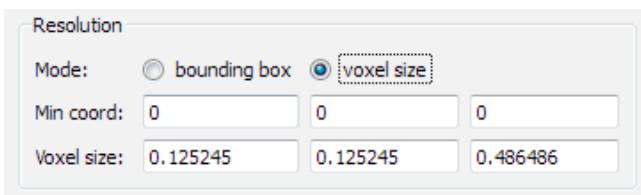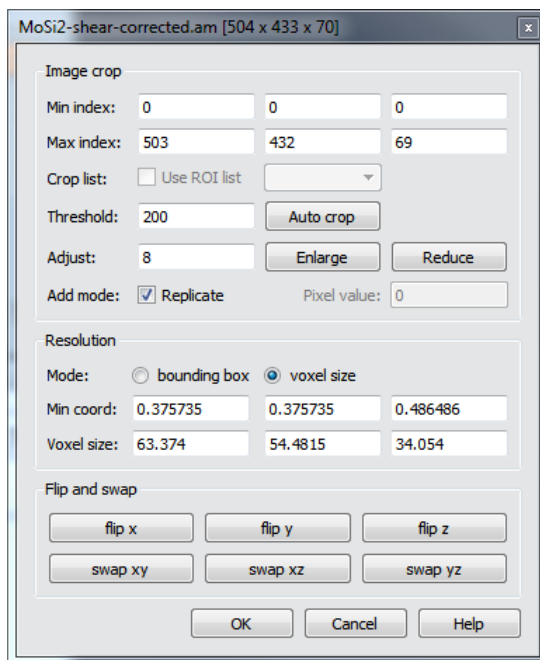
Figure 6.72: Bounding box and voxel size

Non uniform slice thickness

If variation of slice thickness matters for your application, you can import images using the Avizo Stacked Slice file format, which defines the depth for each individual slice. This is described in the tutorial section 3.1 (How to load image data). A number of Avizo tools can be directly applied to such

data with irregular, so-called "stacked slice" coordinates. You may also turn the images into a uniform scalar field using *Arithmetic* or *Resample* modules (use a uniform lattice as reference input for the *Resample* module).

Foreshortening

Depending on how the data was collected, you may have to correct the foreshortening effect seen in introduction. If you entered the field-of-view for the Y-axis, then there will be no need to make this correction. If you entered the same voxel size for x and y, and the instrument did not already apply a stretching correction, then you may want to correct by entering the corrected voxel height directly into the Crop Editor. To apply the foreshortening correction in the *Crop Editor*, multiply the voxel height by a correction factor, depending on the angle between FIB and SEM columns. For instance, for a 52 degrees angle: $1/sin(52°) = 1/cos(90°-52°) \approx 1.269$. The file `MoSi2-shear-corrected.am` was already corrected for foreshortening.

Measurement units

Note: Indicating "nm" as *Display units* in the *Preferences* will output measurements in nm. To learn about unit management in Avizo, see the section 8.2.9 (Units in Avizo) in Avizo User's Guide.

6.7.2 Geometric corrections overview

Upward shift / shearing

Beside the voxel size, the collected images may need further kinds of geometry correction.

In raw images as seen from the SEM viewing angle, the sections appear shifted, while the surrounding background or trench stays aligned horizontally across the series. Obviously the 3D structures inside the sections looks "sheared" on an YZ slice and need to be corrected.
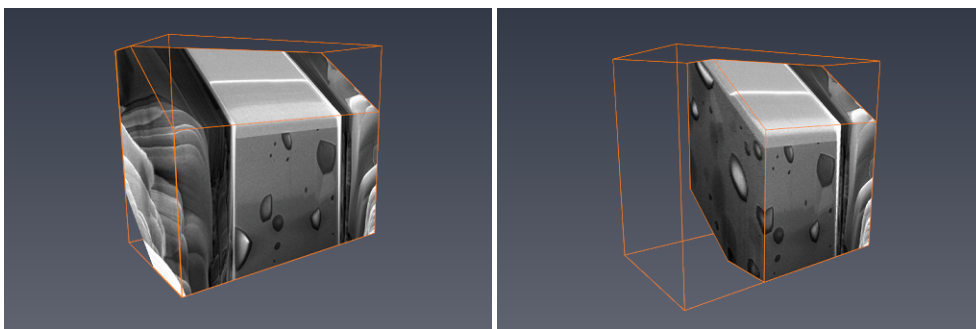
Figure 6.73: Sheared image stack

Raw image stack display showing vertical shift of the serial sections. A clipping oblique Slice is used to highlight the sample slope.

We'll see later in this tutorial how to address this case using data set `MoSi2-sheared.am`.

The data set `MoSi2-shear-corrected.am` was already corrected for electron beam angle as you can see below. The sample sections are aligned horizontally, while the surrounding trench looks sheared in turn since it is fixed relative to the microscope raw images.
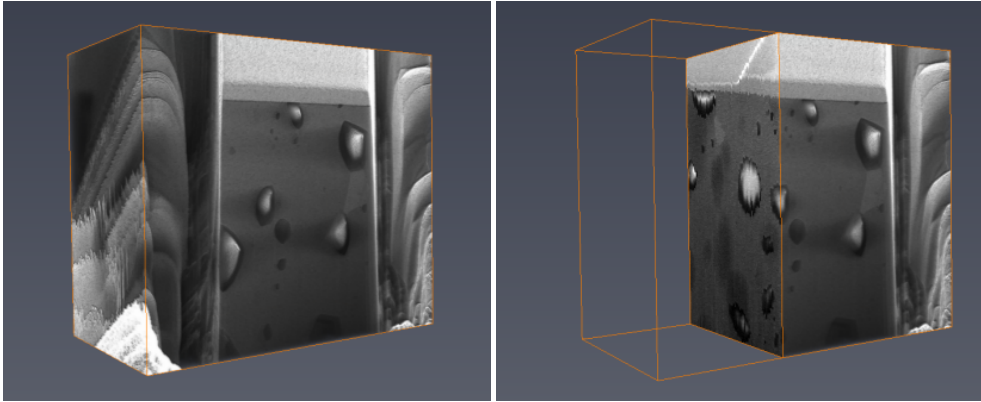
Figure 6.74: Shear-corrected image stack

Drift correction

Images corrected for viewpoint angle may still need correction from drift as show below, mostly visible in XZ slices of `MoSi2-shear-corrected.am`.
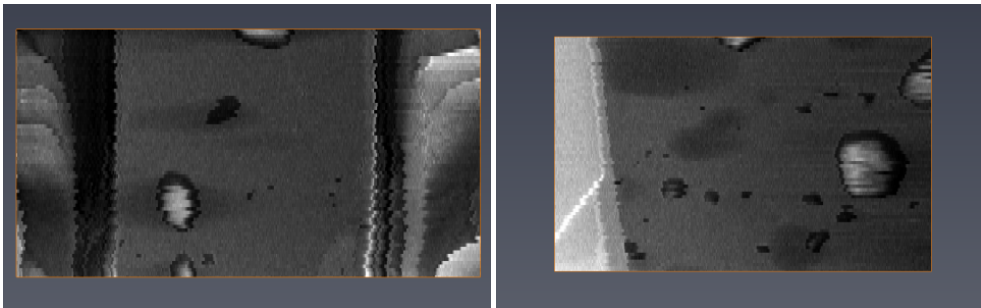
Figure 6.75: XZ and YZ slices with jittering effect

Cropping

Cropping is necessary to remove unwanted areas in the 3D reconstruction, such as the trench surrounding area, or textual information that may be located at the bottom of the images.

6.7.3 Getting started with FIB Stack Wizard

The *FIB Stack Wizard*, accessible from the menu *Geometry Transforms / FIB Stack Wizard*, can be used for alignment/shearing correction as well as for shading correction. The wizard module guides you step by step along the process. Data will be duplicated since the wizard allows you to go backward in the steps to correct specific actions. The following steps are proposed by the *FIB Stack wizard*:

1. Initial cropping before alignment
2. Automatic alignment
3. Crop after alignment
4. Shearing correction
5. Shading correction

- You can now remove any remaining *Ortho Slice* or other display modules in the project: the *FIB Stack Wizard* automatically sets up an *Ortho Slice* display.
- Attach the module *FIB Stack Wizard* to `MoSi2-shear-corrected.am`.

An *Ortho Slice* with a tight *Color Wash* module is attached to the data set, and a *Crop Editor* dialog is displayed.
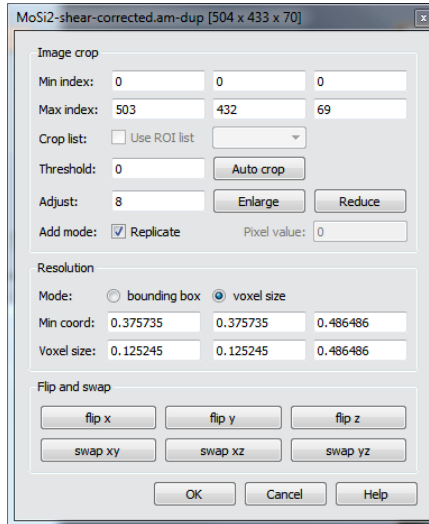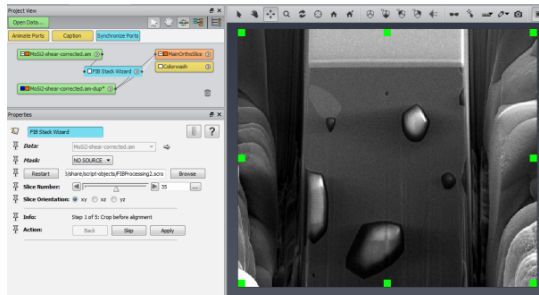
Figure 6.76: FIB Stack Wizard at step 1

Once the *FIB Stack Wizard* is selected in the Project View, its control ports are displayed in the *Properties Area*.

A *Mask* input connection port lets you attach a label image serving as a mask during alignment. This will be detailed later.

- You can use the *Slice Number* port to change the displayed slice along current axis.
- You can change the *Slice Orientation*.

The *Info* port indicates the current processing step. Buttons are available to proceed to next step, possibly skip it, or go back to previous step.

Step 1: Crop before alignment

This step lets you crop the input data before applying the alignment. For now you will select a region contained within the serial sections.

- Drag the green corners of the tab box in the viewer, or index values in the *Crop Editor* dialog.
- You can use the *Slice* number and orientation ports to control the selected region.
- Be sure to slice all the way from the front of the stack to the back of the stack to verify that you are not cropping out desired regions. You may also change temporarily the *Slice orientation*.
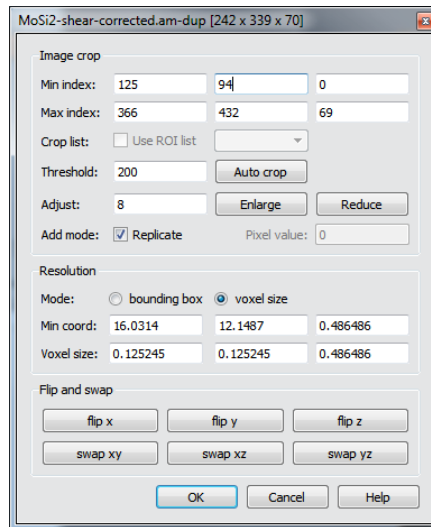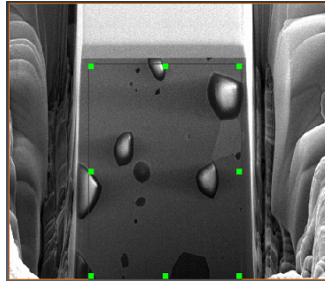
Figure 6.77: Crop before alignment with FIB Stack Wizard

- Once set, press on the *Apply* button in order to go to the next step.

Step 2: Alignment

In this step, the wizard will use automatically the *Align Slices* module. The wizard exposes the following alignment options:

- *Gravity centers*: Align gravity centers and principal axes.
- *Least-squares*: Least squares algorithm based on gray values.
- *Enable rotations* options: enables rotations during slice alignment. If this options is disabled, only translations will be computed.

- For this example our case let's keep the default values: least squares alignment method without rotation.

Important note: when processing grayscale images, the visible gray values for display and for alignment can be restricted to a *data window* defined for the data set. Only the values between the two bounds of the data window are used by the gray value-based alignment algorithm. You can check if an appropriate data window is defined in the data information displayed in the *Properties Area*. For editing the data window, see the *Intensity Range Partitioning editor* and tutorial section 3.4 on Intensity Range Partitioning.
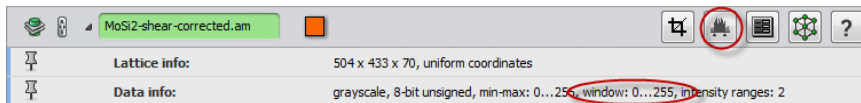
Figure 6.78: MoSi2-pre-corrected Data window

- Press on the *Apply* button in order to go to the next step. The *Align Slices* editor window is displayed showing the alignment progression. You can see how slices are aligned relative to each other. Depending on the distribution and the orientation of shapes across the slice stack, some smooth drift may occur.

Step 3: Crop after alignment

Slices of the input data are now aligned relative to each other. The jittering drift effect has mostly disappeared.

- Change slice number to browse slices. You can see black uncovered area as slices have been translated.
- You can select a new crop region to exclude these black areas.
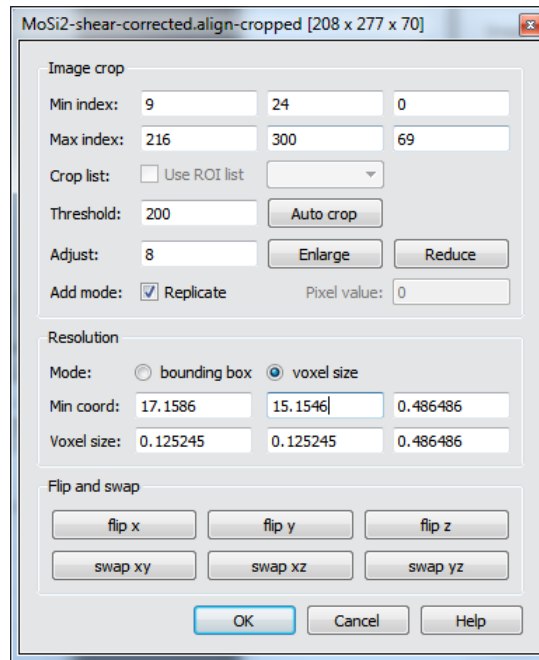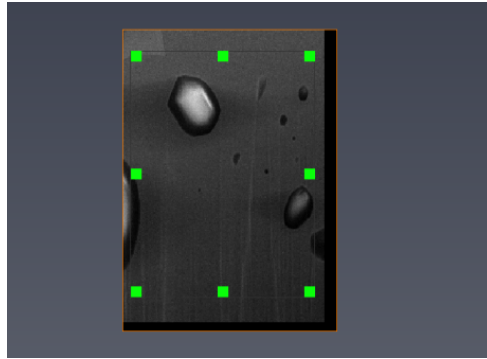
Figure 6.79: Crop after alignment with FIB Stack Wizard

- Once set, press on the *Apply* button in order to go to the next step.

Step 4: Shearing correction

This step may be needed when dealing with raw microscope images without shift correction, or when alignment should be performed with a mask corresponding to a fixed part (e.g., landmark) in the images. This will be detailed later.

- For now, alignment has been performed, so click on the *Skip* button in order to go to the next step.

Step 5: Shading correction

This step can help to compensate for non-uniform illumination, for instance, to facilitate future image segmentation. The voxel values are normalized by an estimated background. You can specify a range of intensities for voxels to be considered for estimating background. The number of electrons detected can be altered by the surrounding materials, in particular when trench is not large enough or when redepositon causes mass to accumulate that reduces electron detection.

- Change the Low mask threshold value. You can see the non-uniform distribution of background intensity. See the figure 6.80 below to adjust values.
- Change both thresholds to cover as much as possible the area to be approximated for background, excluding as much as possible the dark or bright features that could alter the background estimate. See the figure 6.80 below to adjust values.
- The *shading correction* parameter is a factor applied to the voxel values normalized according to the estimated background.
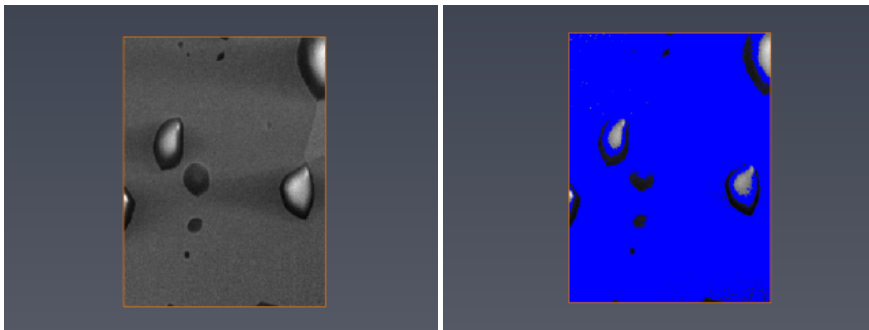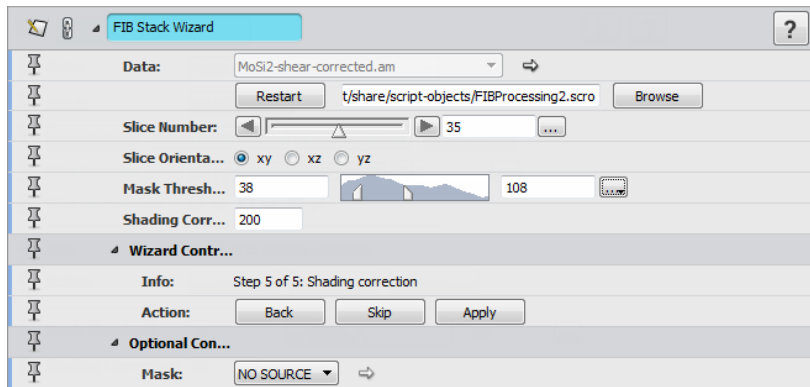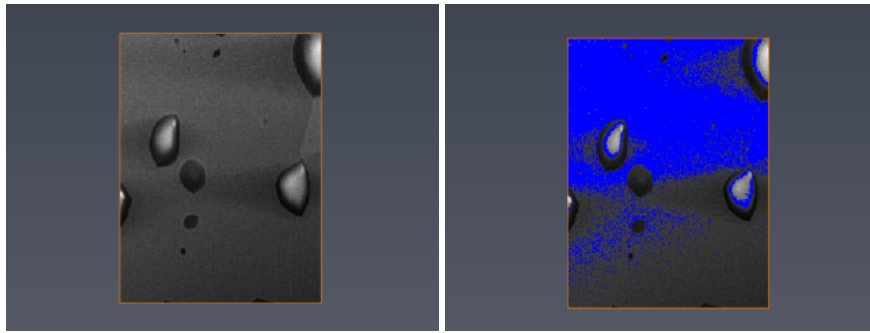
Figure 6.80: Shading correction with FIB Stack Wizard

- You can click on the *Back* button in order to go to previous steps and modify some processing parameters.
- Remove the *FIB Stack Wizard* in order to delete intermediate modules and data objects.

6.7.4 Selecting what to align using masks

Three methods can actually be used to choose what to align:

1. Cropping the data set.
2. Changing the data window using the Intensity Range Partitioning editor (or Align Slices data window port).
3. Specifying a 3D mask.

With previous steps, you could achieve quickly reasonable results, with limited image drift. However, in some cases, you may need to select more precisely the image information to be considered for alignment.

1. The overall image structure may induce drift, in particular if there are few large or oriented structures. Subsets of images might better drive the alignment.
2. You may want to take advantage of fixed markers or fiducials. If the volume and resolution you wish to capture allow for it, you can keep such markers in the field of view for the purpose of accurate alignment.
3. You may have raw images with uncorrected upward shift. For best results you should normally apply first a shearing before aligning the sections area so that images are roughly axis aligned. You can use the Avizo*Shear* module for this. Alternatively, you could instead align images based on the fixed area surrounding the sections.

- Load `MoSi2-sheared.am` located in *data / fib*
- Attach *FIB Stack Wizard*
- Press *Apply* to keep all volume at *Crop* step
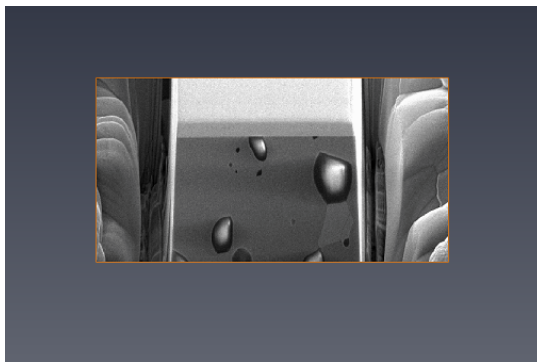
Figure 6.81: FIB Stack Wizard at step 1

We now need to create a mask to be used as input by the *FIB Stack Wizard*. For this one could use the *Segmentation Editor* or any other tools creating a label image. Using the *Volume Edit* module is convenient here.

- Attach `MoSi2-sheared.am` to *Compute / Volume Operations / Volume Edit*.
- In the *Properties Area* leave the *Tool* port of *Volume Editor* set to *Draw*.
- Click on the *Outside* button of the *Cut* port: you can then encircle with a lasso a region in the 3D viewer. You can hold Alt key to draw straight lines. The drawn contour defines an extruded volume along the view axis. Make sure to use the viewer's orthographic camera mode in order to make the contour extrusion parallel to the Z axis. (for this press the *Perspective/Orthographic* button in the viewer's toolbar).
- Pressing the *Create Mask* button will then create a label image corresponding to that region.
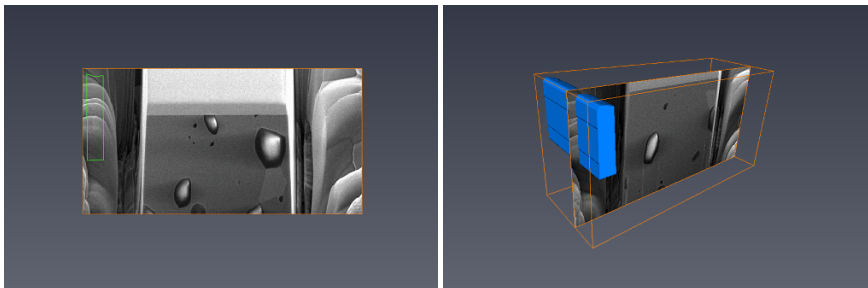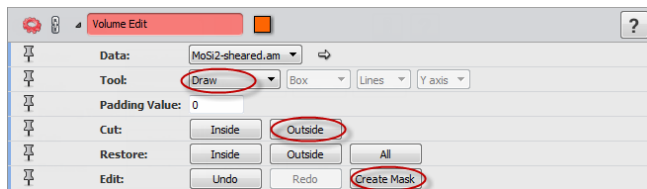
Figure 6.82: Createting mask for MoSi2

- Attach the result mask `MoSi2-sheared.mask` as *Mask* input for the *FIB Stack Wizard*.
- Press the *Apply* button to trigger alignment. The *Align Slices* editor window shows the image masked by the region you defined.
- Press *Apply* to keep all volume at *Crop* step.
- Apply *Shear* step with -38 degrees (i.e., 52 - 90).
- Proceed to *Shading correction* as above.

6.7.5 Further processing of FIB Stacks

More image processing may be required to further improve FIB/SEM images, for compensating for instance noise or curtaining effects. Filtering can be applied either before or more usually after alignment such as processing by *FIB Stack Wizard*. In particular if you want to apply image filters in 3D mode, images should be properly aligned.

See section 4.7 for hints about image filtering. In particular, the *Non-Local Means* image filter is usually effective with FIB/SEM image. See also *Filter Sandbox* for trying conveniently image filters.

6.8 Registration of 3D surfaces

The *Transform Editor* and *Landmarks* tools introduced in previous tutorials may be used for surface registration. This section focuses on automatic optimized registration based on surface distance minimization. In this tutorial, you will learn how to register 3D triangulated surfaces that are wholly or partially overlapping.

This section has the following parts:

- *Getting started with Align Surfaces module*
- *Align Surfaces guidelines*
- *Alignment of surface subsets*
- *Measure and visualize surface distance*

6.8.1 Getting started with Align Surfaces module

Align Surfaces is the main tool used for surface registration. The module *Align Principal Axis* can also be useful for alignment constrained along certain axes.

The *Align Surfaces* module allows the automatic alignment of a triangulated surface with respect to a reference surface. It computes either a rigid transformation or an affine transformation.

The following example shows step by step how to register two surfaces. We start with similar first steps used in the *Transform Editor* tutorial.

- Start a new Avizo Project
- Load `chocolate-bar.simplified` surface file from the *data / tutorials* directory,
- Duplicate the `chocolate-bar.simplified` data object (Ctrl-D).
- Attach two *Surface View* modules to `chocolate-bar.simplified` and `chocolate-bar2.simplified`. Both surfaces are shown overlapping for now.
- Activate the *Transform Editor* for `chocolate-bar2.simplified`, and change position and rotation of `chocolate-bar2.simplified`.

- Press the *Apply Transform* button in the *Transform Editor* to modify the vertex coordinates according to the transformation. You can confirm the transformation when asked.
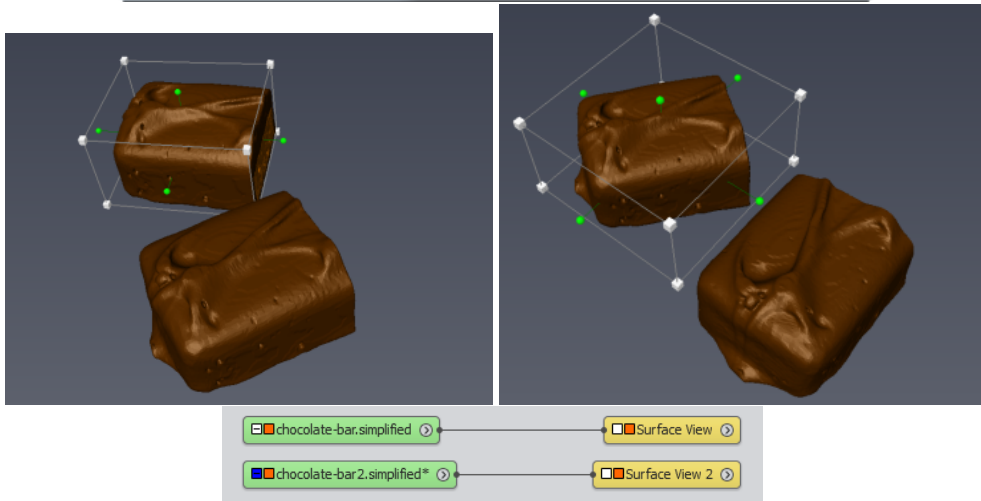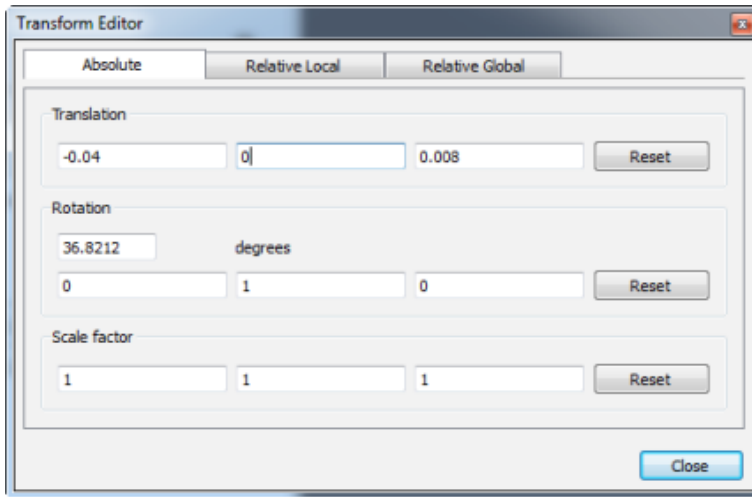
Figure 6.83: Surface transformed

In order to introduce some differences between the surfaces before experimenting with alignment, we will use the *Surface Simplification Editor*. This tool reduces the number of triangles by edge collapsing and vertex shifting to approximate the original surface. This operation can also be very useful to reduce the computation time for alignment, especially for large surfaces.

- Select `chocolate-bar2.simplified` in the Project View. Then in the *Properties Area*, activate the *Simplification Editor*.
- In the port *Simplify*, set the desired number of faces to 10000. You can check *fast* toggle for quicker computation.
- Press the *Simplify now* button in the *Action* port.
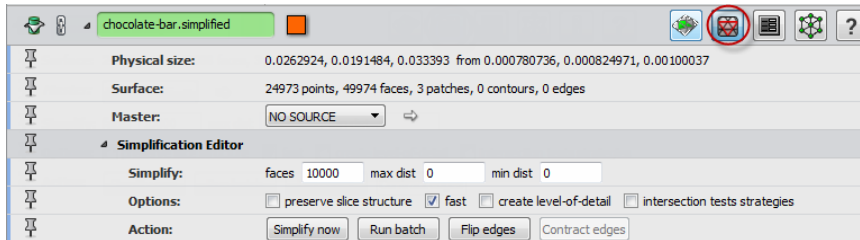
Figure 6.84: Simplification Editor

- Attach the *Geometry Transforms / Align Surfaces* module to `chocolate-bar2.simplified`, i.e., the surface to be transformed.
- Then connect the *Reference surface* port to `chocolate-bar.simplified`. Leave the default parameters for now.
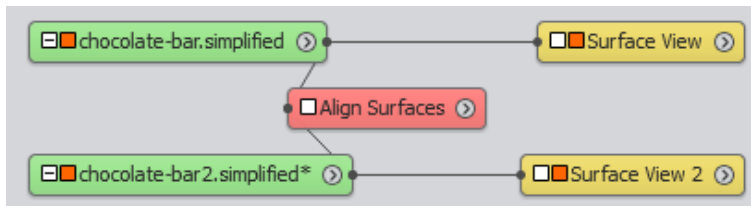
Figure 6.85: Align Surfaces project

Let's focus for now on the *Align* port, exposing two pre-alignment methods and the actual optimized registration.
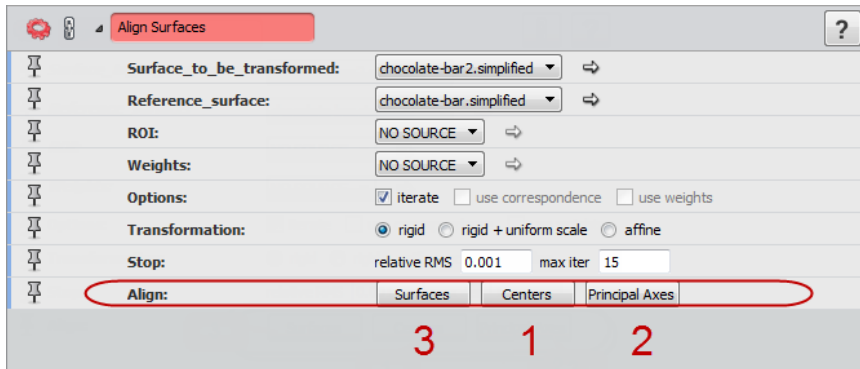
Figure 6.86: Align Surfaces properties area

- Press the *Centers* button in the *Align* port.

You can see that the surfaces are now centered. However, the model orientation does not match the reference orientation.
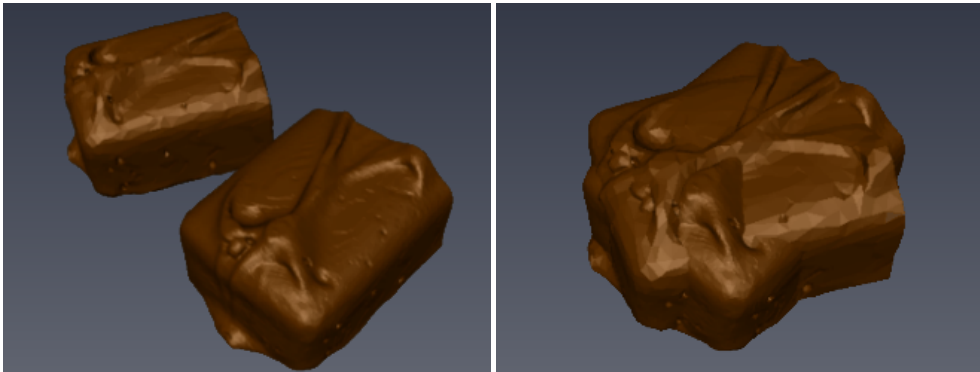
Figure 6.87: Align: Centers result

- Now press the *Principal Axes* button in the *Align* port.

The model and reference are now almost aligned. You can still notice some discrepancy.

- Press the *Surfaces* button in the *Align* port. This starts the iterative alignment optimization.

After a few steps, the surfaces are aligned as well as possible.
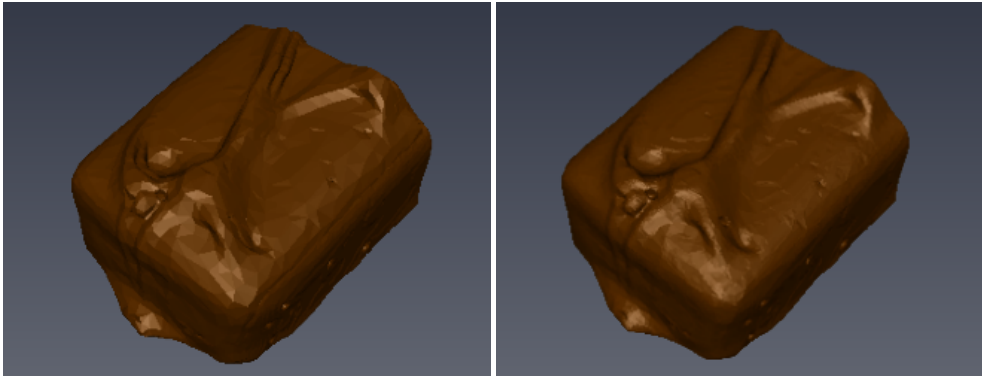
Figure 6.88: Align: Principal Axes and Align: Surfaces results

Note: you could alternatively use an *Ortho Views* module for display, for instance, to overlay cross sections or cross contours of the surfaces.
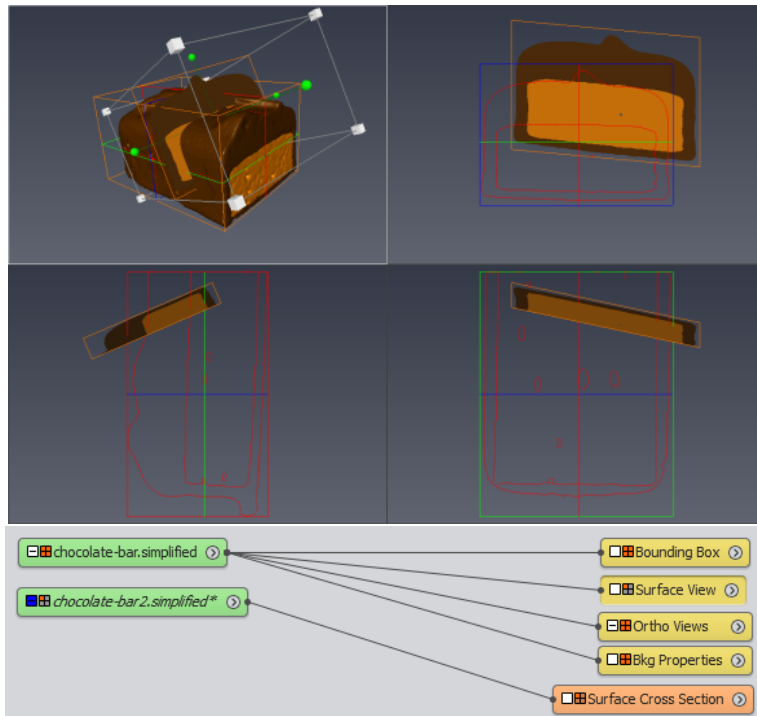
Figure 6.89: Ortho Views display

6.8.2 Align Surfaces guidelines

The *Align Surfaces* module based on an Iterative Closest Point algorithm, repeating the following steps:

1. Associate corresponding points in the two surfaces by nearest neighbor criteria.
2. Estimate and apply a transformation minimizing the root mean square distance between the points of the model surface and the corresponding points on the reference surface.
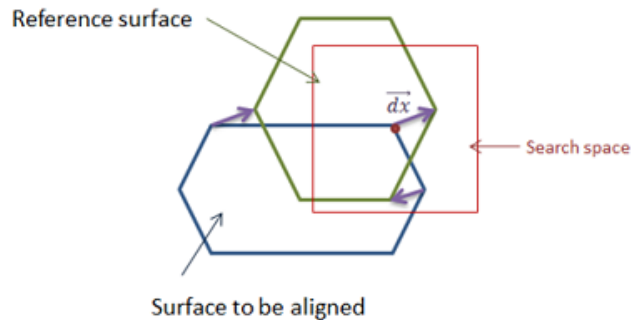
Figure 6.90: Align Surfaces principle

Despite optimized search for nearest neighbors, this can be time consuming especially for large surfaces. The center of mass and principal axes can be aligned quickly.

Like for image registration, in order to limit computation time and risk for mismatch, it is recommended to perform the surface alignment in two steps.

1. Pre-alignment with manual or automatic approximate registration
2. Automatic refined registration

Pre-alignment could be done using the Transform Editor or Landmarks. For automatic alignment, you can more simply attempt in order:

1. Centers
2. Principal Axes
3. Surfaces

Here are some additional hints.

Alignment of the centers of mass.

To align the centers of mass, all vertices of the triangulated surface are assigned the same mass. Therefore, the calculated centers may depend on distribution of vertices along the surface. In some case, it may be helpful to remesh surfaces, using either the *Surface Simplification Editor* or the *Remesh Surface* module.

Alignment of the principal axes.

The moments of inertia and principal axes are calculated, again with the same mass assigned to all vertices of the triangulated surfaces. Since there can be different matching orientations for the three principal axes, each combination is checked and the final solution is the one with the minimum root mean square distance between surface vertices. Usually this gives good results as a starting point for refined registration. However, in some cases, the distribution of surface vertices can be such that a wrong orientation is selected.

Surfaces distance minimization - iterative refined registration.

You can reduce the search in a number of ways to accelerate processing if needed:

1. Degrees of freedom should be left to minimum required.
2. You may not need to register the full surfaces.

   - You can select the Region of interest in the reference surface (ROI port) corresponding to the model, or to an extracted subset of the model
   - You can extract the parts of the surfaces most relevant for registration - this can be done using *Surface View* and *Extract Surface* modules.
   - You can simplify surfaces or Remesh surfaces.
   - You can the copy the transformation obtained to the full data set.

3. Make sure that the *use correspondence* option is set if the input surfaces have exactly matching vertices, stored in the same order. Registration is then dramatically faster. If the surfaces accidently have the same number of points and if these vertices are not matching, then make sure that the option is disabled.

6.8.3   Alignment of surface subsets

In this an example, we will consider two partially overlapping triangulated surfaces. These surfaces have been extracted separately, using an *Isosurface* module, from the two independent volumes used in Image Registration tutorial. The surfaces cannot overlap exactly due to differences in volumes sampling, but we will search now the best registration.

- Load `chocolate-bar.part1.simplified.surf` and `chocolate-bar.part2.simplified.surf` in *data / registration* directory
- Attach *Surface View* modules to `chocolate-bar.part1.simplified.surf` and `chocolate-bar.part2.simplified.surf`.
- Register approximately the two surfaces with a rigid transformation by using the *Transform*

Editor or as shown in tutorial about Landmarks registration.
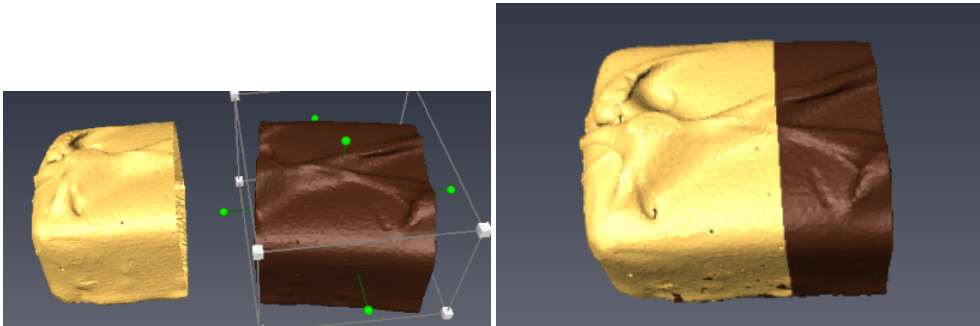
Figure 6.91: Approximate registration

- Attach an *ROI box* to chocolate-bar.part1.simplified.surf. The Region Of Interest must delimit the common region between the two data sets. It should not be much larger.
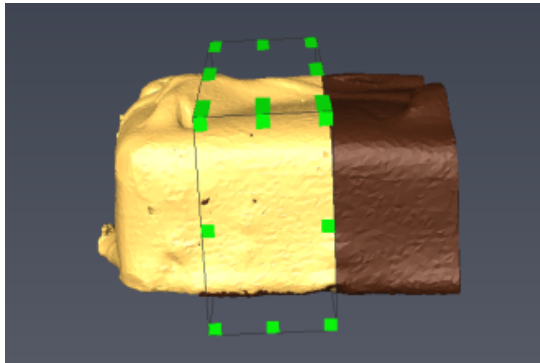
Figure 6.92: Region of Interest (ROI Box)

- Attach the module *Geometry Transforms / Align Surfaces* to `chocolate-bar.part2.simplified.surf`. Set the reference surface to `chocolate-bar.part1.simplified.surf`, and the ROI to the created *ROI box*. Select *rigid + uniform scale* transformation, and set the *max iter* port to 100.
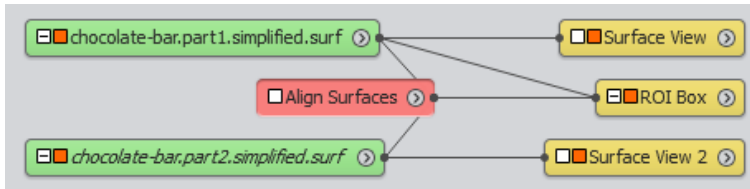
Figure 6.93: Align Surfaces project

- Click on the *Surfaces* button in the *Align* port. At most 100 iterations will be computed for the alignment. If the relative RMS is smaller then 0.001, the algorithm will end too. You can press the *Stop* button to interrupt the registration. You can repeat this step if the alignment is not accurate enough: the relative RMS should be reduced.
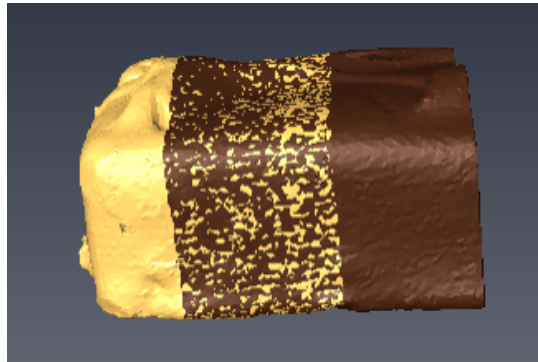
Figure 6.94: Refined registration

Once you obtained an optimized registration, you can optionally check the surface distance using a *Surface Distance* module.

6.8.4 Measure and visualize surface distance

In the following steps, you will compute and display the map of distance between the two surfaces. The *Measure / Surface Distance* module computes several different distance measures between two triangulated surfaces based on closest points.

- You can remove the *Align Surfaces* module.
- Apply the refined transformation to `chocolate-bar.part2.simplified.surf` (*Transform Editor* / *Apply Transform* button).

- Attach the *Measure and Analyze / Surface Distance* module to `chocolate-bar.part2.simplified.surf`. Set *Surface2* port to `chocolate-bar.part1.simplified.surf`.
- Also attach the *ROI* port for specifying the relevant Region Of Interest, already used in previous steps for the registration.
- Select *Distance* output and press *Apply*. This will create a scalar field attached to the surface.
- Set the colorfield of *Surface View 2* (displaying `chocolate-bar.part2.simplified.surf`) to the computed distance. Choose `physics.icol` as colormap, and *adjust range* to [0, 0.01].
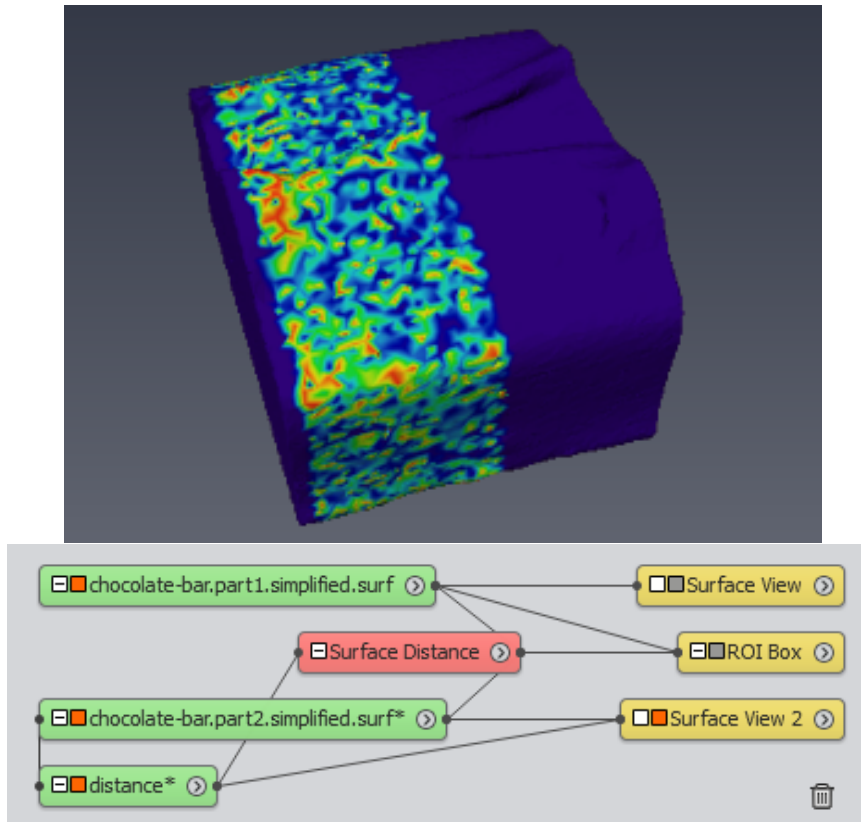
Figure 6.95: Distance between the surfaces - Viewer and network
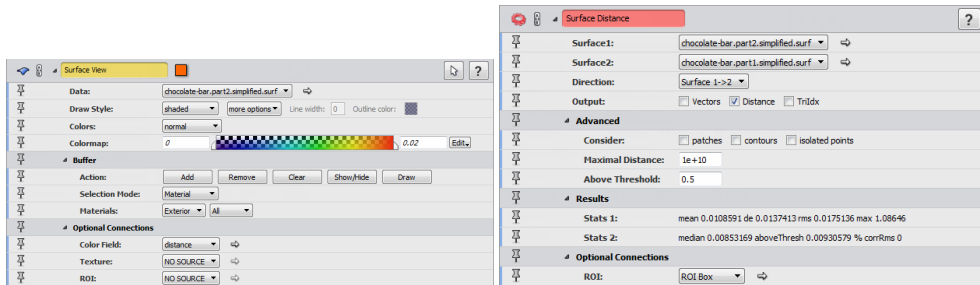
Figure 6.96: Distance between the surfaces - Surface View and Surface Distance

The computed distance statistics and distance field may depend on the distribution of vertices over the two surfaces. You may want to choose the two-sided Direction for a symmetric distance calculation. You could also remesh the surfaces using the *Surface Simplification Editor* or the *Remesh Surface* module to redistribute vertices uniformly.

A *Surface Thickness* module and a *Shortest Edge Distance* module are also available.

Note: For checking surface distance from objects in a 3D volume, one could compute a 3D distance map from the segmented 3D image (using *Image Processing / Distance Maps / Distance Map*), then use the result as a color field with *Surface View*, as shown in *Data fusion* tutorial, or use *Compute / Surface Scalar Field* to attach the distance measures to the surface.

6.9 Registration of 3D image and surface, nominal-actual analysis

A number of applications require registering a 3D model image with respect to a reference 3D surface. In the following example, you will see how to simply register and compare a CT scan image of a product with a corresponding CAD model.

This section has two parts:

- *A simple workflow for nominal-actual analysis*
- *Advanced - Image/surface registration step by step*

Note about file formats: Beside the many 2D and 3D images format that are supported by Avizo, Avizo can also directly import a number of geometric and surface data formats such as STL or Open Inventor. Additional readers for CAD files formats such as CATIA 5, IGES, and STEP are available in the Avizo XReader extension. The Avizo XWind Extension and bundle provides import capability for

surfaces and meshes in a number of standard or commercial numerical simulation formats.

6.9.1 A simple workflow for nominal-actual analysis

We will now use a wizard module that manages for you the registration and comparison process.

- Open the `NominalActualComparison.scro` wizard in *data / pump-bracket* directory.
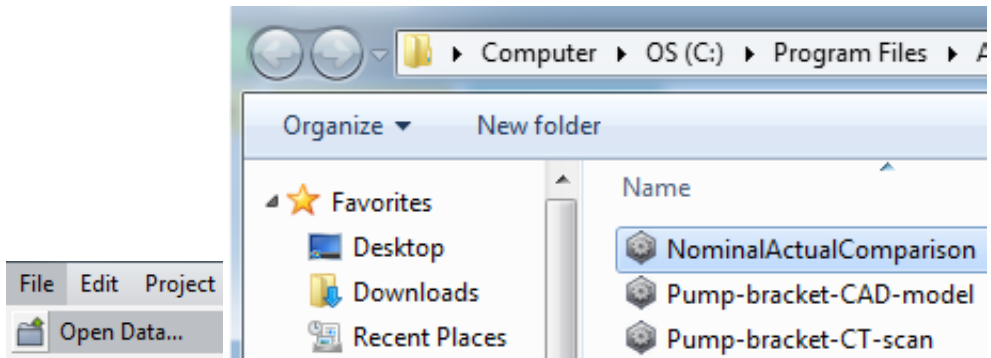
Figure 6.97: Opening NominalActualComparison wizard

You then see the *NominalActualComparison* object icon in the Project View. Its user interface appears in the *Properties Area*.
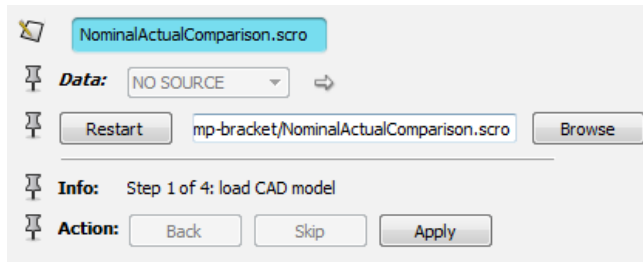
Figure 6.98: NominalActualComparison properties area

- In the *Properties Area*, press *Apply*. You are then prompted for loading the reference CAD geometry file.
- In the *Open* dialog, choose `Pump-bracket-CAD-model.surf` (in *data / pump-bracket* directory), then press the *Open* button.
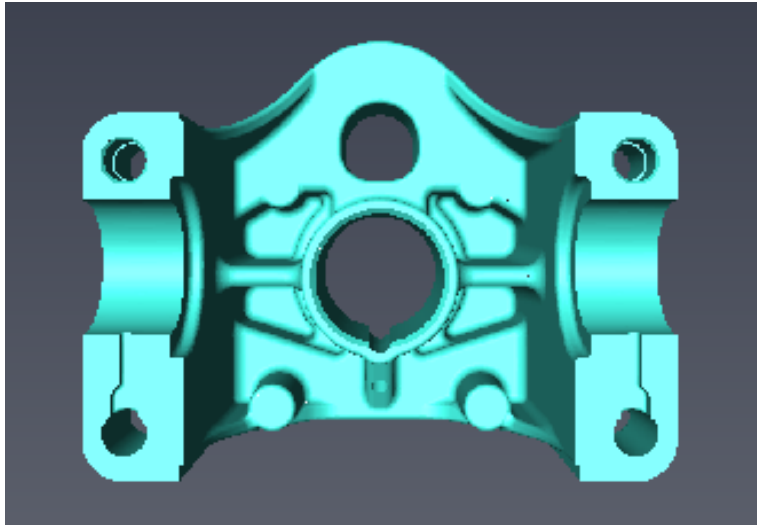
Figure 6.99: CAD surface

The CAD surface is displayed in blue. The next step is to load the model's CT scan 3D volume.
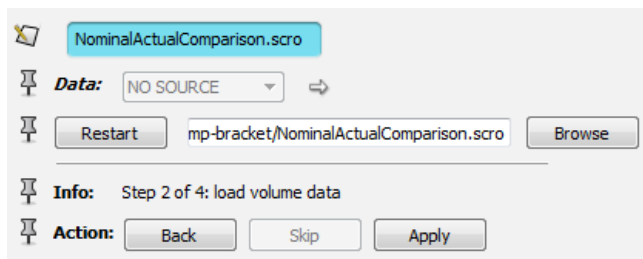
Figure 6.100: Press Apply to load volume data

- Press *Apply*, then in the *Open* dialog select `Pump-bracket-CT-scan.am`, and press the *Open* button.

Note: This data set corresponds to the CT scan acquisition of the raw casting prior to machining. This is used below to better highlight the obvious differences between reference and model. You may also try the included file Pump-bracket-machined-CT-scan.am, for a more relevant comparison between the CAD model and the actual machined part.

The 3D CT volume is displayed in addition to the CAD surface at a different location. You can rotate and move the scene in the 3D viewer window for a closer look.
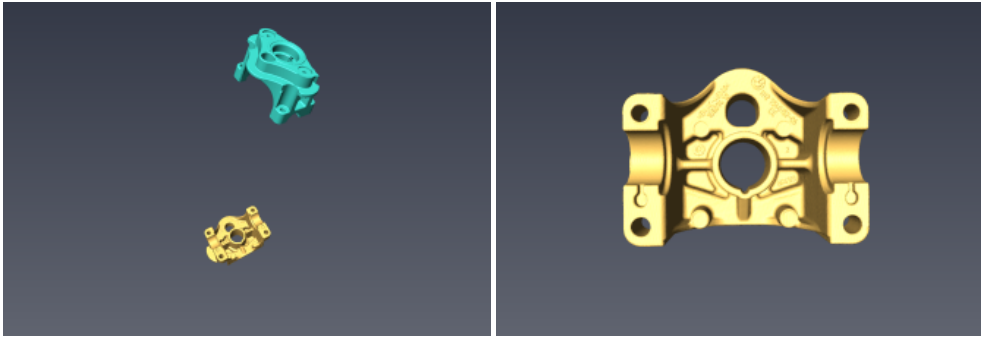
Figure 6.101: Volume data and CAD surface

The 3D volume is displayed using internally an *Isosurface* module with an automatically calibrated threshold to show the object boundaries. See tutorial section 3.4 on Intensity Range Partitioning for more details.

In the *Properties Area*, you could optionally control simplification parameters for faster computation.
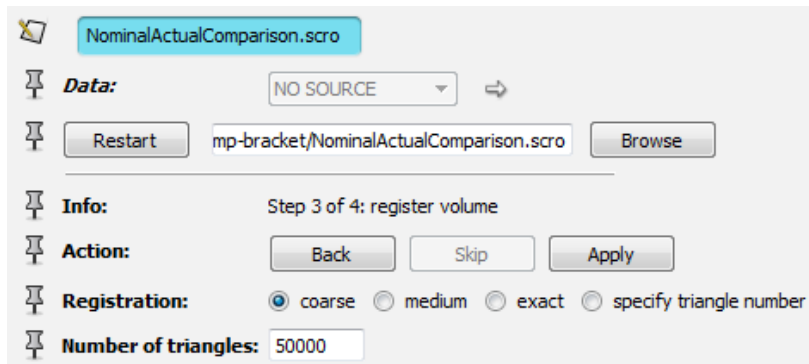
Figure 6.102: Register volume

Let's now proceed to the registration of the scanned part with respect to the reference CAD geometry.

- Leave default options and press Apply.

A model boundary surface is extracted from the 3D volume using the automatic calibration mentioned above. Then a few seconds are required to prepare the surfaces, simplifying the surfaces according to options, and to register the surfaces after automatic principal axis pre-alignment. You can then see the two surfaces overlaid.

Registration of 3D image and surface, nominal-actual analysis
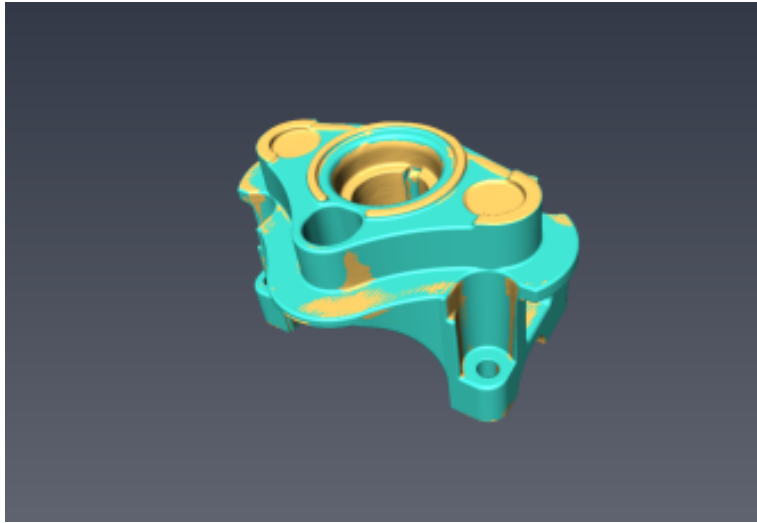
Figure 6.103: Volume and CAD aligned

The last step computes and displays a map of nominal-actual distances. You can optionally define a simplified surface resolution for faster processing.
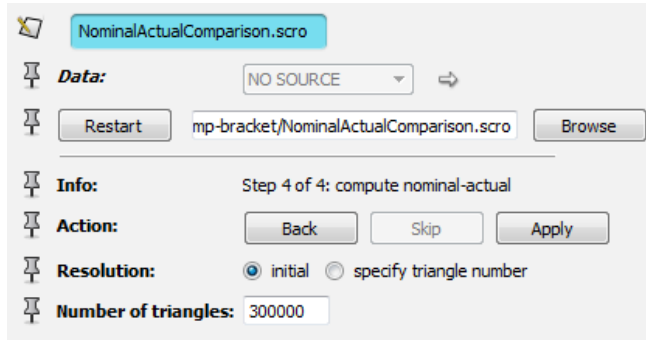
Figure 6.104: Compute nominal-actual difference

- Leave default options and press *Apply*.
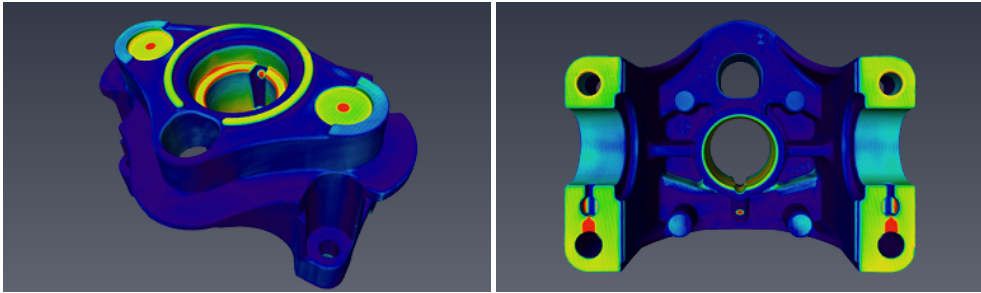
Figure 6.105: Distance between CAD and volume data

6.9.2 Advanced - Image/surface registration step by step
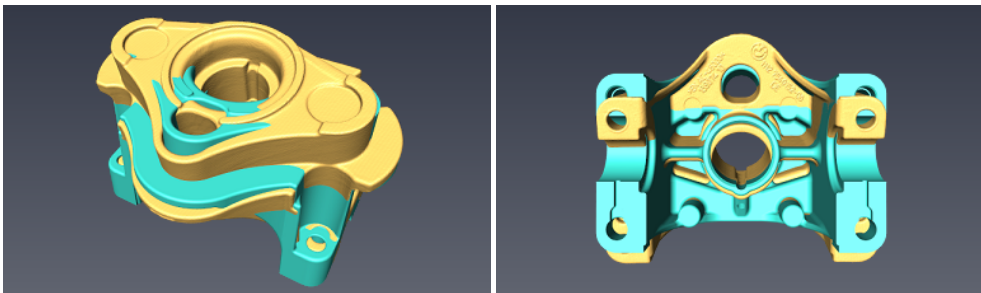
Figure 6.106: Pre-alignment with principal axes registration

We outline here two approaches available in the current version of Avizo for aligning a 3D image and a 3D surface. For more details, please refer to sections about Registration of 3D Images and Registration of 3D Surfaces.

Approach 1: Align a surface generated from the 3D image with the reference surface.

1. Create a surface from the image using *Isosurface* and *Extract Surface*, or *Generate Surface* from segmented image.
2. Use *Align Surfaces* module for pre-alignment and refined registration.

This method is the most robust with respect to possible topological inconsistencies in surfaces imported from the CAD model, i.e., holes. This is the approach used in the example above. Advanced users familiar with scripting may look at the *NominalActualComparison* wizard as an example of a workflow-driven script module.

Approach 2: Register the 3D image with another 3D image generated from the reference surface

1. Turn the surface into a 3D image using *Convert / Scan Surface To Image*. The input triangulated surface is then required to be watertight (have no holes).
2. Use the *Register Images* module or the *Image Registration Wizard* for pre-alignment and refined registration.

This method can be effective for fast or repeated registration of 3D images with a single reference.

Chapter 7

Animations and Movies

The tutorials in this chapter introduce the following topics:

- *Creating animations* - creating animations using the Animation Director
- *Creating movie files* - using the Movie Maker module

7.1 Creating animations

In this tutorial, you will learn how to use the *Animation Director* module for creating an animated sequence of operations within Avizo. In our example, we will visualize a polygon model using effects such as transparency, camera rotation, and clipping to make the visualization more meaningful and attractive.

The tutorial covers the following topics:

- creating an initial project for the demo
- animating an *Ortho Slice* module
- activating additional modules during the demo
- using a camera rotation or path
- removing events that are already defined
- overlaying the inside surfaces with outer surface
- using clipping to add the outer surface gradually
- advanced clipping issues
- inserting breaks and defining demo segments
- using function keys for jumping between demo segments
- defining partial loops within the demo sequence

- storing and replaying a demo sequence

Once you have learned how to define an animated demo sequence, you can further learn how to record the demonstration into a movie file in Section 7.2.

7.1.1 Creating a project

First, we need an Avizo project that contains all the data and modules for the visualization and animation we want to do. In our example, we pick the chocolate bar scan data set `chocolate-bar`. Start by loading *data/tutorials/chocolate-bar.am* from the `AVIZO_ROOT` directory. By right-clicking on the green data icon and selecting from the data set's popup menu, attach a *Bounding Box* module to the data. Use the mouse to navigate around the model in the 3D viewer.
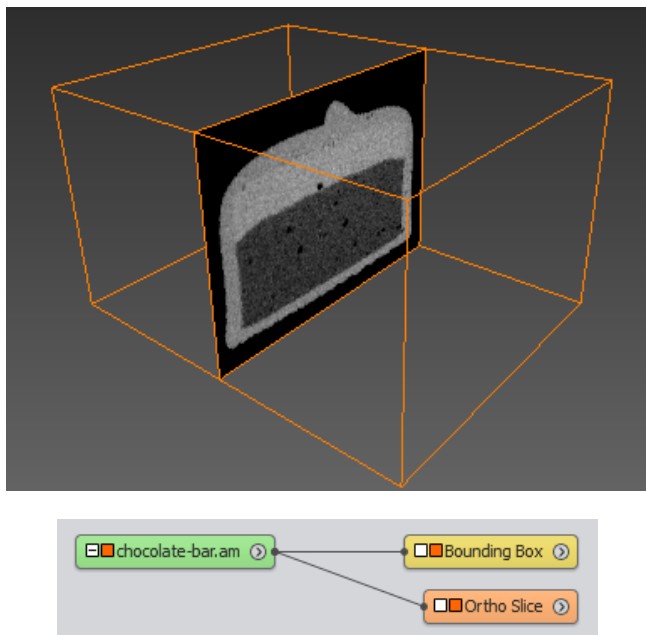
Figure 7.1: Data `chocolate-bar.am` with a Bounding Box and an Ortho Slice

7.1.2 Animating an Ortho Slice module

Let us move the *Ortho Slice* plane up and down to show what the data looks like. Note that the *Ortho Slice* module has a port called *Slice Number*. If you change the value of that slider, you see the plane move in the viewer.

Now let us animate this slider using the *Animation Director* module as our first exercise. From the toolbar, click on the 🎬 *Animation Director* button.

A new widget becomes visible hosting the *Animation Director* user interface.
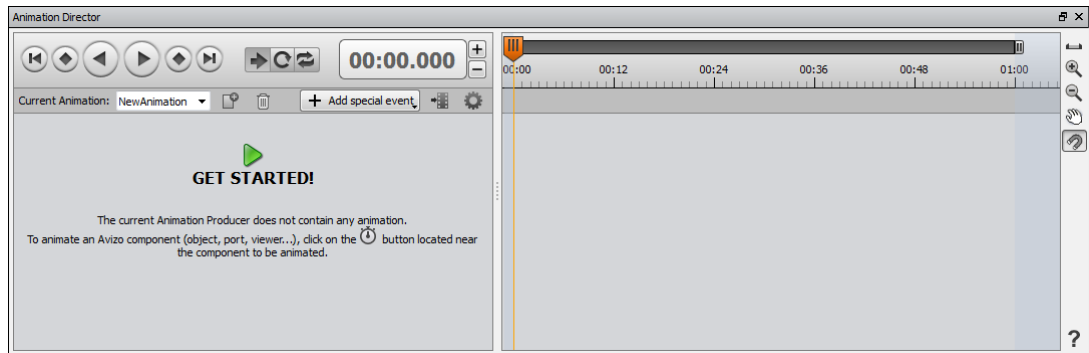
Figure 7.2: *Animation Director* user interface

Like the other widgets, this widget is also dockable and you can place it at a convenient position within the Avizo user interface. After activating the *Animation Director* by clicking on the related button in the toolbar, all ports of the currently available modules that can be animated are extended by an additional button representing a stopwatch ⏱ .
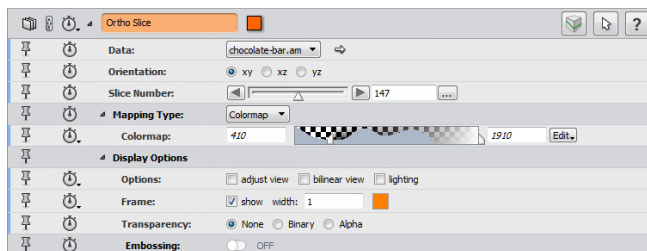
Figure 7.3: Extended ports in the module's properties

Before we start animating the *Ortho Slice'* slice position, let's first have a quick look at some GUI elements of the *Animation Director* widget (shown below in figure: *event and keyframe in timeline*).

As you can see, the user interface is divided into a left and a right panel. On the left panel, you can control the timeline that is displayed on the right panel. The timeline has a main time slider. The name of the current animation can be changed by writing directly into the current animation name field. For example, we can name it "CandyAnimation".

We can now animate the *Ortho Slice'* slice position. We do this by clicking on the stopwatch button of

Creating animations   **358**

the Slice Number port in order to schedule the start event:
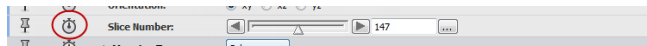
Figure 7.4: extended Slice Number port

Clicking on the stopwatch button creates a new keyframe in the *Animation Director* timeline and the event is listed in the left panel of the user interface. If you hold the mouse cursor above the small orange diamond symbol ◆ in the timeline panel, this will activate a small input field where you can adjust the time and the accompanying value for the port with which it is associated. In order to adjust the schedule, you can simply drag the diamond icon to the desired position on the timeline.
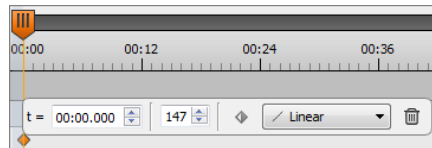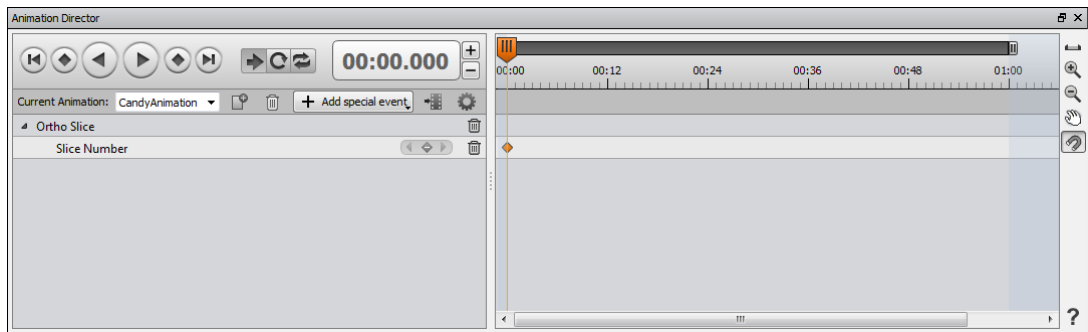
Figure 7.5: event and keyframe in timeline

With this operation, we have defined the beginning of the animation of the slice position. Next we want to define the time where the animation should end. To do this, we drag the master time slider ▼ to the desired time on the timeline, e.g., to 00:04.000, which means 4 seconds. As a next step, we set the slice position of the *Ortho Slice* module by either setting the *Slice Number* port in the properties of the module or by positioning the slice interactively in the viewer window. Using either method, set the value of the *Slice Number* port should be set to 294. After you click the stopwatch button again, the keyframe is created in the timeline:

You can test your first animation by positioning the master time slider back to 00:00, either by clicking on the ⏮ *Jump To Start* button or by entering the desired time into the *Current Time* control
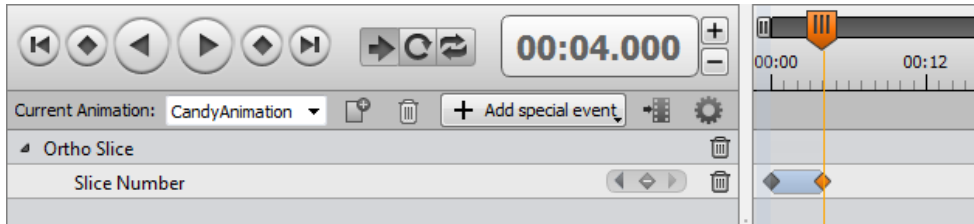
Figure 7.6: start event keyframe and end stop event keyframe in timeline

. Finally, start the animation by clicking on the ⏵ *Play Forward* button or by pressing [F4]. To stop the animation click on the ⏹ *Stop* button or press [F3].

Note: The animation time is not a real physical time. The time that has been specified is the time that Avizo will try to respect, if possible. If an animated property needs a specific amount of time to be processed (computation, viewer redraw, etc.), Avizo will not skip it and will wait until the animation step has finished before continuing. This implies that, for instance, an animation with a time range of 30 seconds won't run for exactly 30 seconds, but will run 30 seconds or longer.

As an exercise, adjust the master time slider to the time 00:12, set the position of the *Slice Number* of the *Ortho Slice* to the value 0, and click the stopwatch button again. This will create another keyframe in the timeline. Restart the animation again as described above. The slice should first move from slice number 147 up to 294 during 4 seconds and then decrease down to the value 0 within the next 8 seconds.

Note: As described above, you can edit the time and the value of single keyframes by hovering the mouse cursor above the keyframe icon and by clicking in its user interface after it has been become visible (hover dialog box). Alternatively, you can modify the time of the keyframe by dragging it on the timeline. In order to change the value of the associated port, you can position the master time slider on the time of the keyframe and then adjust the value of the port by using the user interface in the properties area of the module. In order to shift all keyframes of a port forwards or backwards on the timeline, position the mouse cursor between two subsequent keyframes. You will notice a connecting bar in dark gray and that all keyframes of this sequence are selected (indicated by orange colored diamonds).

You can now shift all selected keyframes forwards or backwards by dragging the dark gray bar.

7.1.3   Activating a module in the viewer window

Next, let us add a visualization of the internal structure in the data set after we have moved the *Ortho Slice*. Load the data set *data/tutorials/chocolate-bar.surf* in addition to the current project. Attach a *Surface View* module to it. Click on the yellow *Surface View* module to see its user interface. Select

Creating animations   360
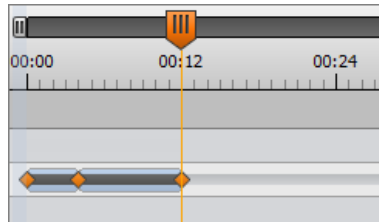
Figure 7.7: shifting subsequent keyframes

Chocolate and *All* in the *Materials* port and press the *Remove* button in the buffer port. This will visualize the inside surfaces of the model.
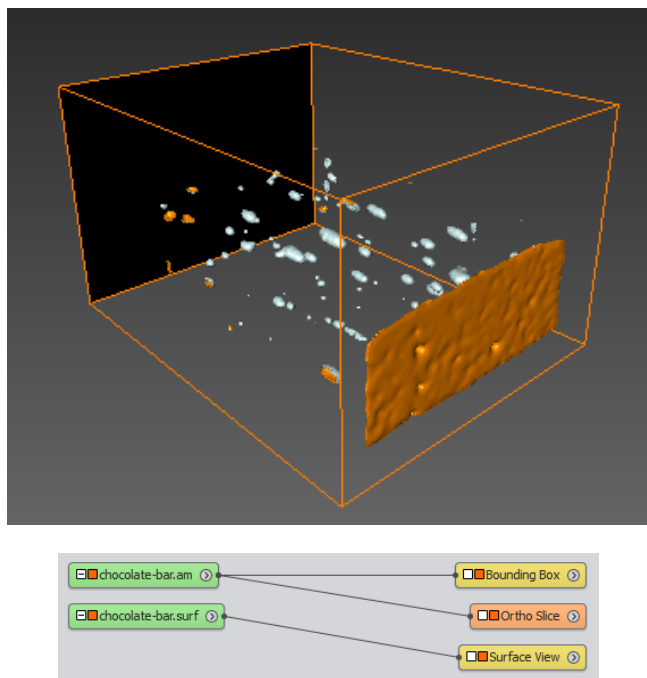
Figure 7.8: Viewer and Network when you visualize Ortho Slice and Surface View modules

If you want to switch the surface visualization on and off manually, you would use the *viewer toggle* (orange rectangle) of the *Surface View* module. If you want to include this action in your animation sequence, you need to do the following:

- Click on the *Surface View* module to select it.

- Adjust the master time slider of the *Animation Director* to, e.g., 00:12.
- Click on the top most stopwatch icon in the properties area of the *Surface View* module.
- Select *Visibility in Viewer 0* from the drop-down menu.
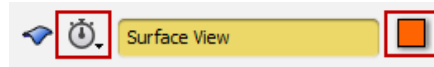
Figure 7.9: Surface View stopwatch and toggle buttons

To test the newly added event, set the master time slider of the *Animation Director* back to the beginning by using one of the methods described above. As you have may noticed, another event has been automatically added at time 00:00 to consider the surface model as invisible by default. Start the animation. As before, the slice moves back and forth. When the master time slider reaches 00:12, the surface model is switched on.

Note: You can change the time when the *Surface View* becomes visible by simply dragging the appropriate keyframe diamond on the timeline.

7.1.4   Using a camera rotation

To look at the 3D model from all sides, let's add a camera rotation to our demo sequence. From the *Project / Create Object...* menu, select *Animations And Scripts / Camera Orbit*. Try the rotation by playing the time slider in the *Camera Orbit* module. If you do not like the axis of rotation, reset the time slider to 0, navigate to a good starting view in the viewer window, and click on *recompute* in the *Camera Orbit* module. Note that the values of the *Camera Orbit* time slider range from 0 to 360.

Once you are satisfied with the camera rotation, add it to the timeline:

- Set the master time slider to the time where the camera rotation should begin, e.g., 00:10.
- Select *Camera Orbit* module.
- Click on the stopwatch button next to the *Time* port of the *Camera Orbit* module and click on *Time: Value*.
- Set the master time slider to the time where the camera rotation should end, e.g., 00:14.
- Adjust the *Time* port of the *Camera Orbit* module to its maximum (360).
- Click on the stopwatch button of the *Time* port again.

Now play the demo to see the result. After moving the slice and switching on the surface model, the view is rotated so that the surface can be seen from all sides.

7.1.5 Removing one or more events

You can remove events by hovering the mouse cursor over the desired keyframe. When the keyframe user interface becomes visible, you can click on the trashcan 🗑 to remove the keyframe.

To remove all keyframes that are associated with a module's port, you can click on the trashcan 🗑 that is located in the same line as the port name on the left panel.

7.1.6 Overlaying the inside surfaces with outer surface

Now we want to show the outer surface overlaid over the inside model.

- Attach a second *Surface View* module to the *chocolate-bar.surf* data set. Since *Exterior* and *All* are selected as the default materials, this brings up the exterior surface.
- Click on the second *Surface View* module. It should be called *Surface View 2*.
- Select *transparent* from the *Draw Style* port.
- Inside the *Buffer* port, select *Chocolate* and *All* in the *Materials* port and click *Add*.
- It will be helpful to show the inside surfaces underneath the exterior surface, so jump to time step 00:14 or later in the *Animation Director* module.
- Adjust the grade of transparency using the *BaseTrans* slider in *Surface View 2* (for instance 0.7).
- Smooth out the outer surface by clicking on *more options* in the *Draw Style* port and selecting *Vertex normals*.

Like we did with the inside surfaces model, we can switch on the outer surface model at some point in the animation sequence:

- Set the master time slider to the time where our surface should become visible, e.g., 00:14.
- Click on the top most stopwatch icon in the properties area of the *Surface View 2* module.
- Select *Visibility in Viewer 0* from the drop-down menu.

Again, check out the results by playing the animation sequence.

7.1.7 Using clipping to add the outer surface gradually

Instead of just switching the outer surface on at one point, we can make it appear gradually over the inside surfaces from top to bottom. In order to do so, we use the *Ortho Slice* plane to *clip* the outer model, and then move the *Ortho Slice* plane down.

- Move the master time slider to the time that you selected to make the *Surface View 2* module visible (e.g., 00:14).
- Click on the *Surface View 2* module.
- Click on the top most stopwatch icon in the properties area of the *Surface View 2* module.
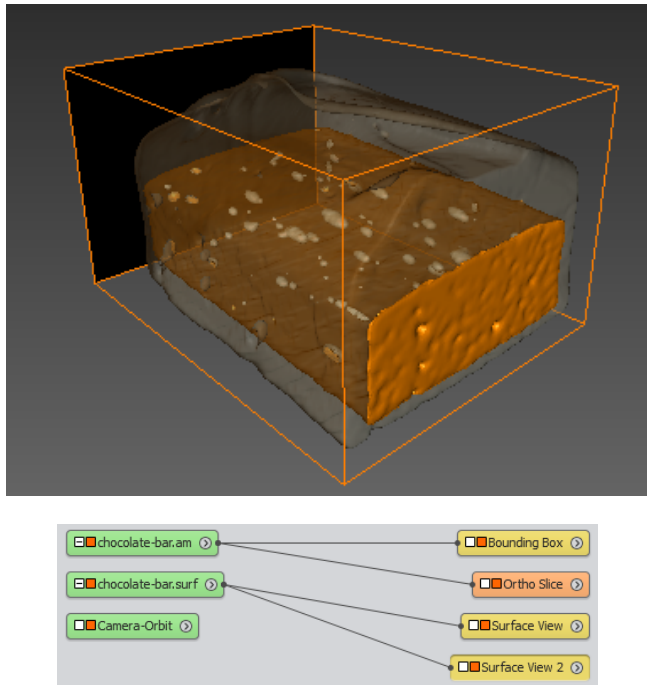
Figure 7.10: Viewer and Network with an Ortho Slice and two Surface Views

- Select *Clip using Ortho Slice* from the drop-down list.
- Move the mouse cursor over the newly created keyframe in the timeline and wait until the user interface for this keyframe has become visibile.
- In the keyframe's user interface, select the radiobox *on* to enable *clipping using Ortho Slice*.

In order to make the outer surface visible, we finally must animate the *Ortho Slice* towards the bottom of the scene:

- Set the master time slider to 00:12.
- Select the *Ortho Slice* module in the *Project View*.
- Click on the small stopwatch icon of the port *Slice Number* to set a new keyframe in the timeline.
- Move the master time slider to the time where you want the animation to be finished, e.g., 00:22.
- Move the *Slice Number* value to the value 294.
- Click again on the small stopwatch icon of the *Slice Number* port.

Start the animation either from the beginning or from 00:14 and watch the result.
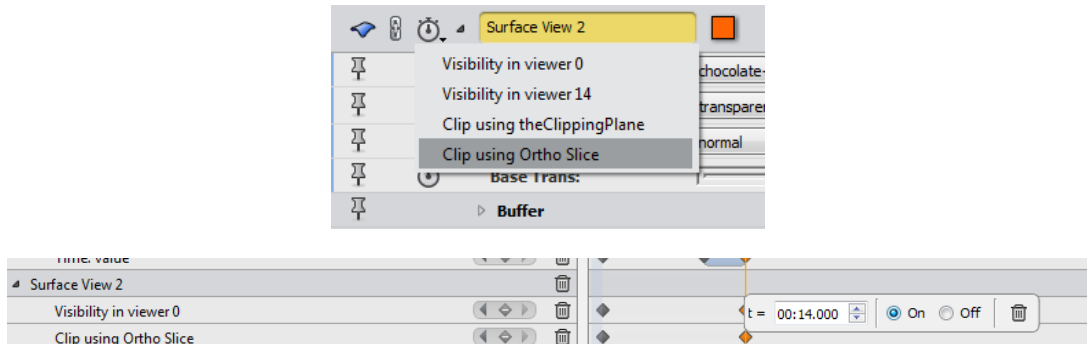
Figure 7.11: selecting *Clip using Ortho Slice* and enabling the clipping in the timeline

As a last step, you might want to rotate the view again while the outer surface is appearing. You can simply reuse the old camera rotation during a second time range:

- Set the master time slider to the time where you started to make the outer surface visible, e.g., 00:14.
- Select the *Camera-Orbit* module.
- Click on the stopwatch icon of the port *Time* and select *Time: value*. This will insert a new keyframe to the timeline.
- Move the master time slider to the time where you want the camera rotation to be finished e.g., 00:22.
- Set the time value of the port *Time* to 0.
- Click again in the stopwatch icon of the *Time* port and select *Time: value*.

To avoid the Ortho Slice to hide the surface view when displaying the slice 294, you can change its *Transparency* mode during the animation.

- Move the master time slider to the time where you want to change the *Transparency* value, e.g., 00:12.
- Set the *Transparency* port of *Ortho Slice* to *Binary*, then click on the stopwatch icon of *Transparency* port. This will insert a new keyframe to the timeline. The *Transparency* port is in the *Display Options* port of *Ortho Slice*.
- Move the master time slider to 00:00, and set the Ortho Slice Transparency to "None" (See Figure 7.12).

Start the animation either from the beginning or from 00:14 and watch the result.
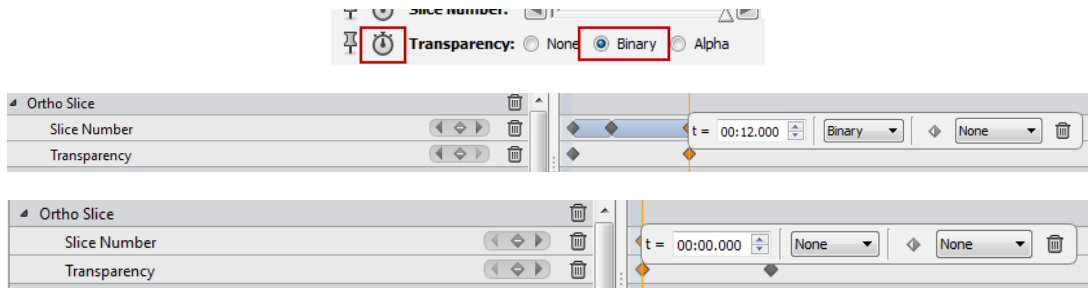
Figure 7.12: Changing the Ortho Slice *Transparency* mode

7.1.8 More comments on clipping

Clipping can sometimes be a little bit more complicated than in our example, because clipping can be applied to a plane in two different orientations. This means that you can either clip away everything *above* the plane, or *below* the plane. Unfortunately, it is not always obvious which of the two cases you are in.

However, you can simply invert the orientation of the clipping in *Animation Director*. In our example, you would simply set the master time slider to a time prior to the actual clipping. Next, you need to select the module that will do the clipping, the *Ortho Slice* module in our case. Now, click on the topmost stopwatch icon in the *Ortho Slice* properties area and select *Invert clipping orientation* from the drop-down list.
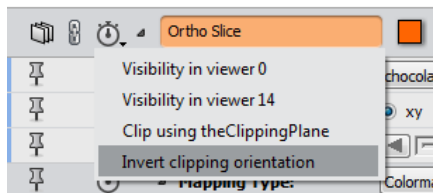
Figure 7.13: How simply invert the orientation of the clipping in *Animation Director*

You do not need to use an *Ortho Slice* module to do clipping. As you may have observed, the *Ortho Slice* might occlude parts of what you want to show. In that case, it is better to create an empty *Clipping Plane* module by selecting *Other / Clipping Plane* from the *Project / Create Object...* menu. Attach the module to the data set you want to clip (e.g., to *chocolate-bar.surf* in our example), and then use the *Clipping Plane* for clipping just as you used the *Ortho Slice* before.

7.1.9 Breaks and function keys

The animation sequence that we have created in this tutorial automatically runs through the complete time range that we defined, one minute by default. Sometimes it may be desirable to split the animation into several segments, so that the animation will stop at designated points and can be continued when the user desires.

To take this into account, you can insert *breaks* in the *Animation Director* timeline. Let us insert one such break right after the inside model appears:

- Set the master time slider to the time when the *Surface View* appears, e.g., 00:12.
- In the left panel of the *Animation Director* click on the + *Add special event* button.
- Select 🏠 *Add Break* from the drop-down list. A new keyframe is created on top of the timeline indicating the *Break*.
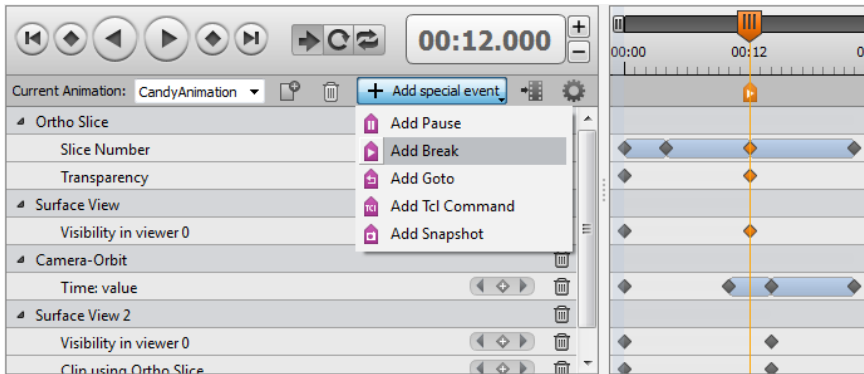
Figure 7.14: Add break in a *Animation Director*

This way the animation will stop at time 00:12, which is right when the model is switched on. When you play the animation from the start, you will notice that after the inner structure is switched on, the animation will stop.

Let us insert a second break at time step 00:14, which is right before the outer surface is starting to show. Proceed as above, using a trigger time of 00:14 instead of 00:12.

If you run the animation from the very beginning, it will stop after the inside surfaces are displayed. Click on the *Play Forward* button or press [F4] to continue the animation.

Try this by pressing the function key [F4]. The demo continues.

Likewise, the animation will stop just before showing the outer surface. Again, you can continue the demo by pressing [F4]. In general, at any point while the animation is running, you can press the [F3] key to stop it manually. Pressing [F4] will continue from the point where the animation stopped.

If you have defined breaks as we did above, there are two additional function keys that in a sense allow you to *navigate* through the animation segments: pressing [F9] will jump back to the previous break or to the very beginning of the animation, and [F10] will jump to the next break, or to the very end of the animation.

Note: If you use [F9/F10], it will just jump, and you need to press [F4] to start playing it from the new time step.

Please note that you can disable the breaks by checking the *Skip Break* toggle in the *More Options* panel of the *Animation Director* module. You may even disable the definition of function keys by checking the *No* toggle in the *Activate Function Keys* port:
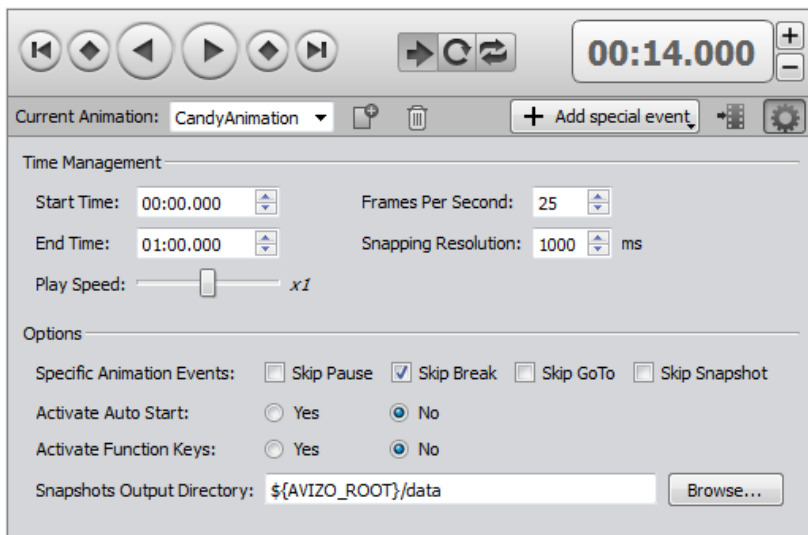
Figure 7.15: Disabling *Breaks* and *Function Keys*

7.1.10   Loops and Goto

One more feature that might be required for certain kinds of animation is the definition of loops. If you just want the whole demo to run in a loop, you can do this easily using the built-in features of the *Animation Director* module:

You can toggle the appropriate button ➡ *Play Once*, ↻ *Play Loop*, or ⇄ *Play Swing* located right between the main *VCR button* like *Play,* and the main *time control*:

Now if you play the animation, it will play from beginning to end once (Play Once), start over from the beginning (Play Loop), or play forwards, backwards, and so on (Play Swing).

However, you may want to define some part of the demo to run in a loop, and then stop the loop

Figure 7.16: Loop options: *Play Once, Play Loop, or Play Swing*

and continue with the animation upon key press. You can easily do this with the *Goto* feature of the *Animation Director* module:

- In the *Animation Director* module adjust the main time control to the time 00:11:950.
- Click on the + *Add special event* button.
- Select ⬚ *Add Goto* from the drop-down list. A new keyframe is created on top of the timeline.
- Move the mouse cursor right over the new created keyframe icon and a user interface for this *Goto* will appear.
- In the *Time To Jump To* field enter the time where you want to go from here, e.g., 00:04:000 in this case.
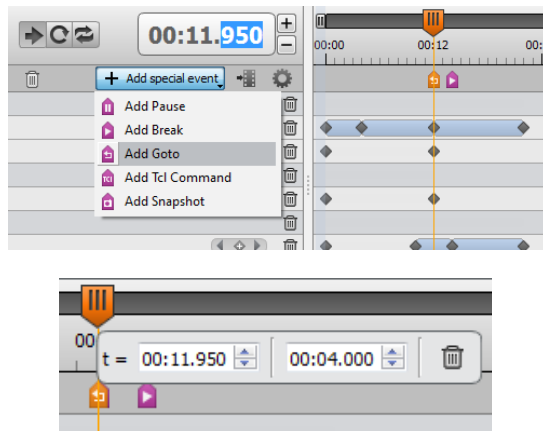
Figure 7.17: *Goto*, jump to user-specified time

When you run the animation sequence now, it will loop in the segment between time 00:04 and 00:12, only showing the *Ortho Slice* move up, jump down, move up again and so on... You can stop this by clicking on the stop button of the *Animation Director* control panel or by pressing [F3]. To continue after the loop, you need to jump to the next segment by pressing [F10], and then start playing again by pressing [F4].

7.1.11   Storing and replaying the animation sequence

As you may have noticed by now, storing an animation sequence once you have defined it is quite easy: simply save the whole Avizo project by selecting *File/Save Project* from the menu. The *Animation Director* module will be saved along with the project, and so will be the animation sequence you have defined.

When you load the project back into Avizo, the state of the project will be the same as it was when you saved it. This means that you should be careful to reset the *Animation Director* master time slider to 00:00 before saving the project if you want the demo to start from the beginning.

After loading the project, you can start the demo by clicking on the play button of the *Animation Director* module, or by pressing [F4]. If you want to run the demo automatically right after the project is loaded, you can use the *auto start* feature that you find when you check *Activate Auto Start* in the *More Options* panel.
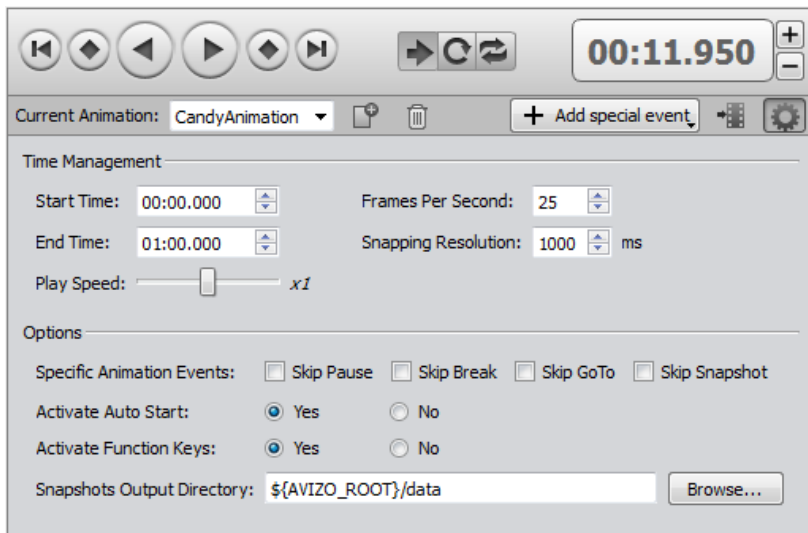
Figure 7.18: Enable auto start mode

Just activate the *Activate Auto Start* functionality by checking the *Yes* radio button and save the project. When you load it again, the demo will start running automatically.

7.2   Creating movie files

In this tutorial, you will learn how to record a self-created animated sequence into a movie file using the *Movie Maker* module.

In our first example, we will just use a camera path to animate the scene, whereas in our second example, we will rely on the animated demonstration created in Section 7.1.

7.2.1 Attaching Movie Maker to a Camera-Path

If you have created a visualization of your data and want to create a movie showing this visualization from all sides or from certain interesting viewpoints, you can create an appropriate camera path and record a movie by following the camera along that path.

Let us create a simple example. Load the *chocolate-bar.am* data set from the *tutorial* subdirectory and attach an *Isosurface* module to it. Choose an isosurface threshold of 410 and press the *Apply* button.
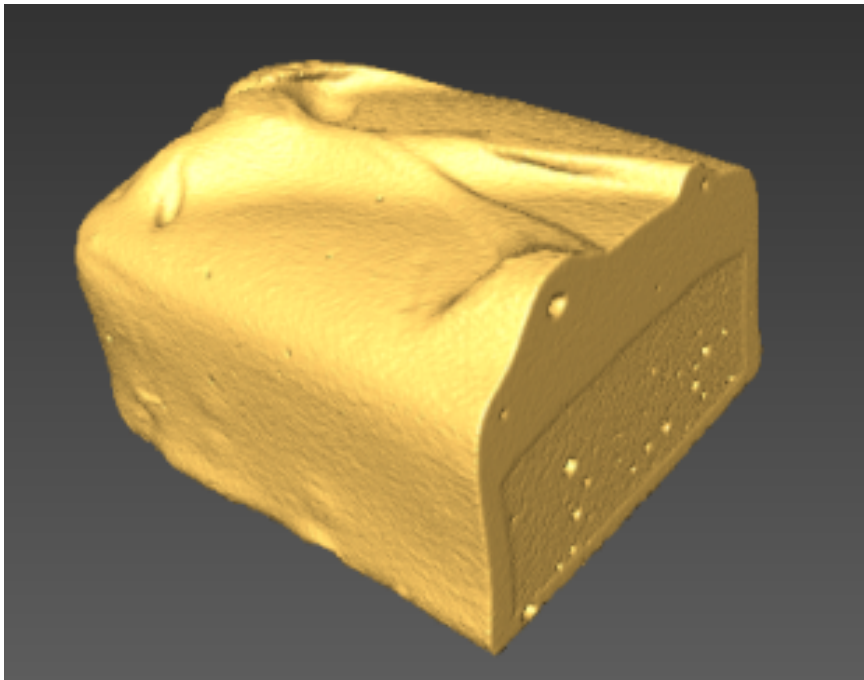
Figure 7.19: Visualize chocolate-bar.am with an *Isosurface*

The easiest way to create a simple camera path is to use the *Camera Orbit* module. From the *Project / Create Object...* menu, select *Animations And Scripts / Camera Orbit* and press the play button of the newly created module. You can watch the scene rotate in the viewer while the time slider is playing.

To record an animated scene into a movie file, you need to attach a *Movie Maker* module to a module that possesses a *time slider port*. The movie is recorded by going through the individual time steps and taking snapshots of the viewer along the way.

In our example, the *Camera Orbit* module has a time slider, so we can attach a *Movie Maker* module to it by right-clicking on the *Camera Orbit* icon in the Project View and selecting *Movie Maker* from the popup menu.
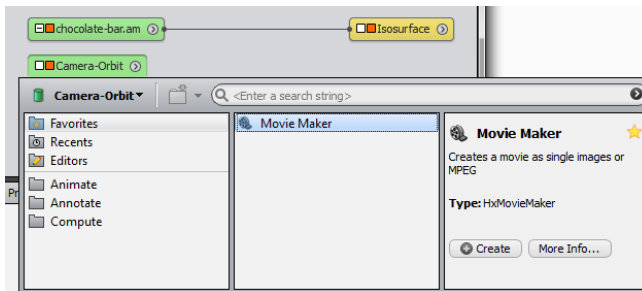
Figure 7.20: Attach a *Movie Maker* to *Camera Orbit*
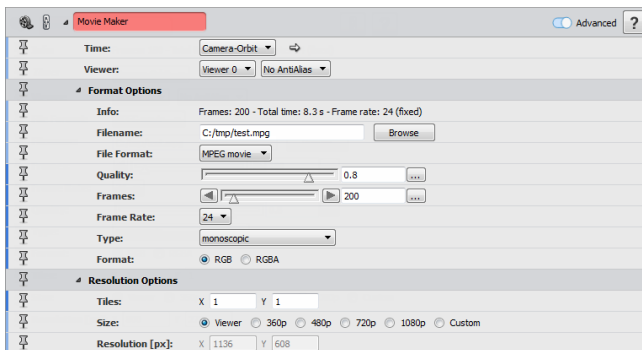
Figure 7.21: Parameters of *Movie Maker* module

In the *Movie Maker* module, first click on the *Browse* button in the *Filename* port and enter a movie file name like `C:/tmp/test.mpg`. The `.mpg` suffix suggests that the movie file format will be MPEG, which is a widely accepted standard format for digital movies achieving a good compression ratio.

Next, adjust the parameters of the *Movie Maker* module to your liking, e.g., change the *number of frames*, the *image size*, or the *compression quality*. Please refer to the *Movie Maker* documentation for details.

In our example, let us choose 180 frames and leave all other parameters untouched. Since the *Camera Orbit* module does a full rotation of 360 degrees, each of the 180 frames will represent a rotation of two degrees with respect to the previous frame. Press the *Apply* button to start recording.

Wait for some time while the *Movie Maker* module drives the *Camera Orbit* module and accumulates the snapshots. **Please note that the speed during the recording process is different than the play-**

back speed of the movie. Now view the resulting movie file `test.mpg` with a movie player of your choice (e.g., Windows Media Player or a similar tool). Experiment with the recording parameters until you get the desired result (e.g., control the file size and image quality by changing the *Compression quality* value, choose different image sizes to see up to which image size your computer is capable of smoothly displaying the movie, and change the number of frames to control the speed of the rotation).

7.2.2 Creating a movie from an animated demonstration

Now we try to record a movie of a more complex animated scene. To this end, we load one of the projects that we have created in in Section 7.1 (Creating animated demonstrations).

As you might remember, the basic idea of the *Animation Director* module was that you define a set of events to be executed on a certain time line. Check this out by clicking the ▶ *play* button of the *Animation Director* module. You should see a nicely animated demonstration.

To create a movie from an animation defined with the *Animation Director*, simply click on the ⊞ *Movie Creation* button of the *Animation Director* panel. The following panel will appear:
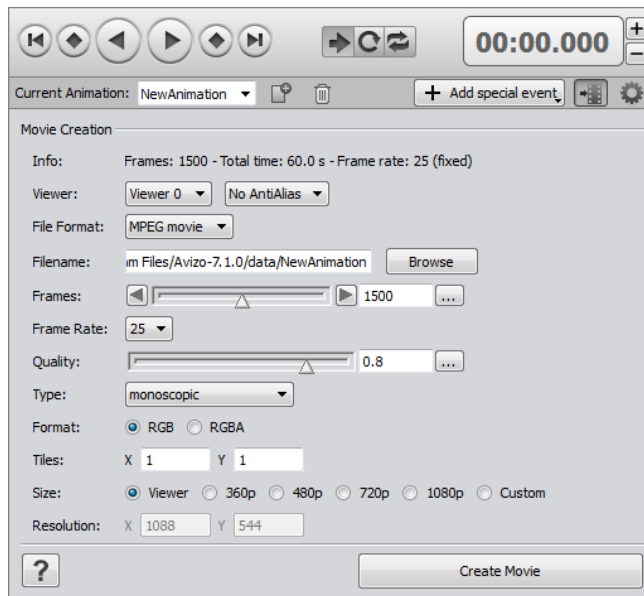
Figure 7.22: Movie creation parameters of the *Animation Director* module

The *Animation Director* module internally uses a *Movie Maker* module to create movies. This module is already pre-configured to create a movie that respects the animation settings (duration, frame rate, filename...) that are defined by the *Animation Director* module. However, you can adjust these

parameters, if needed. Just click on the *Create Movie* button to generate the movie.

Chapter 8

User Interface Components, General Concepts, Start-Up

This chapter contains a description of Avizo interface components, data types, general concepts and start-up options. No in-depth knowledge of Avizo is required to understand the following sections, but it is a good idea to have a look at one of the tutorials contained in Chapter 1.6 (First steps in Avizo), particularly the very first one described in Section 2 (Getting Started).

8.1 User Interface Components

The following interface components are described in this section:

- *File Menu*, *Edit Menu*, *Project Menu*, *View Menu*, *Window Menu*, *Help Menu*
- *Standard Toolbar*, *Workrooms Toolbar*
- *Project View*, *Properties Area*, *Progress Bar*, *Viewer Window*, *Console Window*, *Online Help*, *Histogram Panel*, *Correlation Panel*
- *File Dialog*, *Job Dialog*, *Preferences Dialog*, *Snapshot Dialog*, *System Information*
- *Object Popup*, *Create Object Popup*

8.1.1 File Menu

The file menu lets you load and save data objects as well as Avizo Project scripts. In addition, it gives you access to recent files and projects and allows you to quit the program. In the following text, all menu entries are discussed separately.

8.1.1.1   Open Data

The *Open Data* button activates Avizo's *file dialog* and lets you import data sets stored in a file. Most file formats supported by Avizo will be recognized automatically via the file header or the file name extension. If you try to load a file for which the format couldn't be detected automatically, an additional dialog pops up asking you to select the format manually.

A list of all *supported file formats* is contained in the reference manual. Hints on how to import your own data sets are given in Section 2.6.

If you select multiple files in the file dialog, all of them will be loaded, provided all of them are stored in the same format. 2D images stored in separate files usually will be combined into a single 3D data object. On the other hand, there are some file formats for which multiple data objects will be created. Finally, you can also import and execute Avizo project scripts using the *Open* button.

8.1.1.2   Open Data As

The format of input data can be forced using *Open Data As*. A format selection dialog will be opened, allowing you to choose between all of the supported Avizo file formats. All selected files will be treated as being in the specified format.

8.1.1.3   Open Time Series Data

This button also activates the *file dialog*, but in contrast to the ordinary *Open* option it is assumed that all selected files represent different time steps of a single data object. When loading such a time series, an instance of a *Time Series Control* module is created. This module provides a time slider allowing you to adjust the current time step. Whenever a new time step is selected, the corresponding data file is read, and data objects associated with a previous time step are replaced. The module also provides a cache so that the data files only need to be read once, provided the cache is large enough.

8.1.1.4   Open Time Series Data As

This menu entry works the same way as *Open Time Series Data* except that the format of the input data can be forced (same as *Open Data As*). A format selection dialog will be opened, allowing you to choose between all supported Avizo file formats. All selected files will be treated as being in the specified format.

8.1.1.5   Save Data

The *Save Data* button allows you to save a single modified data object again using the same filename previously chosen under *Save Data As*. The button will only be active if the data object to be saved is selected and if this data object has already been saved using *Save Data As*. A common application of the *Save* button is to store intermediate results during manual segmentation in Avizo's *Segmentation Editor*.

8.1.1.6 Save Data As

This button allows you to write a data object into a file with a native file format. To do so you must first select the data object (click on the corresponding green data icon). Then choose *Save Data As* to activate Avizo's *file dialog*. The file dialog presents a list of all native file formats suitable for saving that data object. Choose the one you like and press *OK*. Note that you must specify the complete file name including the suffix. Avizo will not automatically add a suffix to the file name. However, it will update the suffix whenever you select a new format from the file format list. Also, Avizo will ask you before it overwrites an existing file.

If no native file format has been registered at all for a certain type of data object, the *Save Data As* button will be disabled. It will also be disabled if more than one data object is selected in the Project View.

A native file format is a file format that can save all data properties in Avizo. Actually, there are only two native file formats: Avizo Data Format and HxSurface.

8.1.1.7 Export Data As

This button does the same thing as the *Save Data As* button, however, the file dialog presents a list of all file formats suitable for saving that data object, native or not.

Some file formats create multiple files for a single data object. For example, each slice of a 3D image data set might be saved as a separate raster file. In this case, you can add to the file name a sequence of hashmarks (for instance `<filename>####.jpg`). This sequence will be replaced by consecutive numbers formatted with leading zeros.

If no file format has been registered at all for a certain type of data object, the *Export Data As* button will be disabled. It will also be disabled if more than one data object is selected in the Project View.

8.1.1.8 Convert To Large Data Format

When selecting this menu entry, a *Convert To Large Data Format* object will be instantiated and added to the Project View. This module allows you to convert large image data to *LDA file format*.

8.1.1.9 New Project

This button does a *Remove all objects* so that you can start building a new project in the Project View. It also sets the new project name to "Untitled.hx".

8.1.1.10 Open Project

The *Open Project* button activates Avizo's *file dialog* and lets you load a project stored in a file. Project files show up in the dialog as being in the format "Avizo Project". A *Remove all objects* will be done before the new project is loaded so that effectively the new project replaces the old project in the Project View. Currently, only files with extension *.hx* can be opened.

8.1.1.11   Save Project

This button allows you to save the complete project of icons and connections shown in the Project View. The first time a project is saved in Avizo, a Save Project policy dialog box is displayed (see Figure 8.1). This dialog box suggests different policies and a short description of each of them. More details about the related choices can be found in the *General Tab* section. The chosen policy will apply for the project being saved only, unless the checkbox is selected. If the checkbox is selected, the policy applying to the project being saved will apply to all future projects. This setting can be modified later in the *General Tab* of the Preferences dialog box.
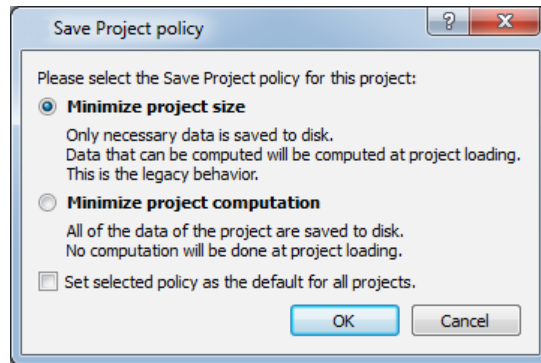
Figure 8.1: Save Project policy dialog

If the project has not been saved previously, you will need to specify the name of an Avizo Project script in the file dialog box. When executed, the project script restores all data objects and modules as well as the current object transformations and camera settings. This feature is useful for resuming work from a point where you left off previously.

Note that usually all data objects must have been stored in a file to be able to save the project. If this is not the case, a dialog box is shown listing all the data objects that still need to be saved. In the dialog box, you can specify that all required data objects should be saved automatically in a separate subdirectory.

Instead of selecting the option *Avizo Project*, you can also choose *Avizo Project and data files (pack & go)* from the file dialog box's format menu. In this situation *all* data objects currently loaded will be saved in a separate directory. More options affecting the export of project scripts can be adjusted in the *Preferences dialog box*.

8.1.1.12   Save Project As

This button works like the *Save Project* button except that, in this case, you will always need to specify the name of an Avizo Project script in the file dialog.

8.1.1.13 Save Project As Template

This button allows you to save the current Project View as a template project. A template project consists of a backup of an original project that can be replicated on different data of the same type. See Section 9.1.1 Template Projects Description.

8.1.1.14 Recent Files

This button can be used to load recently used files. When choosing this menu entry a submenu appears listing the five most recent files. If multiple 2D images have been loaded this is indicated with the name of the first file followed by an ellipsis (...). The number of files displayed in the most recent files list can be modified in the *General* tab of the *Preferences Dialog*.

8.1.1.15 Recent Projects

This button can be used to load recently used project scripts. When choosing this menu entry, a submenu appears listing the five most recent project scripts. The number of projects displayed in the most recent projects list can be modified in the *General* tab of the *Preferences Dialog*.

8.1.1.16 Quit

This button terminates Avizo and prompts you to save the current project configuration, if there are unsaved changes.

8.1.2 Edit Menu

The *Edit* menu provides access to the standard cut/copy/paste/delete commands, as well as to Avizo's dialogs: *Preferences Dialog*, *Job Dialog* and an extended version of the *Parameter Editor*.

8.1.2.1 Cut

In the *Console*, this command cuts selected text and copies it to the clipboard. In the *Project View*, this command removes the selected objects.

8.1.2.2 Copy

In the *Console*, this command copies the selected text to the clipboard. In the *Project View*, this command copies the selected objects.

8.1.2.3 Paste

In the *Console*, this command pastes the text in the clipboard to the current text insertion point. In the *Project View*, this command creates cut or copied objects.

8.1.2.4 Delete

In the *Console*, this command deletes the selected text. In the *Project View*, this command deletes the selected objects.

8.1.2.5 Select All

In the *Console*, this command selects all the text. In the *Project View*, this command selects all objects.

8.1.2.6 Preferences

This option opens the Avizo*Preferences Dialog*. This dialog allows you to adjust certain global settings of Avizo like the Project View options, the user interface layout, how project scripts are exported, segmentation, rendering and performance options...

8.1.2.7 Dialogs

This menu item comprises the dialogs *Database* and *Jobs*.

Database

The *Database* button activates an extended version of the *Data Parameter Editor*, allowing you to manipulate Avizo's global parameter database. Among others, the parameter database contains a set of predefined materials (to be used for image segmentation and surface reconstruction) and of predefined boundary ids (to be used for surface editing and FEM pre-processing). For example, for each material and for each boundary id, a default color can be defined in the database.

Modification, insertion, and removal of parameters is performed in the same way as in the ordinary parameter editor. In addition, the extended parameter dialog provides a menu bar allowing you to load, import, save, or search the global parameter database. Avizo's default database is stored in the file `share/materials/database_avizo.hm` located in the directory where Avizo was installed. Use the *Database* menu option *Set default file* to specify that a different database file be used instead. This change is permanent, i.e., it takes effect also if Avizo is restarted. To switch back to the system default, use the *Database* menu option *Use system default file*.

Jobs

This button brings up Avizo's *Job Dialog* which is used to control the execution of batch jobs running in the background. For example, tetrahedral grids can be generated in a batch job (see module *Generate Tetra Grid*). However, for most users the batch queue will be of minor interest.

8.1.3 Project Menu

The *Project* menu provides control over the visibility of object icons and lets you delete or duplicate objects. Depending on how many icons are selected in the Project View, some menu options might be disabled.

Note: in "Tree View" mode, the *Project* menu is identical except that the *Show Object*, *Show All Objects*, and *Hide Object* items are not available.

8.1.3.1 Graph View

This toggle allows you to select the *Graph View* as current *Project View*.

8.1.3.2 Tree View

This toggle allows you to select the *Tree View* as current *Project View*.

8.1.3.3 Hide Object

The *Hide Object* button hides all currently selected objects. The object's icons are removed from the Project View but the objects themselves are retained. You get the same effect by pressing the `Ctrl-H` key. Hidden objects can be made visible again using *Show Object* or *Show All Objects*. This option will be unavailable if the current Project View is the *Tree View*.

8.1.3.4 Remove Object

The *Remove Object* button deletes all selected objects and removes the corresponding icons from the Project View. You can get the same effect by pressing the `Delete` key. If you want to reuse a data object later on, be sure to save it in a file before deleting it. If a data object has been modified but has not yet been saved to a file, it is marked by a little asterisk in the object icon. In the *Preferences Dialog*, you can choose whether a warning dialog should be printed if you try to delete unsaved data objects which cannot be recomputed by an up-stream compute module. If you delete a data object, all connected modules will be deleted as well. However, if you delete a module, connected data objects (e.g., the results of a compute module) will be retained.

8.1.3.5 Duplicate Object

The *Duplicate Object* button creates copies of all selected data objects. For each copy, a new data icon is put in the Project View. The name of a duplicated data object differs from the original one by one or more appended digits. The duplicate option is not available if you have selected icons that do not represent data objects (e.g., display or compute modules).

8.1.3.6 Rename Object

This button allows you to change the name of a selected object. In *Graph View*, a small dialog box will be popped up when the button is pressed. In *Tree View*, you can directly edit the object item in the view. The button is disabled if no object is selected or if multiple objects re selected. Note that two objects in Avizo can't have the same name. Therefore, the name entered in the dialog may be modified

by appending digits to it, if necessary. You can also rename an object by pressing the F2 key when it is selected or by double clicking on the item in *Tree View* mode.

8.1.3.7   Hide All From Viewer But This

This button allows you to keep visible in the viewer only the display modules of the selected objects.

8.1.3.8   Create Object...

This button allows you to display the *Create Object* popup in order to create modules or data objects that cannot be accessed via the popup menu of any other object. Please refer to the *Create Object* popup documentation for more details.

8.1.3.9   Show Object

The *Show* button allows you to make hidden objects visible, so that their icons are displayed in the Project View. Among the hidden objects there are usually some colormaps which are loaded at startup. This option will be unavailable if there are no hidden objects or if the current Project View is the *Tree View*.

8.1.3.10   Show All Objects

The *Show All* button makes all currently hidden objects visible, so that their icons are displayed in the Project View. This option will be unavailable if there are no hidden objects or if the current Project View is the *Tree View*.

8.1.3.11   Remove All Objects

The *Remove All Objects* button deletes all currently visible icons and the associated objects from the Project View. A pre-loaded colormap that is currently visible is also deleted, but all hidden objects are retained. If you select the option *check if data objects need to be saved* in the Preferences dialog, a warning dialog is popped up if there are data objects which have not yet been saved to a file.

8.1.3.12   Make All Display Modules Pickable

This button makes all display modules in the Project View pickable. A display module is pickable if the associated 3D object can be picked.

8.1.3.13   Make All Display Modules Unpickable

This button makes all display modules in the Project View unpickable. So, the associated 3D object of all the display modules in the Project View can't be picked.

8.1.3.14 Duplicate Mode

This mode is applicable when multiple modules of the same type will be attached to a single data object. If the toggle is on, the port values of the first module will be used to initialize the port values of subsequently attached modules. This can be convenient if you want the modules to have the same initial port values.

Example: Attach an *Ortho Slice* to your data object and set the Mapping type to "Colormap" and the colormap range to 0-3. If you attach another *Ortho Slice* to the same data object, its Mapping type will be "Colormap" and its colormap range will be 0-3.

8.1.3.15 Auto adjust range of colormaps

This option specifies the default behavior of *colormap ports.* If this option is checked, the auto adjust range option of colormap ports will be on.

Note: For some modules, the auto adjust range option of their colormap ports is always checked by default, independently of the global value.

8.1.4 View Menu

The *View* menu provides control over several *Viewer* options which affect the display independent of the *Viewer* input.

8.1.4.1 Layout

The *Layout* button lets you select between one, two, or four 3D viewers. All viewers will be placed inside a common window using a default layout. If you want to create an additional viewer in a separate window, choose *Extra Viewer*. You may create even more viewers using the Tcl command `viewer <n> show`. Starting from *n=4*, viewers will be placed in separate windows.

8.1.4.2 Background

The *Background* button opens the background dialog, allowing you to switch between the *uniform* and *gradient* background styles. In addition, the dialog allows you to adjust the two colors used by these styles. Note that the *checkerboard* style is deprecated.

In order to change the background color via the command interface, use the viewer commands `viewer <n> setBackgroundColor` and `viewer <n> setBackgroundColor2`. The command interface also allows you to place an arbitrary raster image into the viewer background (see Section 9.4.4.1 Viewer command options).

8.1.4.3 Transparency

The *Transparency* button controls the way of calculating pixel values with respect to object transparencies during the rendering process.

- *Screen Door:* Transparent surfaces are approximated using a stipple pattern.
- *Add:* Additive alpha blending.
- *Add Delayed:* Additive alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Add Sorted:* Like *Add Delayed*, but transparent objects are sorted by distances of bounding box centers from the camera and are rendered in back-to-front order.
- *Blend:* Multiplicative alpha blending.
- *Blend Delayed:* Multiplicative alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Blend Sorted:* Like *Blend Delayed*, but transparent objects are sorted by distances of bounding box centers from the camera and are rendered in back-to-front order.
- *Sorted Layers:* Uses a fragment-level depth sorting technique, which gives better results for complex transparent objects. Multi-Texture, Texture Environment Combine, Depth texture, and Shadow OpenGL extensions must be supported by your graphics board. If the graphics board does not support these extensions, behaves as if *Blend Sorted* was set.
- *Sorted Layers Delayed:* Like *Sorted Layers*, but rendering all transparent objects after opaque ones.

Note: Antialiasing is not supported by *Sorted Layers* and *Sorted Layers Delayed* mode.

8.1.4.4 Lights

The *Lights* menu lets you activate different light settings for the 3D viewer. By default, the viewer uses a single headlight, i.e., a directional light pointing in almost the same direction as the camera is looking. The headlight can be switched on or off in each viewer via the viewer's popup menu. Alternatively, the headlight can be switched on or off for all viewers using the headlight toggle in this *Lights* menu. This standard light settings can be restored using the *Standard* button. More light settings can be defined by creating an appropriate file in $AVIZO_ROOT/share/lights.

At any time, additional lights can be created via the *Create light* option. Except for the viewer's default headlight, all lights are represented by little blue icons in the Project View, just like ordinary data objects or modules. In order to make all hidden light icons visible, use the *Show all icons* option. *Hide all icons* hides the icons of all light objects. For more information about *lights*, please refer to the Reference Section of this manual.

8.1.4.5   Fog

The *Fog* button introduces a fog effect into the displayed scene and controls how opacity increases with the distance from the camera. The fog effect will only be seen on a *uniform* background. More fine tuning is provided by the `fogRange` *Viewer command*.

- *None:* No fog effect (default).
- *Haze:* Linear increase in opacity with distance.
- *Fog:* Exponential increase in opacity with distance.
- *Smoke:* Exponential squared increase in opacity with distance.

8.1.4.6   Antialiasing

The *Antialiasing...* button opens a dialog with which the antialiasing quality values of all active viewers can be modified. Antialising is the process of smoothing jagged edges on graphic images by using intermediate shades.

The dialog provides several different interface components: a checkable group box, a quality slider, and three buttons.

The checkable group box allows you to turn antialising on or off for all viewers. The slider and corresponding text field let you set antialiasing quality with a value ranging from 0 to 1, 1 being the best. The buttons are used to apply, save or reset changes and to quit the dialog. A detailed description of the interface components is given below.

Antialiasing group box: The group box is composed of a slider and a checkbox. When checked, antialiasing is applied with the quality value given. Otherwise, antialiasing is disabled.

Quality slider: The antialiasing quality slider presents the quality range. If the slider value changes, the corresponding antialiasing is applied. This slider is accompanied by a text edit field to view the exact value of the current antialiasing and to set its numerical value. The component values are always in the range of 0 to 1.

Antialiasing can be done in two different ways. If full-scene antialiasing is supported via the ARB_multisample and ARB_pixel_format OpenGL extensions, it is used for rendering. Note that the number of samples used in the antialiasing computation depends on your graphics hardware and on your video driver. NVidia graphics hardware can support number of samples * 2 levels of quality (assuming the NV_multisample_filter_hint OpenGL extension is available). If it is not supported, smoothing and multipass antialiasing will be used instead.

Buttons: The three buttons named *OK*, *Save* and *Cancel* are for closing the dialog and applying the changes to the antialiasing (OK), saving the status and the quality in preferences (Save) and closing the dialog without applying the changes (Cancel).

Note: Antialiasing is not supported by *Sorted Layers* and *Sorted Layers Delayed* transparency modes.

8.1.4.7 Enable Shadows

This toggle allows you to activate/deactivate the shadowing for display modules. For more information about shadow casting, please refer to the *Shadowing* documentation.

8.1.4.8 Axis

The *Global Axes* button creates a *Global Axes* module which immediately displays a coordinate frame in the viewer window. This button is a toggle, so clicking on it again deletes the *Global Axes* module and removes the coordinate frame from the viewer window. The axes will be centered at the origin of the world coordinate system. You may also create local axes by selecting the appropriate entry from a data object's popup menu.

8.1.4.9 Measuring

The *Measuring* button creates an instance of a *Measurement* module that lets you measure distances and angles on objects within the viewer.
Note: Depending on the Avizo edition, the created *Measurement* module won't be the same.

8.1.4.10 Frame Counter

The *Frame Counter* toggle lets you switch on a frames-per-second counter that will be displayed in the first viewer (viewer 0).

8.1.5 Window Menu

The *Window* menu allows you to manage the visibility of some interface components like the *Console* or the *Help* panels. These panels are, in fact, dockable windows which can be docked in any part of the Avizo main window.

Note that, when workrooms are defined, the different workrooms will be listed within this menu.

The Window menu is different in each workroom and lists the panels available in the current workroom only. For instance, when the active workroom is Project View, the Window menu will contain:

8.1.5.1 Hide Panels

This toggle allows you to show/hide all the panels of Avizo except the viewers.

8.1.5.2 Colormap

This toggle allows you to show/hide the *Colormap Editor* panel of Avizo.

8.1.5.3   Console

This toggle allows you to show/hide the *Console* panel of Avizo.

8.1.5.4   Tables

This toggle allows you to show/hide the Tables panel of Avizo, i.e., the one containing the tables of a *spreadsheet data.*

8.1.5.5   Project View

This toggle allows you to show/hide the *Project View* panel of Avizo, i.e., the one containing the *Project View* (*Graph View* or *Tree View*).

8.1.5.6   Properties

This toggle allows you to show/hide the *Properties* panel of Avizo.

8.1.5.7   Help

This toggle allows you to show/hide the *Help* panel of Avizo.

8.1.5.8   Correlation

This toggle allows you to show/hide the *Correlation panel* of Avizo.

8.1.5.9   Histogram

This toggle allows you to show/hide the *Histogram panel* of Avizo.

8.1.5.10   Toolbars

In addition of these panels, you can also manage the visibility of some Avizo toolbars via the *Toolbars* submenu.

8.1.5.11   Restore default layout

This toggle is used to restore Avizo to its default layout.

8.1.6 Help Menu

The *Help* menu gives you access to documentation like the Avizo's *User's Guide* or *Programmer's Guide* but also to other information like the *License Manager*, the *News* or the *System Information* dialog.

8.1.6.1 User's Guide

This button shows the *Help* dialog on the Avizo documentation home page. You will have access to the entire Avizo documentation.

8.1.6.2 Examples

This button shows the *Help* dialog on the Avizo documentation demos page. You will have access to the Avizo provided *Examples*.

8.1.6.3 Programmer's Guide

This button shows the *Help* dialog on the Avizo documentation *Programmer's Guide* page. You will have access to the programmer's documentation which is useful if you have an Avizo XPand Extension license.

8.1.6.4 Programmer's Reference

This button launches the Avizo*Programmer's Reference* document (`$AVIZO_ROOT/share/devrefAvizo/Avizo.chm`)

which gives you information about Avizo classes and allows you to navigate into the Avizo class list, hierarchy and members. This documentation is useful if you have an Avizo XPand Extension license.

8.1.6.5 License Manager

This button launches the Avizo*License Manager* which displays the status of your Avizo licenses.

8.1.6.6 Show Available Extensions

This button launches a dialog which displays the activated licenses on your computer.

8.1.6.7 System Information

This button launches the Avizo*System Information Dialog* which displays some information about your system. This information allows you and the Avizo support team to better analyze software problems.
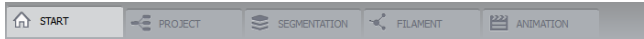
Figure 8.2: The Workroom Toolbar.

8.1.6.8 Online Support

This button launches an external web browser on the Technical Support contacts web page of Avizo.

8.1.6.9 About

This button launches a dialog containing Avizo build and copyrights information.

8.1.7 Standard Toolbar

This toolbar provides quick access to some important actions in Avizo (also accessible from the menus in Avizo) like opening/saving data or a Project or opening the *Preferences Dialog*.

The following actions have a shortcut in the *Standard Toolbar*:

- *Open Data*, *Save Data*
- *New Project*, *Open Project*, *Save Project*
- *Preferences*

Note: by default, this toolbar is hidden, it can be shown via the Window / Toolbars menu.

8.1.8 Workrooms Toolbar

This toolbar provides quick access to some important workrooms and is available in the upper part of the main window.

You can use the following shortcuts to navigate through the workrooms:

- Ctrl + Tab : Open next workroom.
- Ctrl + Shift + Tab : Open previous workroom.

To learn more about workrooms concept, please refer to the following *documentation*.

8.1.9 Project View

The *Project View* of Avizo contains the *Project View* (*Graph View* or *Tree View*). By default, the *Project View* is displayed on top of the *Properties Area* but you can easily move it where you want in the Avizo main window since the *Project View* is a dock widget.

The *Project View* represents data objects and modules currently in use as well as connections indicating dependencies between objects and modules. You can easily switch between 2 different views:

- the *Graph View* where objects are represented with icons and dependencies between objects are indicated with connection lines,
- the *Tree View* where objects are arranged in a tree in which objects are displayed underneath those objects on which they depend.

To change the current *Project View*, you can click on the *Graph View* or *Tree View* buttons in the *Project Menu*.

Note: Other than when attention is being specifically called to highlight the differences between the *Project Graph View* and the *Project Tree View*, *Project View* will be used throughout the Avizo documentation to refer generically to the pane of the Avizo main window containing the collection of objects and modules that define the visualization project.

8.1.9.1   Project Graph View

Concepts
The *Project Graph View* is displayed if *Graph View* is selected. This selection is made by clicking the *Graph View* button in the *Project Menu*.

Once a data object has been loaded or a module has been created, it will be represented by an icon in the *Project Graph View*. Some objects, especially initially loaded colormaps, may not be visible here. Such hidden objects are listed in the *Project > Show Object* menu of the Avizo main window. Selecting an object from this menu causes the corresponding icon to be made visible in the *Project Graph View*.

Icon colors are used to indicate different types of objects. Data objects are shown in green and are the only objects which can be saved to disk using *File > Save Data*, *File > Save Data As* or *File > Export Data As* menu entries. Computational modules are shown in red, visualization modules are yellow, and visualization modules of *slicing* type are orange. These modules, *Ortho Slice* for example, may be used to clip the graphical output of any other module.

Connections between data objects and processing modules, shown as blue lines, represent the flow of data. For display modules, these connections show the data used to generate the display. You may connect or disconnect objects by picking and dragging a blue line between object icons.

As you might expect, not all types of processing modules are applicable to all kinds of data objects. If you click on a data object icon with the right mouse button, a menu pops up that shows all types of modules that can be connected to that object (alternatively, you can click on the small white triangle on the right side of the icon with the left, right, or middle button or press [Ctrl]+[Space]). For more details about this menu, please refer to the *Object Popup* documentation. Selecting one of the modules will automatically create an instance of that module type and connect it to the data object. A new icon and a connecting line will appear in response. This way you can set up a more or less complex project that represents the computational steps required to carry out a specific visualization task and is used to

trigger them. Note that modules used will appear as shortcut (or macro) buttons in the upper part of the window.

If you look closer at an object's icon you will notice two small squares on its left, one white and one orange. If you click on the white square with the left (or right, or middle) mouse button, a menu pops up that shows all connection ports of that object.

If the white square has a "-" or a "+" inside, you can click it with the left (or right, or middle) mouse button to hide ("collapse") or unhide ("expand") the objects connected to that object. The collapse feature is helpful if there are too many objects in the *Project Graph View* to easily view them all at once. The hidden objects will be visible in the *Project > Show Object* menu.

The orange square controls the object's visibility in the viewers. It shows the current viewer layout (1 viewer, 2 viewers, etc.). Click inside the region of the square corresponding to a specific viewer to toggle the visibility of the object in that viewer. The region turns gray when the module is set to invisible. If you are using more than 4 viewers, each additional viewer will have its own orange square on the object's icon. You can also control an object's visibility by clicking on its visibility toggle in the *Properties Area*.
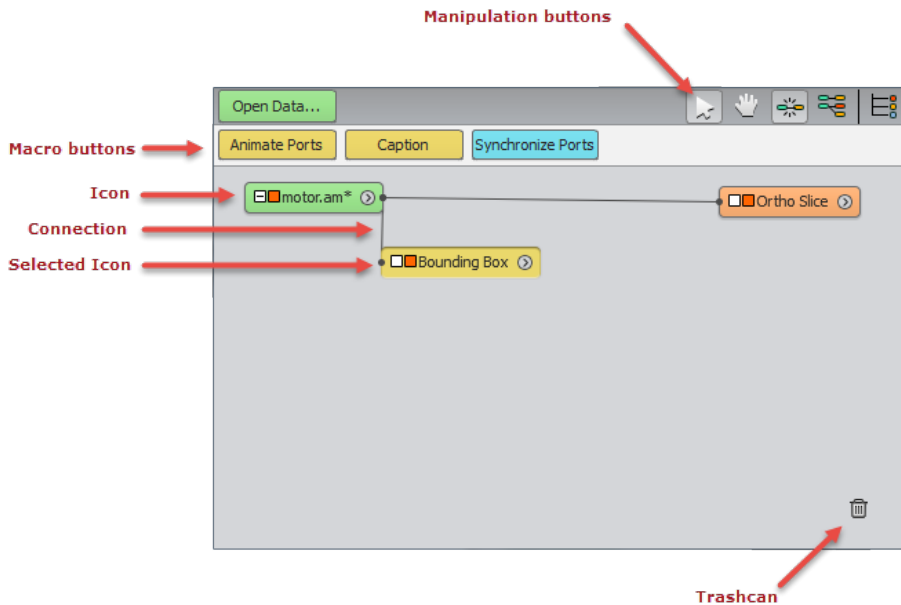
Figure 8.3: The *Project Graph View* contains data objects and module icons.

As mentioned above, for most objects the required connections are established automatically on creation. However, in order to set up optional connections, you must use the connection popup menu. For

example, you may attach an optional scalar field to an *Isosurface* module's *Colorfield* port in order to color the surface using the values in the *Colorfield*.

Once you have selected an entry from the connection popup menu of the object icon, you can choose a new input object for that port. In order to do so, click on the input object's icon in the *Project Graph View*. The blue connection line will become lighter blue if the connection port can be connected to the chosen object. Reasons for not being able to connect an input to a port can include the following: incompatible data type (use *Convert Image Type* to change), incompatible dimensionality of the input (use *Channel Works* to change), incompatible grid type of the data (for example, use *Arithmetic* to sample a regular grid), or simply that the connection does not make sense, for example, connecting a display module for surfaces to a volume data set.

In order to disconnect an input object, click on the icon of the module to which the port belongs. Data objects possess a special connection port called *Master*. This port refers to a computational module or editor to which the data object is attached. It indicates that the computational module or editor controls the data object, i.e., that it may modify its contents.

Each object has an associated control panel containing buttons and sliders for setting or changing additional parameters of the object. The control panel becomes visible once the object has been selected, i.e., by clicking on its icon with the left mouse button. In order to select multiple objects, you can shift-click the corresponding icons or select them by using the rectangular selection tool (press and hold down the left mouse button over the *Project Graph View* background, then sweep out a rectangle). Clicking on the icon of a selected object deselects it again. Clicking somewhere on the background of the *Project Graph View* causes all selected objects to be deselected. One or more selected icons may be dragged around in the *Project Graph View* by clicking on them and moving the mouse pointer while holding down the mouse button.

Interface

At the right of the *Project View* banner are the *Auto-Display activator* ☀, *Project View Select Mode* ⟋, *Project View Pan Mode* ✋, *Reorder Project View* ⬌ and the *Switch To Project Tree View* ☰ buttons. In *Project View Select Mode*, the mouse is used for selecting, positioning, connecting, and disconnecting objects in the *Project Graph View*. In *Project View Pan Mode*, moving the mouse pans the *Project Graph View* workspace. In case your current *Project* is not well organized, i.e., contains a lot of objects and you are not able to easily find an object icon among others or visualize dependencies between objects, you can click on the *Reorder Project View* button. In this case, the *Project Graph View* will be reorganized with object icons reordered according to the following rule: data objects are on the left, compute objects in the middle and visualization modules on the right. The last button switches the Project View to *Project Tree View*.

At the top of the *Project Graph View* is a region containing shortcut (or macro) buttons. The *Open Data* button is a shortcut to the *File > Open Data* menu item. Up to 4 additional buttons are automatically displayed depending on the currently selected object(s). They provide easy access to the modules that are most commonly used and/or that have been recently used with the currently selected object.

The trashcan 🗑 at the bottom right of the *Project Graph View* provides a convenient shortcut for removing an object. Drag the object to the trashcan. When the cursor turns into an "X", release the

mouse button and the object will be removed from the *Project View*. If you remove a data object, all modules downstream from that module will be deleted as well. Alternatively, you can use the *Remove Object* item from the *Project* menu, or you can use the [Delete] key to remove selected object(s).

8.1.9.2   Project Tree View

The *Project Tree View* is displayed if *Tree View* is selected. This selection is made by clicking on the *Tree View* button in the *Project Menu.*
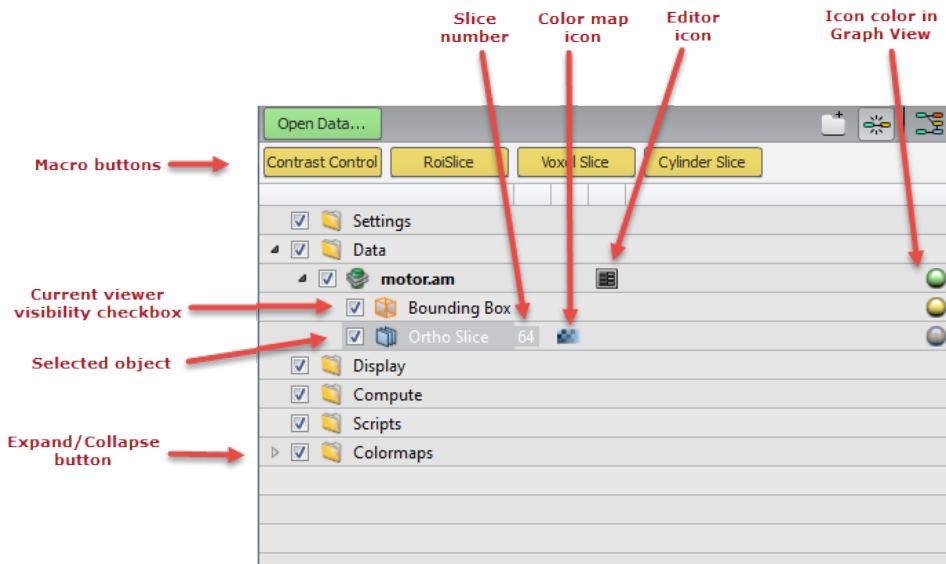
Figure 8.4: The *Project Tree View* contains data objects and module icons organized in a tree view.

Once a data object has been loaded or a module has been created, it will be represented by a new entry in the tree view.

Each object has an associated control panel containing buttons and sliders for setting or changing additional parameters of the object. The control panel becomes visible in the *Properties Area* once the object has been selected, i.e., by clicking on its icon with the left mouse button. In order to select multiple objects, you can shift-click the corresponding icons. Clicking on the icon of a selected object deselects it again. Clicking somewhere off of the tree view, e.g., beyond the bottom of the tree view, causes all selected objects to be deselected.

At the top of the *Project Tree View* is a region containing shortcut (or macro) buttons. The *Open Data* button is a shortcut to the *File > Open Data* menu item. Up to 4 additional buttons are automatically

displayed depending on the currently selected object(s). They provide easy access to the modules that are most commonly used and/or that have been recently used with the currently selected object.

The tree view display has 5 columns:

Column 1: Contains the actual tree view.

- **Organization**: The tree is organized into folders containing different kinds of objects: *Data*, *Display*, *Compute*, etc. Most of the folders are predefined and cannot be modified. A few, however, such as the *Data* folder, can have more folders added beneath.

  Currently, there are two "templates" which control the organization of the tree view: a default template for most Avizo users and a "geophysics" template in the Avizo XEarth Extension which includes predefined folders for 3D surveys, horizons, faults, etc.

- **Connections**: If you right-click on a data object icon, a menu pops up that shows all types of modules that can be connected to that object (alternatively, you can press [Ctrl]+[Space]). For more details about this menu, please refer to the *Object Popup* documentation. Selecting one of the modules will automatically create an instance of that module type and connect it to the data object. A new item in the tree view will appear in response. In this way you can set up a more or less complex project that represents the computational steps required to carry out a specific visualization task and is used to trigger them. Note that modules used will appear as shortcut (also known as "macro") buttons in the upper part of the window.

  For most objects, the required connections are automatically established on creation. The connection is shown in the connected object's input port in its control panel in the *Properties Area*. For example, when you attach a *Ortho Slice* to my_data.am, in the "Data" port of the *Ortho Slice* control panel, you will see "my_data.am" in the pulldown list. If other objects in the *Project Tree View* can be connected to the *Ortho Slice*, they will be in the list as well. To switch the connection, simply select a different item from the list. The tree view (and possibly the viewer display) will be updated accordingly. Selecting "NO SOURCE" from the list disconnects the module from any object.

  Reasons for not being able to connect an input to a port can include the following: incompatible data type (use *Convert Image Type* to change), incompatible dimensionality of the input (use *Channel Works* to change), incompatible grid type of the data (for example, use *Arithmetic* to sample on a regular grid), or simply that the connection does not make sense, for example, connecting a display module for surfaces to a volume data set.

  Data objects possess a special connection port called *Master*. This port refers to the computational module or editor to which the data object is attached. It indicates that the computational module or editor controls the data object, i.e., that it may modify its contents.

- **Expand/Collapse**: You can click on the *arrow* icons at the left to expand or collapse portions of the tree.

- **Drag-and-Drop**: You can use drag-and-drop to move items in the tree view. For example, to connect an existing visualization module to a different data set, you can drag and drop it from its current location onto a compatible data set. Drag-and-drop does not work for disconnecting objects. You must use the connection ports for this.

- **Viewer Toggle**: The check boxes control the visibility of the object in the *currently active* viewer. If there are multiple viewers, e.g., a 4-viewer layout, you can select a viewer, set the object's visibility in that viewer, then repeat as often as necessary to set the desired visibility options for each viewer. However, it will probably be more convenient to control the visibility in individual viewers by using the viewer toggle button in the object's control panel in the *Properties Area*. The viewer toggle button is the orange square just to the right of the module name in its control panel. The button is divided into regions corresponding to the current viewer layout. Click on the region to toggle visibility of the object on/off in the corresponding viewer. If there are more than 4 viewers, each additional viewer will have it own toggle button.

- **Adding a Folder**: At the right of the *Project Tree View* banner are the *New Folder* and the *Switch To Project Graph View* buttons. The *New Folder* button is used for adding a new folder directly below the currently selected folder. You can also add a folder by right-clicking on an existing folder and selecting *New Folder* from the context menu. At some positions in the tree, it may not be possible to add a new folder. The last button switches the Project View to *Project Graph View*.

- **Removing an Object**: To remove one or more objects from the *Project Tree View*, first select the object(s) to be removed. You can then use the *Remove Object* item from the Project context menu, or you can press the [Delete] key to remove selected object(s). If you remove a data object, all modules downstream from that module will be deleted as well.

- **Hovering**: Hovering the mouse over an item in the tree view displays information about that item, usually its type, e.g., HxUniformScalarField3.

Column 2: Shows, for certain objects, the current value of a port of interest. Currently, the *slice number* is shown for slice modules (*Ortho Slice*, *Inline*, etc.) and the *time scale factor* is shown for the *Seismic Settings* module.

Column 3: Shows the current colormap, if any, associated with the module or object. You can right-click on it to show the standard colormap context menu. Or you can left-click on it to bring up the *Colormap Editor*.

Column 4: Displays the icon of the editor, if any, currently operating on the object. For example, the *Crop Editor*, the *Parameter Editor*, etc.

Column 5: Displays a color-coded marker indicating the kind of object or module, as follows:

- *Red:* computational modules

- *Orange:* visualization modules of *slicing* type. These modules may be used to clip the graphical output of any other module.
- *Yellow:* visualization modules
- *Green:* data objects. These are the only objects which can be saved to disk using *File > Save*.
- *Blue:* script objects
- *Purple:* settings modules, e.g., *Seismic Settings*, *XScreen Settings*, etc.

8.1.10   Properties Area

The *Properties Area* is the place where the user interface of selected objects is displayed. Typically, the interface consists of buttons and sliders arranged in *Groups* and *Ports*. Ports are objects that allow the user to pass information to a module through them. A group is an object that contains several ports associated to a common theme. They can be collapsed or expanded. As an example, on the Figure 8.5, ports regarding resampling and optimizations options are respectively gathered in **Resampling Options** and **Optimizer Options** groups. Two others groups that are collapsed are **Metric** and **Localizers**. Some ports are advanced ports, which means they will only be visible if the **Advanced** toggle at the top right corner of the *Properties Area* panel is ON. Otherwise, these ports will be hidden and set to a default value. You can distinguish advanced ports by the darker color indicator on the left of the *Properties Area*.

Once an object has been selected, its input controls will be displayed in the *Properties Area*. By default, the *Properties Area* is displayed under the Avizo*Project View,* but you can easily move it where you want in the Avizo main window since the *Properties Area* panel is a dock widget. You can also quickly show/hide the *Properties Area* panel by clicking **Properties** in the *Standard Toolbar* or in the *Window Menu.*

Each object has a specific set of controllable parameters or options. These are described in detail for each module in the *modules index section* of the reference manual. Objects (modules, data...) also provide a question mark button that lets you access the documentation of that object directly.

Apply is active (green) for modules that require you to explicitly initiate their action. Typically, this is true for modules whose actions may take a significant amount of time like some of the compute modules.

Check the **auto-refresh** check box to automatically force an *apply* for any change in the project.

An object's name is displayed at the top of its control panel along with various control buttons. All data objects and display modules have one or more orange viewer buttons for each 3D viewer. These buttons control whether any graphical output of an object is displayed in a particular viewer or not. For example, if you have two viewers and two *Isosurface* modules you may want to display one *Isosurface* in each viewer.

Display modules of *slicing* type (orange ones like *Ortho Slice*) provide a clip button. Clicking this button will cause the graphical output of any other module to be clipped by that slice. Clipping does not affect modules with hidden geometry, or modules that are created after the clip button has been
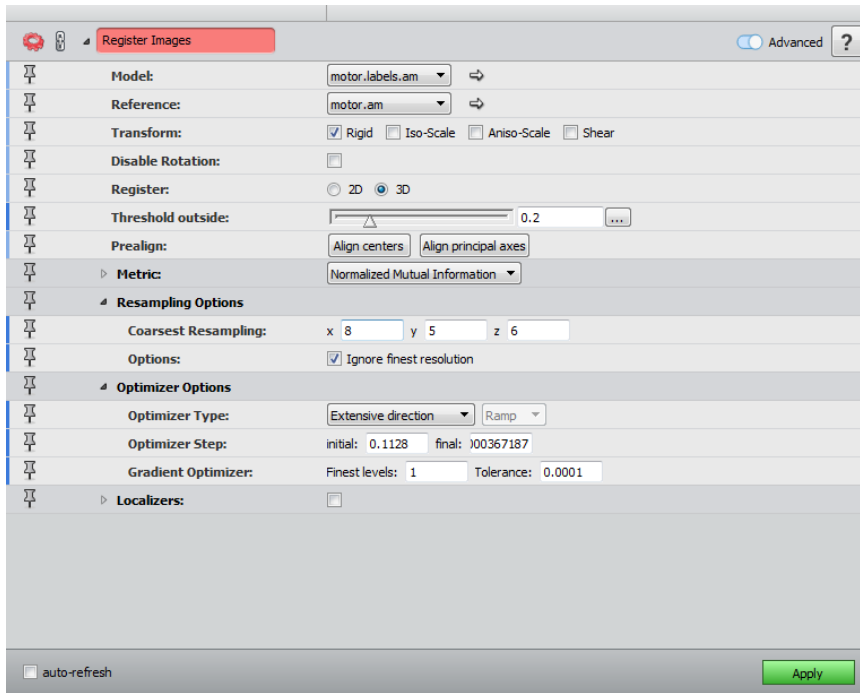
Figure 8.5: The *Properties Area* of *Register Images* module.

pressed.

Data objects provide a number of additional *editor* buttons. Editors are used to modify the contents of a data object interactively. For example, you can perform manual segmentation of 3D image data by editing *Label Fields* using the *Segmentation Editor*. Some editors display their controls in the *Properties Area* like all other objects, while others use a separate dialog box that allows you to perform object manipulations.

As mentioned, specific input controls of an object or a module are organized into *Ports*. Each port has a pin button on its left. If a port is pinned, it will still be visible even when the object is deselected. The ports are composed of widgets that can be used to set the parameters pertaining to various operations (e.g., a value is entered by a slider, a state is set by radio buttons, a binary choice is presented as a toggle button). The control elements have a uniform layout and are divided into several basic types. A description of the basic port types is contained in the *ports Index Section* of the *User's Reference Manual*.

Ports whose labels are displayed in italics are special since they are input connection ports. They are used for connecting data objects and modules into a project that represents the computational steps required to carry out a specific visualization task. Each connection port contains the list of objects to

which it can be possibly connected, plus a "NO SOURCE" item, which means it should not connect to anything.

In *Graph View* mode, a display of these connection ports can be toggled on and off via the *Layout* tab of the *Edit / Preferences* menu. They are displayed by default to help beginners understand the *Project View* architecture.

In *Tree View* mode, these connection ports are always displayed, because this is the only mechanism in this mode for defining projects. When an object is completely disconnected from all others, it will be moved in the tree to an appropriate folder. For example, to *Display* for a display module, to *Compute* for a compute module, and so on.

If the shadowing effect is switched on in the *Rendering* tab of the *Edit / Preferences* menu, some visualization modules will display the **Shadow** button (See Figure 8.6). Clicking **Shadow** will change the following modes:

- *Cast Shadow:* the object will cast shadows
- *Receive Shadow:* the object will only receive shadows
- *Cast & Receive Shadow:* the object will cast and receive shadows
- *No Shadow:* no shadows will be visualized
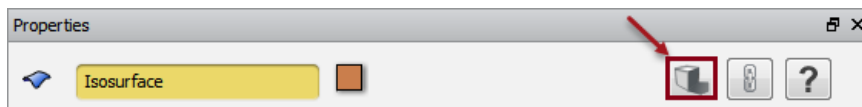
Figure 8.6: *Shadow* Button Example

Only display modules have *pickable* toggles (See Figure 8.7). When the pickable toggle of a display module is off, the user cannot pick it in the viewer.
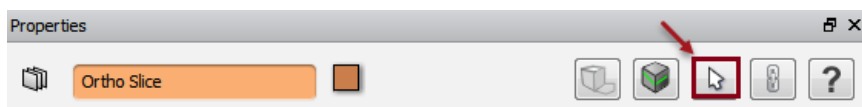
Figure 8.7: *Pickable* Toggles in Display Modules.

8.1.11   Progress Bar

The *Progress Bar* is located in the lower part of the Avizo main window. It is used when computational operations have been started to indicate the percentage of computation which is complete and provide information on what task is currently being performed.

Figure 8.8: Default state of the *Progress Bar*: "Ready" is displayed and the *Stop* button is disabled.

Figure 8.9: State of the *Progress Bar* during a resampling operation (*Resample* module): current computational action is "Resampling" and the *Stop* button is red and enabled.

It provides a *Stop* button which, when red, allows you to interrupt the current action. This button is grayed out when there is no current action or when the current action cannot be interrupted.

You can refer to the *Progress bar command options* for the set of Tcl commands available for the *Progress Bar*.

8.1.12 Viewer Window

The 3D viewer plays a central role in Avizo. Here all geometric objects are shown in 3D space. The 3D viewer offers powerful and fast interaction techniques. It can be regarded as a virtual camera which can be moved to an arbitrary position within the 3D scene. The left mouse button is used to change the view direction by means of a virtual trackball. The middle mouse button is used for panning, while the left and the middle mouse button pressed together allow you to zoom in and out on objects.

The following controls can be used:

- hold down the left mouse button and move the mouse to rotate the camera around its current focal point (the focal point can be changed by doing a seek operation), hold down the left mouse button + [SHIFT] to constrain this rotation around the X or Y axis, and the left mouse button + [CTRL] for the Z axis.

- Hold down the middle mouse button to pan.

- Hold down the left + middle mouse buttons to zoom / dolly, or [CTRL] + the middle mouse button, or [CTRL] + [SHIFT] + the left mouse button

- Click the right mouse button to open the popup menu (see *Viewer popup description* for more details).

- Press the [S] key, then click on an object with the left mouse button to "seek" to that position.

- Press the [P] key, then click the left mouse button to pick a 3D Object in the viewer. The

corresponding module will be selected.

- Press the [Space] key to view the entire 3D scene.

- Press the [ESC] key to switch between "interaction" mode and "trackball" mode.

The virtual trackball, controlled by the left mouse button, allows for free rotation of the camera.

Some *viewer gadgets* may be displayed in one of the corners of the display:

- *camera trackball*: used for constrained rotation of the camera about the screen-aligned X, Y, or Z axes. Click on the vertical wheel (it becomes red when you select it) and move the mouse up/down (while the left mouse button is pressed) to rotate about the X axis. Click on the horizontal wheel (it becomes green when you select it) and move the mouse left/right (while the left mouse button is pressed) to rotate about the Y axis. Click on the third wheel (it becomes blue) and move the mouse up/down (while the left mouse button is pressed) to rotate about the Z axis.
- *3D compass*: indicates the current camera viewing direction. It is an indicator only; you cannot use it to control the viewing direction.

The *Edit > Preferences > Layout* dialog can be used to control the visibility, auto-hide option, and position of the trackball and the compass. Refer to the *Viewer gadgets* section for more details.

Sometimes you need to manipulate objects directly in the 3D viewer. For example, this technique, called 3D interaction, is used by the *Transform Editor*. You can swith on this editor with the transform editor button in the properties of a data object. The editor provides special draggers that can be picked and translated or rotated in order to specify the transformation of a data object. Before you can interact with these draggers, you must switch the viewer into *interaction mode*. This is done by clicking on the arrow button in the upper left corner. If the viewer is in interaction mode, the mouse cursor will be an arrow instead of a hand symbol. You can use the [ESC] key in order to quickly switch between interaction mode and viewing mode. If the viewer is in interaction mode, use the [Alt] key to temporarily switch to viewing mode.

More than one viewer can be active at a time. Standard screen layouts with one, two, or four viewers can be selected via the *View menu*. Additional viewers can be created using the Tcl command viewer <n> show, where <n> is an integer number between 0 and 15. While viewers 0 to 3 will be placed in a common panel window, viewers 4 to 15 will create their own top-level window. For more specific control, the viewer provides an extensive command set, which is documented in Section 9.4.4.1 Viewer command options.

8.1.12.1   Viewer toolbar description

The toolbar of the main viewer window provides several buttons and controls, allowing you, for example, to switch between viewing mode and interaction mode, to choose certain orientations, or to take
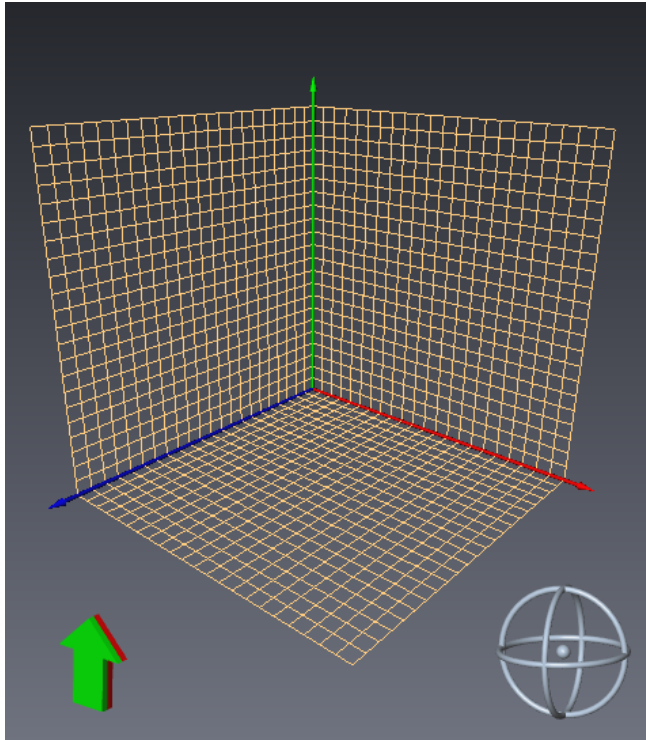
Figure 8.10: Avizo's viewer window provides a virtual trackball for easy navigation, as well as optional viewer gadgets: camera trackball (lower right) for constrained rotation and 3D compass (lower left) for camera viewing direction indication.

snapshots. The precise meaning of these controls is described below.

Interact:

Switches the viewer into interaction mode. You can also use the [ESC] key to toggle between viewing mode and interaction mode.

Trackball:

Switches the viewer into viewing mode. You can also use the [ESC] key to toggle between interaction mode and viewing mode. The left mouse button is used to change the view direction by means of a virtual trackball.

Translate:

Same as *Trackball* except that the left mouse button is used for panning (translation).

Zoom:

🔍

Same as *Trackball* except that, in this mode, vertical motion of the left mouse button controls zooming.

Rotate:

🔁

Rotates the camera around the current view direction. By default, a clockwise rotation of one degree is performed. If the [SHIFT] key is pressed while clicking, a 90 degree rotation is done. If the [CTRL] key is pressed, the rotation will be counterclockwise.

Seek:

⊕

Pressing the seek button and then clicking on an arbitrary object in the scene causes the object to be moved into the center of the viewer window. Moreover, the camera will be oriented parallel to the normal direction at the selected point. Seek mode may also be activated by pressing the [S] key in the viewer window.

Important note: When using front face culling mode on an object, Seek will work on the hidden faces unless you move the camera "through" front hidden faces. This limitation will be solved in the future Avizo releases.

Home:

🏠

Resets the camera to the home position.

Set Home:

🏠

Sets the current position as the new home position.

View All:

Repositions the camera so that all objects become visible. The orientation of the camera will not be changed.

XY-, XZ- and YZ-Views:

Adjusts the camera according to the specified viewing direction. The viewing direction is parallel to the coordinate axis perpendicular to the specified coordinate plane. The opposite view direction is used if the [SHIFT] key is pressed.

Note: In the some Editions, these buttons will be replaced by the geographic view orientation buttons.

Geographic View Orientation:

Adjusts the camera according to the specified viewing direction: from top, from bottom, from west, from east, from south, from north.

Note: In some Editions, these buttons replace the XYZ view buttons.

Perspective/Ortho:

Toggles between a perspective and an orthographic camera. By default, a perspective camera is used. Orthographic camera may be used in order to measure distances or to exactly align objects in 3D space.

Note: Only one of these buttons will be visible at a time, the button indicating the currently active camera type.

Stereo:

Allows you to enable or disable stereo viewing, as well as specify various stereo viewing parameters via the *Stereo Preferences* dialog.

Pick:

Pressing the pick button and then clicking on an arbitrary object in the scene causes the corresponding module to be selected. Picking mode may also be activated by pressing the [P] key in the viewer window.

Measuring:

Pressing this button creates an instance of a *Measurement* module that lets you measure distances and angles on objects within the viewer. Clicking on the down arrow will display a menu of measuring tools from which to choose: *3D Length*, *3D Angle*, *3D Annotation*, *3D Box*, *3D Circle*. Only one of these buttons will be visible on the toolbar at a time, the button of the measuring tool most recently accessed from the viewer toolbar.

Note: Depending on the Avizo edition, the created *Measurement* module won't be the same.

Quick Probe:

This button allows you to choose the probing mode available in **Interaction mode**.

By default, Quick Probe is move sensitive, i.e., voxel value and coordinates under the mouse cursor are displayed in the progress bar.

With click on this button you can disable the quick probe mode, make it move sensitive or click sensitive. Click sensitive behavior displays coordinates and value if [SHIFT] key is pressed and left mouse is clicked on some visualization in the viewer.

Note: If [SHIFT] key is pressed, no other interaction is possible in the viewer except clicking to probe.

Display unit:

nm ▼

The button specifies the currently selected display unit.

Clicking on the down arrow of this button will display a menu of all possible display units.

Note: This button is visible only if unit management is activated and display units are modifiable, i.e., "Lock display units on working units" is unchecked on preference options.

Snapshot:

📷

Takes a snapshot of the current rendering area and saves it to a file. The filename as well as the desired output format must be entered through the *Snapshot dialog*. Snapshots may also be taken using the *viewer command* `snapshot`.

Layout:

▣ ▥ ▤ ▦

Selects the viewer layout: a single view, two viewers side-by-side, two viewers stacked, or four viewers.

Fullscreen:

▢

Selects fullscreen mode. In this mode, the viewer occupies the entire screen and no other windows will be visible. To exit fullscreen mode, click the right mouse button and uncheck *Fullscreen* in the popup menu.

Link objects visibility:

⧉

Links the visibility of objects between all viewers. When checked, it means that changing the visibility (viewer mask) of an object in one viewer will change it in all viewers.

In addition to these buttons, the Avizo viewers provide an extensive set of Tcl commands, which are listed in Section 9.4.4.1 Viewer command options.

8.1.12.2   Viewer popup description

All viewers include a popup menu that allows you to configure various options. A right mouse button click opens this popup menu. The following menu options are available:

Functions:

Contains a sub-menu with items which are respectively shortcuts to *Home*, *Set Home*, *View All* and *Seek* icons.

Viewing:

Switches between viewing mode and interaction mode (see *Interact* and *Trackball* icons for more details). Same as pressing the [ESC] key.

Fullscreen:
Shortcut to the *Fullscreen* icon.

Headlight:
Activates and deactivates the headlight (see *Lights* section for more details).

Preferences:
Contains the following menu items:

- *Seek to point*: When activated, the *Seek* tool will focus on the clicked point, else it will focus on the object's origin.
- *Auto clip planes*: Adjusts the camera near and far plane at each camera move.
- *Automatic interactive mode* : This option allows to get better performance when viewer is in *Interact* mode.
- *Stereo*: Shortcut to the *Stereo* icon.
- *Spin animation*: When spin animation is enabled, if the mouse is moving while you release it, the trackball will continue to spin afterwards. By default, spin animation is disabled.
- *Rotation axes*: Displays rotation axes at the center of the viewer.

Show trackball:
Shows/hides the camera trackball (see *Viewer gadgets* section for more details).

Show compass:
Shows/hides the compass (see *Viewer gadgets* section for more details).

Link camera to:
This menu option creates a camera link from the current viewer to the one you select (by clicking it after you select the menu option); pressing the [ESC] key before selecting a viewer aborts the link operation. When the cameras of two viewers are linked, they view both scenes from the same 3D point and looking in the same direction. The section *Unlink camera* below explains how to remove a camera link.

Unlink camera:
This menu option removes a camera link between two viewers by selecting the viewer to unlink.

Show objects in extra viewer:
This menu option allows showing objects from the current viewer in the extra viewer using the same viewer masks.

Object visibility:
This sub-menu controls the object visibility in the viewer.
The following options are available:

- *Hide All*: Hide all objects for the viewer.
- *Show All*: Show all objects for the viewer.

- *Visible objects*: Sub menu to control the visibility of all display modules in the current viewer. When checked, it means that the module is visible.
- *Link to all viewers*: Link the visibility of objects between all viewers. When checked, it means that changing the visibility (viewer mask) of an object in one viewer will change it in all viewers.
- *Same as viewer 0, Same as viewer1,...,Same as extra viewer*: Copy the object visibilities of the viewer i.

Identify:
This menu option briefly displays the viewer identifier in the viewer.

Identify all viewers:
This menu option causes each viewer to briefly display its viewer identifier.

8.1.13   Consoles Panel

The *Consoles Panel* provides command shells allowing access to Avizo's advanced control features.

By default, the *Consoles Panel* is hidden, but you can easily display it by clicking on the *Consoles* button in the *Standard Toolbar*, in the *Window Menu* or at the right of the *Progress Bar*, or using the CTRL+ALT+A shortcut. In this case, the *Consoles Panel* will be displayed under the viewer, but you can easily move it where you want into the Avizo main window since the *Consoles Panel* is a dock widget.

Figure 8.11: The *Consoles Panel* for Info Messages and Scripting Commands

Two command shells are available: one based on the *Tcl* script language *(Tool Command Language)*, one based on *Python* script language.

8.1.13.1   Tcl Console

The default shell of Avizo serves two purposes. First, it gives you some feedback on what is currently going on. Such feedback messages include warnings, error indications and notes on problems as well as information on results. Second, it provides a command line interface where Avizo commands can be entered using the *Tcl* script language. Examples are:

```
load C:/MyData/something.am
viewer 0 setSize 200 200
viewer 0 snapshot C:/snapshot.tif
```

The Tcl scripting syntax and the specific commands are described in the chapter *Scripting*. To execute a single command, just type in its name and arguments and press *Enter*. If you select an object and then press the TAB key on the empty command line, then the name of the object will be automatically inserted.

You can also type the beginning of a command word and type the TAB key to complete the word. This only works if the beginning is unique. Pressing TAB a second time will show the possible completions. Often, this saves a lot of typing. Commands provided by data objects and modules are documented in the reference section of the users guide. Pressing the F1 key for such a command without any arguments pops up the help text for this command. This is also true for commands provided by the *ports* of an object.

Additionally, the *Tcl Console* provides a command history mechanism. Use *up arrow* and *down arrow* to scroll up and down in the history list.

To execute a file containing many Tcl commands, use source <filename> or load the script file via Avizo's file dialog from the file menu. Avizo script files are usually identified by the extension .hx. For advanced script examples, take a look at Avizo's demo files located in $AVIZO_ROOT/share/demo.

8.1.13.2 Python Console

An alternative script shell is available, based on *Python* language. For more information, go to the *Python* section.

8.1.14 Online Help

Avizo user's documentation is available online. You can access it via the *User's Guide* entry of the main window's *Help* menu. The user's guide contains some introductory chapters, as well as a reference section containing documentation for specific

- modules,
- data types,
- editors,
- file formats,
- and other components.

You can access the documentation of any such object via a separate index page accessible from the home page of the online help browser. Avizo modules also provide a question mark button in the Properties Area. Pressing this button directly pops up the help browser for the particular module.

By default, the help browser is displayed in its own top-level window, but you can easily integrate it where you want into the Avizo main window since the help browser is a dock widget.

Going through the online documents is similar to text handling within any other hypertext browser. In fact, the documentation is stored in HTML format and can be read with a standard web browser as well. Use the *Backward* and *Forward* buttons to scroll in the document history and *Home* to move to the first page.

Searching the online documentation
The online help browser provides a very simple interface for a content and full text search. If you want to search through the complete documentation, enter the desired text into the text field. Pressing *CTRL-SHIFT-F* moves the focus to this text field and marks all text in this field for quick access. The search will be performed upon pressing the *Enter* key.

The search is done by using the complete search phrase. For example, suppose you are looking for information about the *Surface Editor*. The output shows the page title where the phrase is contained and a small text surrounding the search phrase.

Searching in a help document is done by entering text in the *Find pane*. You can open it by clicking on the *Show find pane* icon or by pressing *CTRL-F*. This function searches the content and looks for the occurrence of the complete phrase you type into the text edit field. Pressing *Enter*, *F3* or clicking on *next* will mark the searched phrase in the text. Another click/keypress takes you to the next occurrence of this phrase. You can search backwards by pressing *SHIFT-F3* or clicking on *prev* to find the previous occurrence of the phrase. If the search reaches the beginning or end of the document, it starts over continuing in the same search direction. The searched phrase is colored with a yellow background wherever it is found in the text. The *Find pane* can be closed by clicking the red X on the pane or by unchecking the *Show find pane* icon.

All searches are case insensitive!

Running demo scripts
In the demo section of the on-line manual, you can easily start any demonstration just by clicking on the marked text. The script will be loaded and executed immediately. You may interrupt running demo scripts by using the stop button in the lower right of the Avizo main window.

Commands

```
help
```
Makes the help dialog appear and loads the home page of the online help.

```
help getZoomFactor
```
Returns the zoom factor of the browser.

```
help setZoomFactor <ZoomFactor>
```
Sets the zoom factor of the browser.
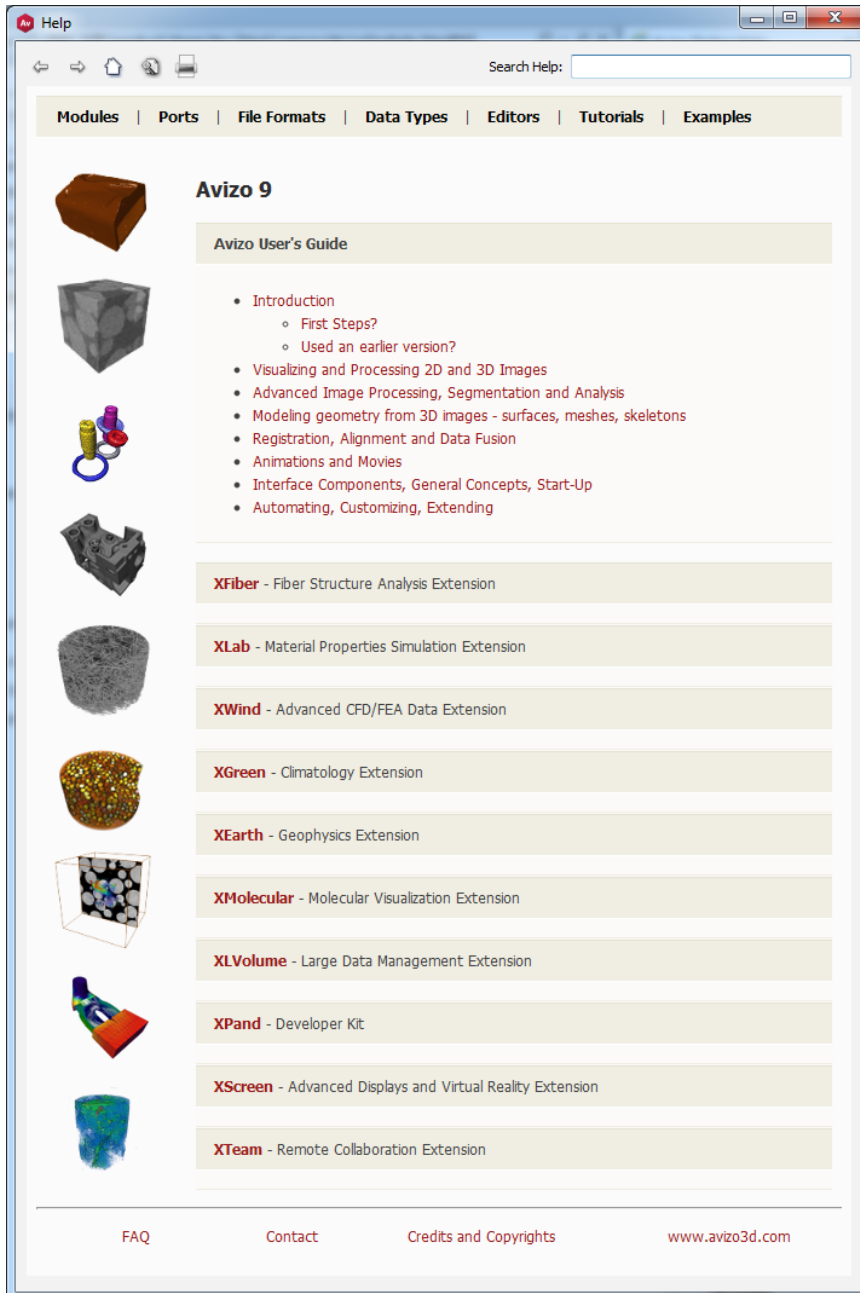
Figure 8.12: Avizo's help window.
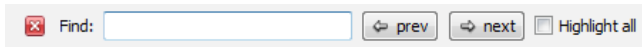
Figure 8.13: The *Find pane*.

`help load `*`file.html`*

Loads the specified hypertext document in the file browser. Note that only a subset of HTML is supported.

`help reload`

Reloads the current document.

`help invalidateCache`

Destroys the search cache. In order the search function in Avizo's help to be more responsive, the help files are parsed and cached. If for any reason there is a need to invalidate this cache, it could be performed by typing this Tcl command.

8.1.15   Histogram Panel

The histogram panel lets you display the distribution of a given measure. The entire range of values is divided into a series of intervals (bins), and the histogram panel displays how many values fall into each interval. The Avizo histogram panel allows the display of multiple measures in the same window. And multiple histogram windows can be created. Typical inputs include:
- *Images*
- *Spreadsheets*
- *Label analysis*
- *Spatial graph*
- *Pore network models*

Here is an example of use:

- Load `data/tutorials/motor.am` and `data/tutorials/motor.labels.am`
- Select the *Histogram Panel*
- In the histogram panel, select *motor.am* in the pool down menu of the new input option
- A new menu will appear next to the first input, if the input data is not an *image (grayscale data, label data)*, listing the list of measures that can be selected. Otherwise, an input mask can be used. This mask is the label data. A material can be defined as a mask to do the histogram.

- Press the + button
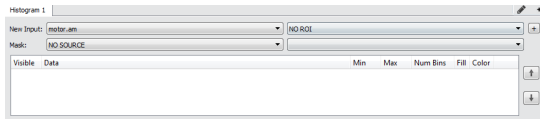
Figure 8.14: Measure list
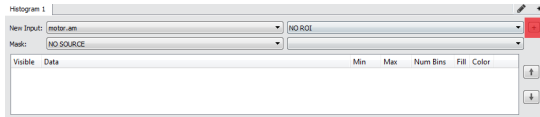
Figure 8.15: Add histogram button

- Pixel distribution is displayed

- In the histogram panel, select `motor.labels.am` in the input *Mask*.
- A new menu will appear next to the mask input, listing the list of materials that can be selected.

- Choose a material and press the + button. Two histograms are now displayed next to each other.
- The second histogram shows the pixel distribution which correspond to the selected material at the step before.

Note: When a first histogram has been displayed, only histogram from the same data type can be overlayed. If you load a 3D image data, it will not be present in the new input drop down list, if an histogram has already been displayed. You will need to remove the histogram to get this option or create a new tab (see below).

Multiple tabbed histograms can be created:

- Load `data/tutorials/chocolate-bar.am`
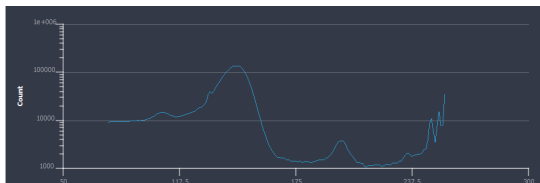
Figure 8.16: Data histogram
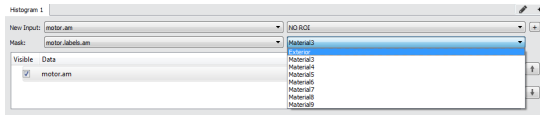
Figure 8.17: Data histogram and material histogram
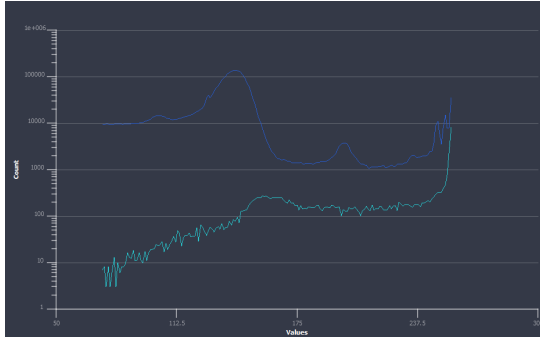
Figure 8.18: Data histogram and material histogram

- Click the Add New Histogram tab button:

- New histogram tab can also be renamed using the Rename Tab button:

- In the histogram panel, select chocolate-bar.am in the pool down menu of the new input option.
- Select and click the Add Histogram:

- As you see in the last image, it is possible to set the range in which the histogram should be displayed. The number of bins can also be set to increase or decrease the histogram resolution.

- New tabs can be removed: Hover the mouse on the tab to see the Close button.
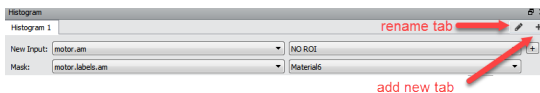
Figure 8.19: Histogram tab
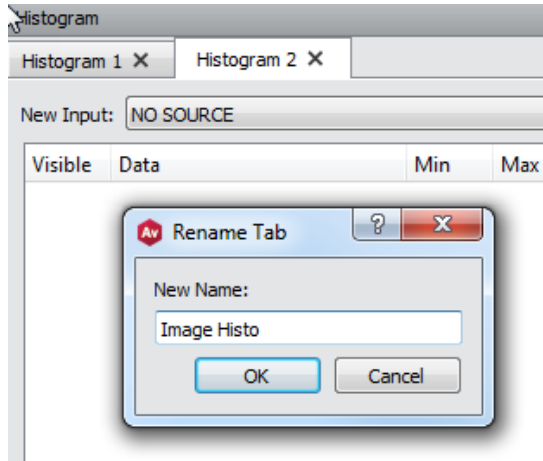
Figure 8.20: Rename tab
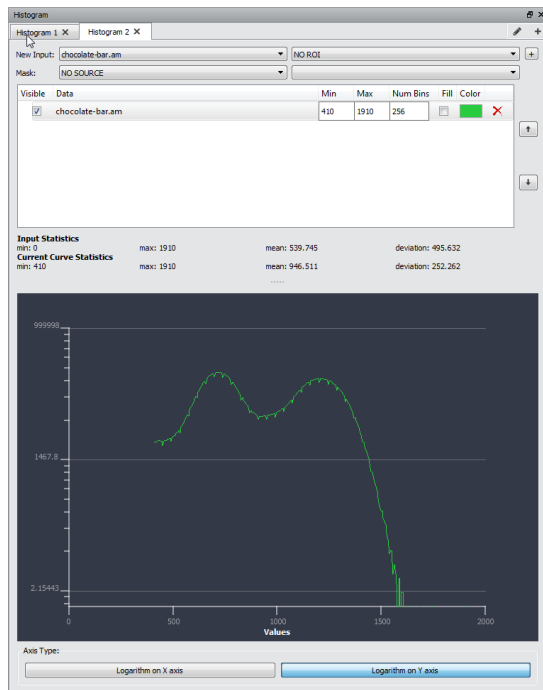
Figure 8.21: New renamed image histogram tab

User Interface Components 413
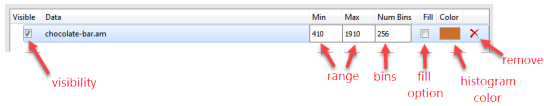
Figure 8.22: Options
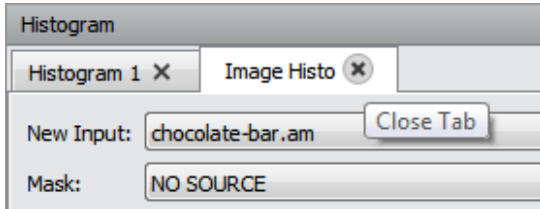
Figure 8.23: Close histogram tab

It is also possible to restrict the histogram of an image within a region of interest defined by a box (ROI). This ability is a way of extracting a subvolume representative of the larger one. Multiple ROI can be created and set, and then the histogram of each can be calculated and displayed to see which one fits best the full volume.

- Load `data/tutorials/chocolat-bar.am`
- Attach a *ROI Box* to the data.
- Attach another *ROI Box* to the data.
- Select the *Histogram Panel*
- In the histogram panel, select *chocolat-bar.am* in the pool down menu of the new input option
- In the histogram panel, select *ROI Box* in the pool down of the ROI input.
- Press the + button.
- In the histogram panel, select *ROI Box 2* in the pool down of the ROI input.
- Press the + button.

8.1.16   Correlation Panel

The correlation panel lets you plot a given measure against another one. Typical inputs include:
- *Spreadsheets*
- *Label analysis*
- *Spatial graph*
- *Pore network models*
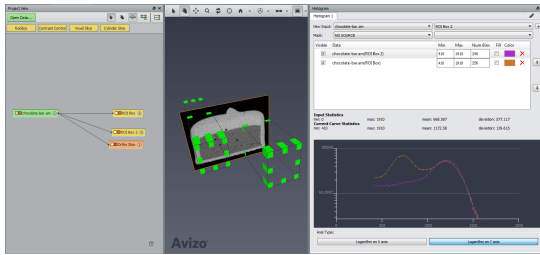
Figure 8.24: Multiple histogram in various ROI
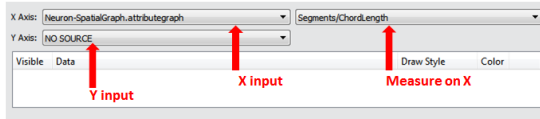
Figure 8.25: Correlation panel example

Here is an example of use:

- Load the *spatial graph* `data/tutorials/neuron/Neuron-SpatialGraph.am`.
- Attach a *Spatial Graph Statistics* module to `Neuron-SpatialGraph.am`.
- Select the output `Spatial Graph` in the *Spatial Graph Statistics* module and click on Apply.
- The data `Neuron-SpatialGraph.attributgraph` is created.
- Select the *Correlation Panel*.
- In the correlation panel, select `Neuron-SpatialGraph.attributgraph` in the pool down menu of the X axis option.
- A new menu will appear next to the X axis input, listing the list of measure that can be selected, as well as a new source input below the first X axis input. This new input is for the Y axis.
- Select *Segments/ChordLength* in the menu next to X axis input.

- Select `data/tutorials/neuron/Neuron-SpatialGraph.attributgraph` in the pool down menu of the Y axis.
- Select *Segment/CurvedLength* in the menu next to Y axis input.
- Click on + in the *Correlation Panel*.

- The *Correlation Panel* plots *CurvedLength spatial graph* at a given *ChordLength* for each segment of `Neuron-SpatialGraph.attributgraph`.
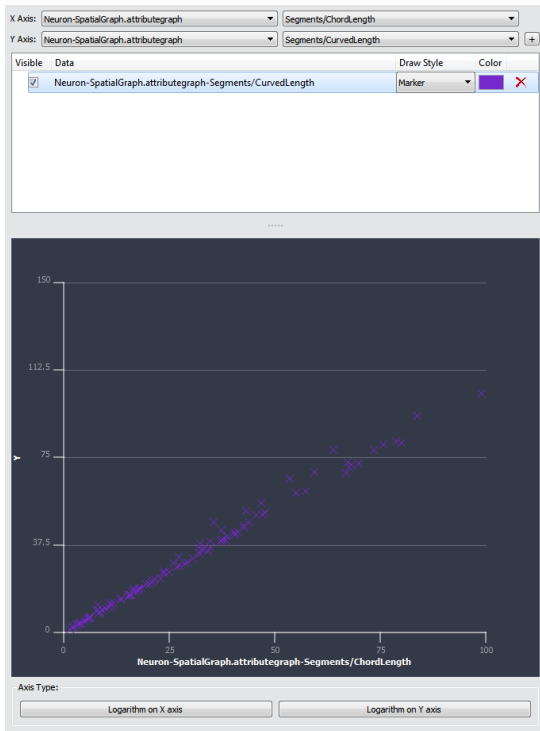
Figure 8.26: Correlation panel example

- In this exemple, the *Correlation Panel* helps to conclude majority `data/tutorials/neuron/Neuron-SpatialGraph.attributgraph` segments are straight lines because the repartition is overall linear.

8.1.17 File Dialog

The *File Dialog* is the user interface component for importing and exporting data into and out of Avizo. It is used in several places in Avizo, most prominently by the *Open Data*, *Save Data As*, *Export Data As*, and *Save Project* items of the main window's *File Menu*.

Most file formats supported by Avizo will be recognized automatically, either by analyzing the file header or by looking at the file name extension. A list of all *supported file formats* is contained in the reference section of this manual. You may manually set the format of a file by means of the dialog's popup menu (see below).

Platform considerations
The dialog uses native dialogs under Microsoft Windows operating systems while it uses specific dialogs under Linux and Mac OS X. Those specific dialogs may differ from the system native dialogs. On Mac OS X, to open a folder, choose an item in the "Look in" combo box. If the desired folder does not appear in the list (*/Volumes* for ex.), type the path directly in the "File name" field and press `[Enter]`.
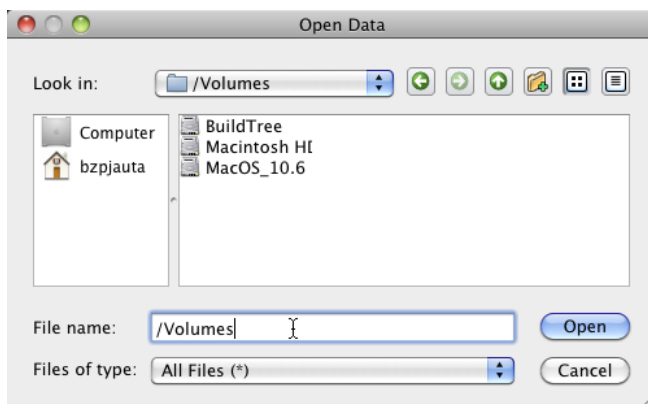
Figure 8.27: The Avizo File dialog on Mac OS X.

8.1.18 Job Dialog

Certain time-consuming operations in Avizo can be performed in batch mode. For this purpose, Avizo provides a job queue, where jobs like generation of a tetrahedral grid can be submitted. You can inspect

the current status of the job queue, start and delete jobs from the queue by selecting *Dialogs > Jobs* from Avizo's *Edit Menu.* This will bring up the *Job Dialog*.

The current list of jobs of a user is shown in the upper part of the *Job Dialog*. For each job, a short description is displayed, as well as the time when the job has been submitted and the current state of the job. A job may be waiting for execution, running, finished, or it may have been killed.

Note that Avizo uses a network interface to communicate with a batch job, which may be blocked by your firewall.

The job directory

For each job, a temporary directory is created containing any required input data, scripts, state information, and log files. On Unix systems, this directory is created at the location specified by the environment variable TMPDIR. If no such variable exists, /tmp is used. On Windows systems, the default temporary directory is used. Typically, this will be C:/TEMP or %USERPROFILE%/AppData/Local/Temp.

Controlling the job queue

A job's state may be manipulated using the action buttons shown above the job list. In order to start the job queue, select the first job waiting for execution and then press the *Start* button. Note that only one job can be executed at a time. In order to kill a running job, select it in the job list and press the *Kill* button. You may delete a job from the job queue using the *Delete* button. When deleting a job, the temporary job directory will be removed as well.

Information about a job

Once you have selected a job in the job queue, more detailed information about it will be displayed in the lower part of the dialog window, notably the state of the job, the temporary job directory, the submit time, the time when the job has been started, the run time, and the name of the command to be executed. Any console output of a running job will be redirected to a log file located in the temporary job directory. Once such a log file exists and has non-zero size, you may inspect it by pushing the *View output* button.

network issues

Job success and failure notifications require that Avizo open a TCP/IP port. Depending on your network configuration, it is possible that you will not be able to receive these notifications. In these cases, contact your system administrator or change your firewall settings, if you have sufficient permissions.
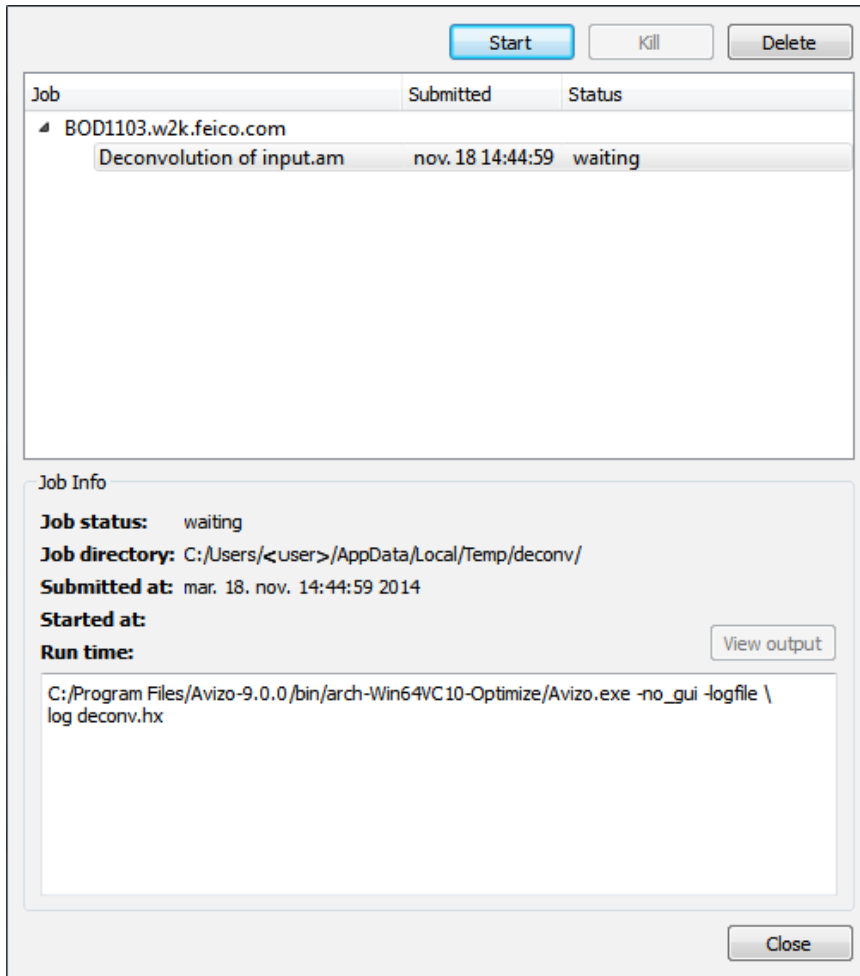
Figure 8.28: The *Job Dialog* lets you start, stop, examine, and delete batch jobs.

Commands

`job submit` *cmd info [tmpdir]*
Submits a new job to the job queue. *cmd* specifies the command to be executed. *info* specifies the info string displayed in the *Job Dialog*. *tmpdir* specifies the temporary job directory. If this argument is omitted a temporary job directory is created by Avizo itself. In any case, the directory will be automatically deleted when the job is removed from the job queue. Example: `job submit "clock.exe" "Test job"`

`job run`
Starts the first job in job queue pending for execution. When a job is finished, execution of the next job in the queue starts automatically, thus all jobs in the queue will be executed consecutively by `job run`.

8.1.19   Preferences Dialog

The *Preferences Dialog* allows you to adjust certain global settings of Avizo. The preferences are stored in a permanent fashion on a per-user basis. The dialog contains a tab bar with several tabs.

The first tab, *General*, is dedicated to general preferences like *Project View* customization and saving options, web news activation/deactivation, the number of recent files or projects displayed, etc.

The second tab, *Layout*, affects the layout of the user interface.

The third, *On Exit*, controls what conditions Avizo checks in the projects when it exits.

The *Molecules* tab allows you to specify options for handling molecular data.

The *LDA* tab allows you to specify options for out-of-core data.

The *Segmentation* tab allows you to specify options for the Segmentation Editor.

You can set the rendering options in the *Rendering* tab.

You can specify the number of CPU threads for compute modules in the *Performance* tab.

You can set the network options in the *Network* tab.

The *Units* tab provides options related to unit management and customization of the way the units will be used in Avizo.

The automatic computation of Intensity Range Partitioning can be activated from the *Range Partitioning* tab.

Note that this last tab will be only available if you have an Avizo license, except if you launch Avizo in its Avizo Lite Edition.

In the *Metrology* tab, you can configure some parameters used in the *Metrology Workroom*.

In the *Recipes* tab, you can configure some parameters used in the *Recipes Workroom*.

In the *Auto-Display* tab, you can configure the Automatic Display feature in Avizo. The aim of this

feature is to automatically connect a display module when a new data is added in Avizo.

8.1.19.1   The General Tab

Project Modules and Data Objects

2-pass firing algorithm
If set, a slightly more complex firing algorithm is used which ensures that down-stream modules connected to an up-stream object via multiple paths are only fired once if the up-stream object changes. The default is on.

Auto-select new objects
If set, a new object selected from the popup menu of its parent object is shown automatically in the *Properties Area*. The default is on.

Deselect previously selected objects
This option can only be set if auto-selection is turned on. If set, all objects are deselected before selecting the new object. Otherwise, the new object will be appended at the end of the *Properties Area*. The default is on.

Use Legacy Demo Maker
When checked, this option allows to retrieve the legacy module *Demo Maker* which has been replaced by the *Animation Director*. The *Demo Maker* is available by doing a right-click in the *Project View*, then *Create Object...* and *Animations and Scripts / Demo Maker*.

Draw viewer toggles on icons
If set, small viewer mask toggles are drawn on the icons of data objects and display modules. This allows you to show or hide a module in a viewer without selecting it first. The default is on.

Draw compute indicator
If set, a small red rectangle is drawn inside the icon of a module to indicate that the module is currently working. The default is on.

Save Project

Include unused data objects
If set, all data objects including hidden colormaps are stored in project scripts. When executing such a script, all existing objects are removed first. If not set, only visible data objects and objects which are referenced by others are stored in a project script. When executing the script, hidden data objects are not removed. The default is off.

Include window sizes and positions
If set, window sizes and positions are stored in project scripts. Be careful using this option if you want to send your script to a machine with a different screen resolution. The default is off.

Include background settings
If set, viewer background settings are stored in project scripts.

Overwrite existing files in auto-save

If set, no overwrite check is performed for data objects that need to be saved automatically to create a project script. Otherwise, a unique file name will be chosen. Details about the auto-save feature are described in Section 8.1.1.11. The default is on.

Policy:
The default setting applying to the save project policy can be modified using the combo box with the following choices:

- *Minimize project size:* Only necessary data to restore the project is saved to disk. Data that can be computed will be computed at project loading. Some data that could not be recomputed may be saved to disk if needed. This is the legacy and default behavior.
- *Minimize project computation:* All of the data of the project are saved to disk. No computation will be done at project loading, unless that data cannot be saved in its current state.
- *Always ask:* Always ask what policy to use when saving a new project.

Language

Set program language to
As mentionned, this option allows selecting the language that will be used inside the program.

Online Documentation

Display only available features
Only the available features are displayed in the online documentation.

Preferences and Settings

The first button allows you to restore default preferences and/or default layout and/or to clear the recent documents lists. The second and third buttons allow you to load or save predefined sets of preferences.

Current Profile

Here you could select which profile you wish to launch Avizo:

- *The Geophysics profile*
- *The Climatology profile*

The application must be restarted in order for the option to apply. This group box is available in the Avizo and Avizo Lite Edition editions only.

Maximum Number of Recent Documents

Set here the number of available recent files and projects.

8.1.19.2   The Layout Tab

Windows

These items allow you to configure the layout of the Avizo windows.

Save window layout on exit
The window layout is saved when Avizo is closed. The default is on.

Show viewer in top-level window
The main *Viewer window* will be displayed in a top-level window. This gives you additional flexibility in managing the "real-estate" of your graphics display. For example, on a dual-head display, it can be interesting to display the *Viewer window* on one display and the rest of the Avizo interface on the other. The default is off.

Show DoIt buttons
Some modules have a button, usually labeled "DoIt", to initiate the action of the module. By convenience, these ports are not displayed in the *Properties Area*. This last provides green *Apply* button which is used to initiate the action of all selected objects. If this box is checked, the *DoIt* button will be displayed in the *Properties Area*. The green *Apply* button will still be available for use as well.

Show projection buttons
This option gives you the possibility to view or not the projection buttons displayed in the module header in the *Properties Area*. The default is off.

Enable docking "Help" panel
This option gives you the possibility to dock "*Help*" panel. The default is on.

Tools buttons style
This options allows you to change the way tools buttons are displayed in the *Standard Toolbar*. There are 5 different choices:

- *Tool button icon only:* only the tool button icon is displayed
- *Tool button text only:* only the tool button text is displayed
- *Tool button text beside icon:* the tool button text is displayed beside its icon
- *Tool button text under icon:* the tool button text is displayed under its icon
- *Tool button follow style:* the tool button is styled according to the style of your platform

The default tools buttons style is *Tool button text under icon* for Mac and *Tool button text beside icon* for other platforms.

Finally, you have the choice to *Restore current layout*, and *Save current layout*.

Viewer gadgets

There are two viewer gadgets, a camera trackball and a compass.

The camera trackball, used for constrained rotation of the camera about the screen-aligned X, Y, or Z axes, is described in Section 8.1.12.

The 3D compass indicates the direction from which the camera is viewing the scene. See Section 8.1.12 for more details on the compass.

Click on the tab of the gadget whose attributes you wish to control, then set its attributes as described below.

Show the camera trackball / Show the compass
This toggle controls the visibility of the trackball/compass. The default is off.

Auto-hide the camera trackball / Auto-hide the compass
When this box is checked, the trackball/compass is only displayed while the mouse is within the trackball/compass display area. It is hidden as soon as the mouse moves outside the trackball/compass display area. The default is on for the trackball and off for the compass. This item is not active if the *Show the trackball/compass* item is off.

Camera trackball position / Compass position
This menu allows you to specify which corner of the viewer window in which to display the trackball/compass. The default is *Lower right* for the trackball, and *Lower left* for the compass. This item is not active if the *Show the trackball/compass* item is off.

Project View

These preferences allow you to customize some options linked to the *Project View*.

Group by display/compute/data in tree view
This option allows the user to change the way objects are organised in the *Tree View*. By default, objects are organized according to their dependencies/connections to other objects. For example, a *Bounding Box* module will be displayed under the data object to which it is connected. By checking this option, objects will be organized by type. With the previous example, the *Bounding Box* module will be displayed under the *Display* category.

Show port interconnection in Project Graph View
When checked, the interconnection between objects and ports will be displayed as white lines connecting objects in the *Graph View*. The default is off.

Show colormaps connected to objects
When checked, *Colormap* objects will be automatically shown in the *Graph View* when objects are connected to these lasts. By default, *Colormap* objects are always hidden in *Graph View*. The default is off.

Show histogram in background
When checked, the data's histogram is displayed in the colormap editor's background and in the background of some ports. The default is off.

Glue attached display modules to data object
If you enable this functionality, all created display modules will be tight with their associated data.

Advanced mode of modules in Properties Area by default
When checked, advanced properties of modules are shown by default.

Preview
Allow to hide or to change the size of the data preview displayed in the *Properties Area*.

8.1.19.3   The On Exit Tab

These options allow you to control what conditions are checked for in the projects when Avizo exits.

8.1.19.4   The Molecules Tab

Color Schemes

These check boxes allow you to chose alternate color schemes: CPK for atoms, and RasMol for amino acids.

Selection Info

These items control how much information is printed into the console when parts of a molecule are selected. Activate *Molecule name* if the name of the molecule should be printed. Activate *Group name* if the name of the selected group should be printed. If you activate *Group attributes*, all attributes of the selected group are printed. If *Explicit attributes* is activated, the printed attributes are restricted to those explicitly named in the corresponding text field.

Atom Expressions

ID case sensitive
Specifies if atom identifiers are case sensitive or not.

8.1.19.5   The LDA Tab

Conversion

Out-of-core threshold
Specifies the size above which data sets will be treated as out-of-core data.

Compression
Specifies the type of data compression. It enables generation of smaller LDA files, however it could be a little bit slower at the loading stage because of the decompression process.

Sampling
Specifies the algorithm of data sampling. Available options are :

- *Sharp* to use decimation algorithm (one voxel out of two).
- *Average* to use weighted average algorithm: voxels of tile of resolution N+1 are built from the average of the 6 neighbors from resolution N and the current voxel value weighted by n.

Tile Size
Sets the size of the tiles to be compressed.

Rendering Quality

Main Memory Amount
Sets the maximum allowed main memory in MB (megabytes) for all the out-of-core and in memory

data sets. Increasing this value enables the use of higher resolutions when loading very large files.

Video Memory Amount
Sets the maximum allowed texture memory in MB (megabytes) for all the out-of-core and in memory data sets. Increasing this value allows visualization at higher resolutions for very large files.

Loading Priority
Specifies whether Avizo should load slices before volumes when running out of memory (Tcl command *thePrefDialog setLDAGeometryPriority* and *thePrefDialog enableLDAGeometryPriority*). Move the slider on the left to increase the resolution of the slices (it will decrease the volume rendering resolution) and move the slider on the right to increase the resolution of the volume rendering (it will decrease the slice rendering resolution).

Options

Viewpoint Refinement
If set, refinement depends on the viewpoint.

View Culling
If set, refinement takes place only in the view frustum. Note that View Culling setting is ignored with XScreen immersive configurations because of incompatibilities.

Move Low Resolution
If set, ortho slices and slices are computed in half resolution when moving.

Loading policy
Sets loading behavior. If *No Interaction* is selected, the asynchronous loading thread will only load when the user does not interact with the scene. If *Always* is selected, loading occurs as long as there is something to load. If *Never* is selected, no loading occurs. The default is *No Interaction*.

8.1.19.6   The Segmentation Tab

3D Draw Style
This option lets you choose the material draw style used in the 3D viewer.

Selection Draw Style
This option lets you choose the draw style used to highlight the voxels selection.

Labels Draw Style
This option lets you choose the default draw style used to render a material in the material list. Note that you can also individually change the draw style for a particular material in the *Segmentation Editor*.

See the *Segmentation Editor* documentation for more details.

8.1.19.7   The Rendering Tab

Quality

Use high precision framebuffer
This option enables high-quality frame buffers, yielding better volume rendering quality with highly transparent colormaps.

This is useful in the case of multiple overlapped volumes (or if volume raycasting is explicitly disabled). If volumes do not overlap, or if there is only one volume displayed at a time, there is no benefit to enable this mode. However, this is incompatible with some other rendering features, such as antialiasing or shadowing, so it is disabled by default.

Simplified rendering during interaction
This option activates a set of rendering techniques to speed the interactions. This option has some effect only with a small set of display modules (e.g., *Isosurface*).

Raycasted spheres
This rendering option brings a huge improvement in terms of quality and performance in the rendering of spheres, such as in point clouds. This option has, for example, effect on the *Point Cloud View* module used with the *Plates* option activated.
Limitations: this option is not activated by default because of two limitations that should be solved in a next release:

- picking is not available for spheres in this mode
- under some specific circumstances, the rendering can be slow

Note: This option is not available on a Mac OS X platform.

Shadowing

If enabled, shadows will be generated. You can also select the default behavior, the intensity, and the quality of the shadows. If this option is enabled, some visualization modules show the **Shadow** button, through which you can directly specify the object behavior.

Misc

Size of handles for ROI draggers
This option allows specifying the size of the displayed handles when a ROI Box is used.

8.1.19.8   The Performance Tab

CPU

Here you can decide to specify the maximum number of threads for compute modules or to let Avizo automatically choose the number of threads.

8.1.19.9   The Network Tab

Connections

By checking *Open listening port*, a socket will be opened at the port indicated by *Port*.

In the *Outgoing Connections* section, the *Host* and *Port* are the default host and port used by the *app send* command. See section *Application Commands*.

Email Notification

By selecting the *Email Notification* option, an email will be sent when an Avizo Module terminates a long computation. The following settings must be defined:

- Sender Address: The sender address displayed in your email client.
- Receiver Address: The address the email will be sent to.
- Email Server: The SMTP server that will be used to send the email. It should not require an authentification.
- Email Server Port: The port used by SMTP on the Email Server (usually 25).
- Notification Minimum Time: The email will be sent when computation time exceeds the duration defined here (expressed in minutes).

Web News

Do not show news
Web News will not be displayed at Avizo startup, if checked. The default is off. Web News are displayed in the middle of the welcome workroom.

8.1.19.10 The Units Tab

Unit Management

Select *None* to deactivate the unit management or *Spatial information* to activate it and use coordinates and angle units information.

Show units dialog when loading data
This checkbox activates the units editor dialog when loading spatial data objects.

Automatically determine working units
When this checkbox is selected, Avizo automatically determines the working coordinate units.

Lock display units on working units
Setting this checkbox ensures that the units displayed in the interface components are the same as the one used internally by Avizo.

When the coordinate units are unknown at data loading
Thanks to a radio box, this option allows selecting a behavior when a data with unknown units are loaded. Available options are :

- *Show Units Editor dialog*
- *Use "unit" as default coordinate units* where an unit can be selected as default coordinate units.

See the description of *available options linked with unit management*, the *Units* chapter and the *Units*

Editor documentation for more details.

8.1.19.11 The Range Partitioning Tab

Note that this tab will be only available if you have an Avizo license. Even with an Avizo license, it will be unavailable if you launch Avizo in its Avizo Lite Edition.

Automatic Intensity Range Partitioning at load
When enabled, activates automatic Intensity Range Partitioning on data loaded and generated by Avizo. Then you can choose to impose the number of regions that must be found or to let the Intensity Range Partitioning algorithm detect it automatically.

Use only for 3D boundaries displays and segmentation thresholds
When enabled, the data window computed by the Intensity Range Partitioning will only be applied for initializing the colormap range of 3D display modules such as Volume Rendering and for intensity related ports such as threshold. Any other display will be initialized with the data min-max range. When disabled, all display modules will use the data window computed by the Intensity Range Partitioning to set the initial range of the display module.

See the *Intensity Range Partitioning editor* documentation for more details.

8.1.19.12 The Metrology Tab

Fitting Options
Fitting method: The *fitting method* used by default to do the fit.

Auto-Adjustment Refit:
Auto-Adjustment Refit is an algorithm used to improve support points distribution and fitting accuracy. Its computation is easily customizable through input parameters (see section *Auto-Adjustment Refit*).

RANSAC
RANSAC is an algorithm used for filtering the input point cloud used to fit a geometry. It selects points that are near theoretical model of the geometry. Its sensibility of selection is managed by its parameters (see section *RANSAC Parameters*). For example, if five points have been picked to fit a circle and one of them makes the circle extend beyond the data, it is not considered to fit. Only the others are finally considered to fit the circle.

Location Options
Geometries Location options:

- *Enable automatic location from the Geometries panel:* The current geometry selected in the *Geometries* panel is automatically located. The viewers are automatically set to display the slices where the current selected geometry appears.

- *Enable automatic location from the Measures panel:* The latest geometry selected in the combo box of the Measures panel is automatically located. The viewers are automatically set to display the slices where the current selected geometry appears.

- *Enable automatic location from the Local Coordinate System wizard:* The latest geometry selected in the combo box of the Local Coordinates System wizard is automatically located. The viewers are automatically set to display the slices where the current selected geometry appears.

Measures Location Options

- *Enable automatic location from the Measures panel:* The current measure selected in the Measures panel is automatically located. The viewers are automatically set to display the slices where the current selected measure appears.

Decimal Display
Option to customize the number of decimals displayed in the Metrology Workroom.

8.1.19.13 The Recipes Tab

Directories
These parameters control the default directories when looking for existing recipes and saving the recipes results.
Note: The recipes results directory is automatically determined from the recipes directory.

8.1.19.14 The Auto-Display Tab

Automatic Display

Check the box to activate the Automatic Display, uncheck to deactivate it.

Automatically connect a display module:
These three buttons allow you to adapt the automatic display behavior to your working habits.

Select the display module to auto-connect to:
Allows you to customize each association between data type and their associated display module.

See the chapter *Automatic Display* for more details.

8.1.20   Snapshot Dialog

The *Snapshot Dialog* provides the user interface of the viewer's snapshot facility. You get the dialog by clicking on the camera icon in the *Viewer window* toolbar.
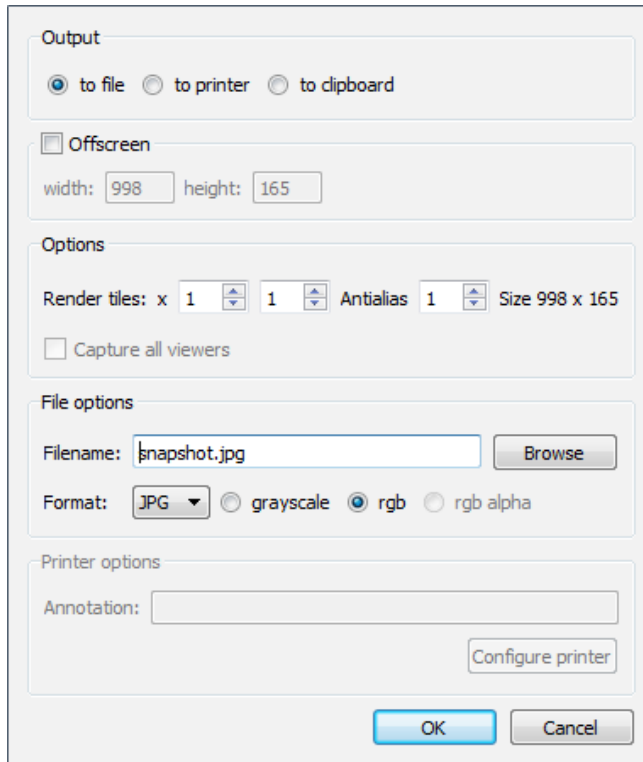
Figure 8.29: The Snapshot Dialog allows you to save or print the contents of a *Viewer window*.

- **Output:** Specifies the output device. With *to file*, the grabbed image is saved to a file, with *to printer*, the image is sent directly to the printer, and with *to clipboard*, it is sent to the clipboard. In the *to printer* mode, you must first select and configure a printer by pressing the *Configure printer* button. In addition, you may enter an arbitrary text string which is printed as an annotation text below the snapshot image.

- **Offscreen:** Lets you grab images larger than the actual screen size. When this option is checked, the output dimensions can be specified in the *width* and *height* text fields. If one of the offscreen dimension is bigger than 4096, the tiles algorithm is automatically used. Notice that the offscreen option overwrites the tiles option.

- **Render tiles:** Use this option to render snapshots of virtually unlimited resolution (e.g., for high quality printouts). In this mode, the scene is divided into n x m tiles where n and m can be entered into the adjacent text fields. Then the camera position is set such that each tile fills

the current viewer and a snapshot is taken. Finally, the tiles are internally merged to a single image and sent to the device specified in the *Output* port. Note that [*Annotations—Colormap displays— Plot Viewer—SpreadSheet Viewer— Scale*] will change their size when using tiling with snapshots.

- **Antialias:** Use this option to render snapshots with antialiasing enabled. In this mode, the antialiasing is performed by using the tiling rendering followed by a downscale. The Antialias factor determines the degree of antialiasing.

- **Capture all viewers:** This option cancels the *Render tiles* option and allows creating one snapshot of all viewers when several viewers are used.

- **Filename:** Lets you specify the filename, if the *to file* option is set. The *Browse* button allows you to browse to a desired location within the filesystem.

- **Format:** The format option lets you select the file format to be produced for file output. The following formats are supported: TIFF (`.tif, .tiff`), SGI (`.rgb, .sgi, .bw`), JPG (`.jpg, .jpeg`), PPM (`.pgm, .ppm`), BMP (`.bmp`), PNG (`.png`), DCM (`.dcm`), Encapsulated PostScript (`.eps`), JPEG2000 (`.jp2`) and PDF (`.pdf`). In addition, this port offers three radio buttons to choose between *grayscale*, *rgb*, and *rgb alpha* type of raster images. If *rgb alpha* option is set, images are produced such that the viewer background is assigned to the alpha channel. This option is not available for file formats that do not support an alpha channel.

- **Auto-save project:** Use this option to save the project in addition to the snapshot. If the option is checked, the prohect will be saved at the same location as the snapshot specified by filename, and ending with .hx. (i.e snapshot.jpg.hx)

Warning: it is not possible to generate a snapshot bigger than 2GB. The picture size depends on:

- *Width*, *Height* and *Antialias* if *Offscreen* is selected
- *Render tiles x*, *Render tiles y* and *Antialias* otherwise.

If the snapshot size exceeds 2GB, a warning tooltip is displayed and the value leading to the bigger size is not validated.

Commands

snapshot [*options*] [*filename*] [*filename2 (for stereo mode only)*]

Options:

- `-stereo`
  If this option is used, the stereo mode image is created. In this case, *filename2* file can be

used to specify where the second image of the stereo image is stored.

- `-alpha`
  If this option is used, the snapshot image is created with a transparent background.

- `-tiled nx ny`
  If this option is used tiled rendering is used with *nx* number of tiles in the horizontal direction, and *ny* number of tiles in the vertical direction.

- `-offscreen [witdh height]`
  If this option is used offscreen rendering is used where *width* and *height* are width and height of rendered image.

8.1.21   System Information Dialog

The system information dialog provides diagnostics information allowing the user or the Avizo support team to better analyze software problems. The dialog contains a tab bar with two pages. The first page lists information about the current CPU. The second page lists information about the current OpenGL graphics driver.

In the lower left part of the dialog, you will find a button *Save Report*. With this button, all information can be written into a text file. In case of a support call, you may be asked to send this text file to the hotline.

8.1.21.1   The CPU Tab

This page displays information about the system on which you are running Avizo. This information can be useful when reporting problems to technical support.

8.1.21.2   The OpenGL Tab

This page displays information about the current OpenGL graphics driver. In particular, a list of available OpenGL extensions is printed. This list allows it to check if certain rendering techniques, like direct volume rendering via 3D textures, are supported on a particular hardware platform or not.

8.1.22   Object Popup

This popup menu lets you attach modules to a specific object. This popup menu contains:

- An explorer used to browse the different categories of modules and data objects.
- A preview panel displaying information about the selected module or data object.
- A search field used to search an item within the categories tree.
- An options button used to perform specific actions on the object (rename, remove...).
- A save button (only on data objects) used to save (or save as, or export as) the data object.
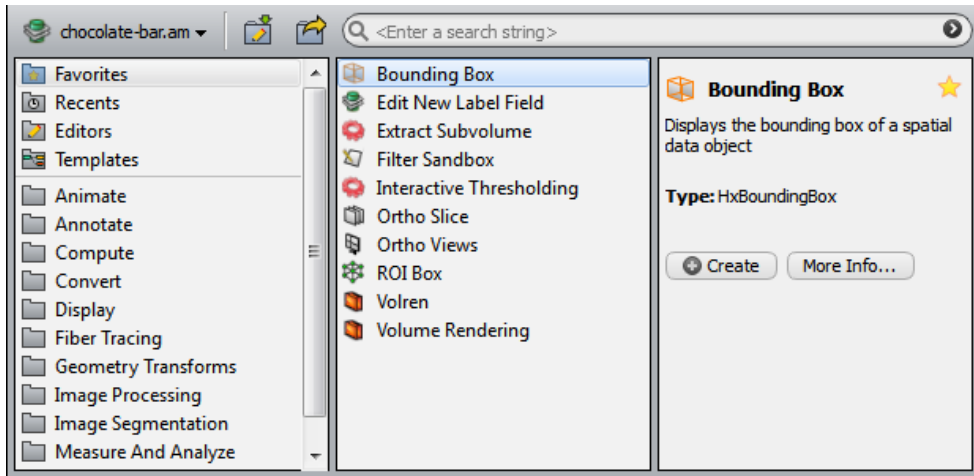
Figure 8.30: The popup menu on chocolate-bar.am.

To create an object like a *Bounding Box*, select the *Annotate* category and, then, double-click (or press [Enter]) on the *Bounding Box* item.

For a quicker access, it is also possible to use the search field and start to enter the "Bounding Box" string. A search will be done on the categories tree to find the modules and data objects whose names begin with the entered string and the search results are automatically displayed within a completion popup. You can instantiate the requested object by clicking on the associated completion result.

8.1.22.1   Explorer

This component is used to browse the categories of modules which can be attached to the object. The categories tree is represented using cascading panels allowing you to navigate through the categories, visualizing the modules hierarchy.
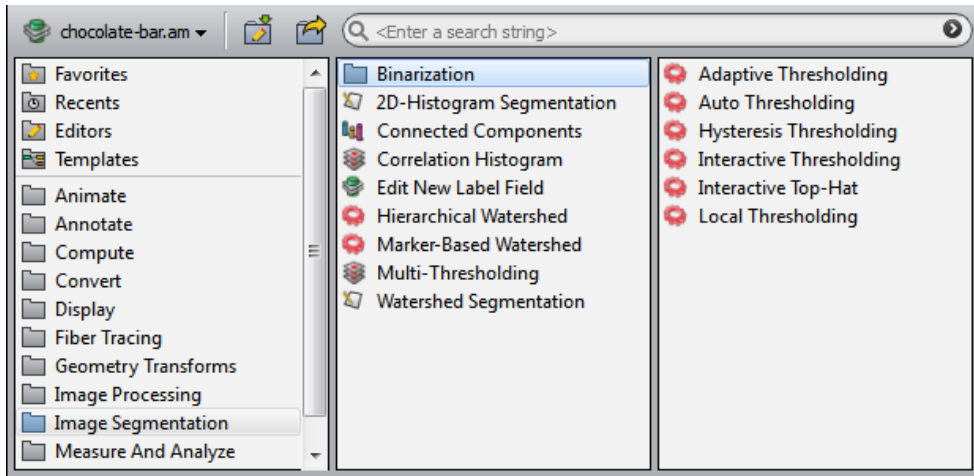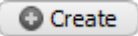
Figure 8.31: The categories explorer on an image.

Once a category is selected (items with a folder 📁 icon), the next panel is automatically updated with the category contents (note that a horizontal scrollbar can be displayed when the sub-categories level exceeds three). Specific categories are listed before the other:

- *Favorites*: lists the modules which have been set as favorites (using the star ⭐ button within the preview panel). This category contains default favorites modules at the first start of Avizo.
- *Recents*: lists the modules which have recently been created on the object. By default, this category is empty but its contents are saved from one Avizo execution to another, allowing you to retrieve the modules created during the last session.
- *Editors*: lists the editors which can be attached on the object. Note that this category is not displayed when no editor is available.
- *Templates*: lists the template modules which can be attached on the object. Note that this category is not displayed when no template module is available.

Once an object item is selected, the preview panel is displayed on the right, displaying information about the selected object. To create the selected object, double-click (or press [Enter]) on the item, or click on the ⊕ Create button within the preview panel.

8.1.22.2 Preview Panel

This component is displayed when an object item is selected within the categories explorer. It provides information about the selected object such as a short description, the object's type and its former name(s).
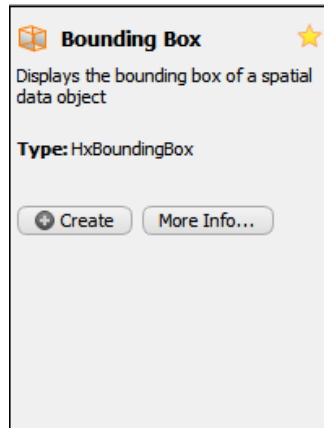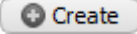
Figure 8.32: The preview panel for *Bounding Box* module.

Thanks to the preview panel, it is possible to:

- Create the object by clicking on the [⊕ Create] button.
- Display the entire object's documentation within the Avizo*Help dialog* by clicking on the [More Info...] button.
- Set/unset the object as favorite by clicking on the star ⭐ button.

8.1.22.3 Search Field

This component is used to search a specific object without navigating through the categories explorer. When entering a search string, the objects whose names contains the entered string will be displayed within a completion popup, allowing you to quickly create an object. Note that, when the search string exceeds 3 characters, an item is displayed at the end of the list of completion results to search the entered string within the Avizo documentation (the results will be displayed within the Avizo*Help dialog*).
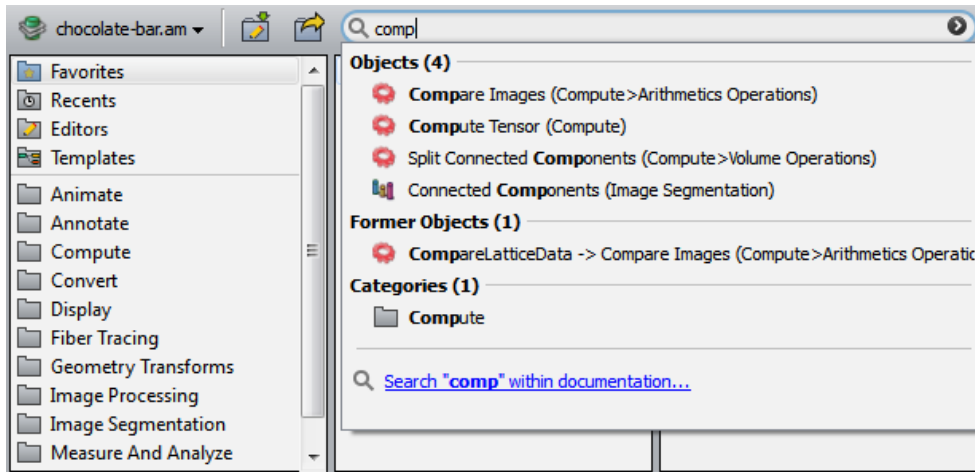
Figure 8.33: The completion results for the "comp" string on chocolate-bar.am.

Once the completion popup is displayed, you can create the requested object by double-clicking on it or pressing [Enter] after selecting it.

By default, the search is performed on:

- The name of the objects.
- The former name(s) of the objects.
- The name of the categories within the explorer.
- The name of the editors.

It is possible to filter the completion results by selecting the search filters via a menu displayed when clicking on the search field arrow ❷ button.
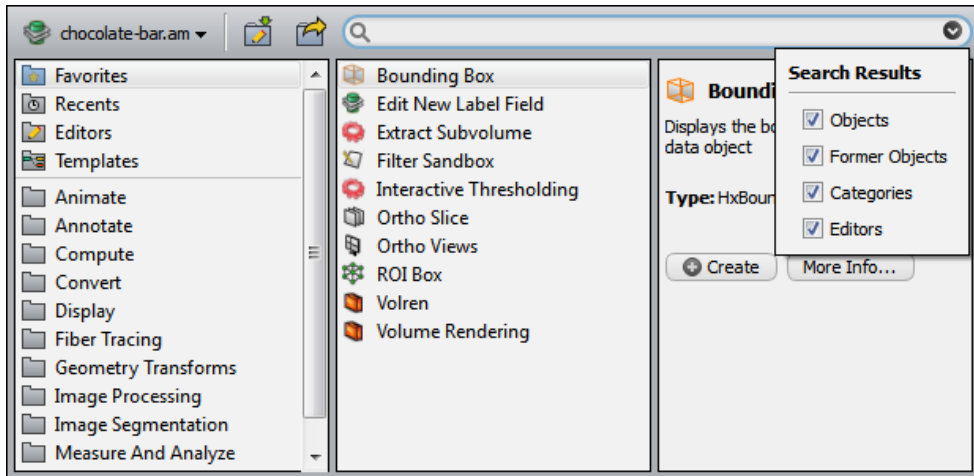
Figure 8.34: The search filters menu.

8.1.22.4 Options Button

When clicking on the object's name, a menu is shown providing different actions on the object:

- *Hide Object*: to hide the object from the Project View (only in Graph View mode).
- *Remove Object*: to delete the object.
- *Duplicate Object*: to create a copy of an object and add it to the Project View.
- *Rename Object...*: to rename the object (will pop up a renaming dialog).
- *Hide All From Viewer But This*: to keep visible in the viewer only the display modules of the selected objects.
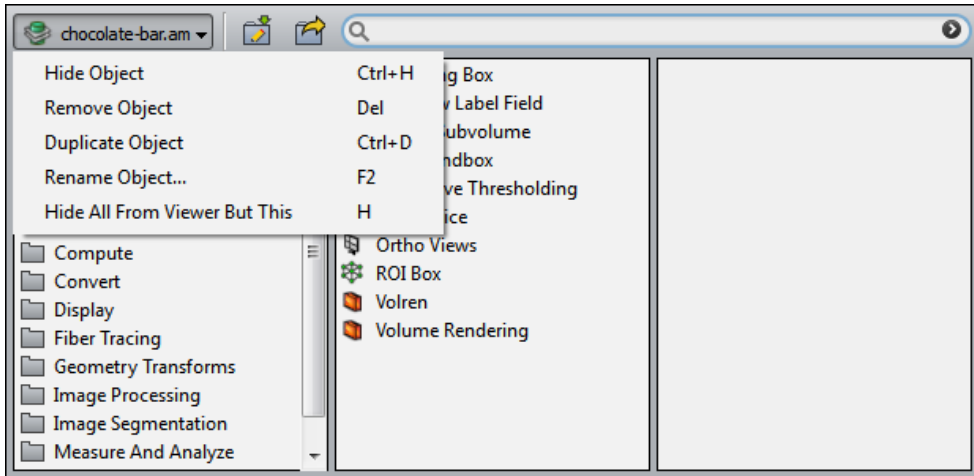
Figure 8.35: The Options button menu.

8.1.22.5   Save/Export Buttons

The first button allows saving the data, the second button allows exporting the data.
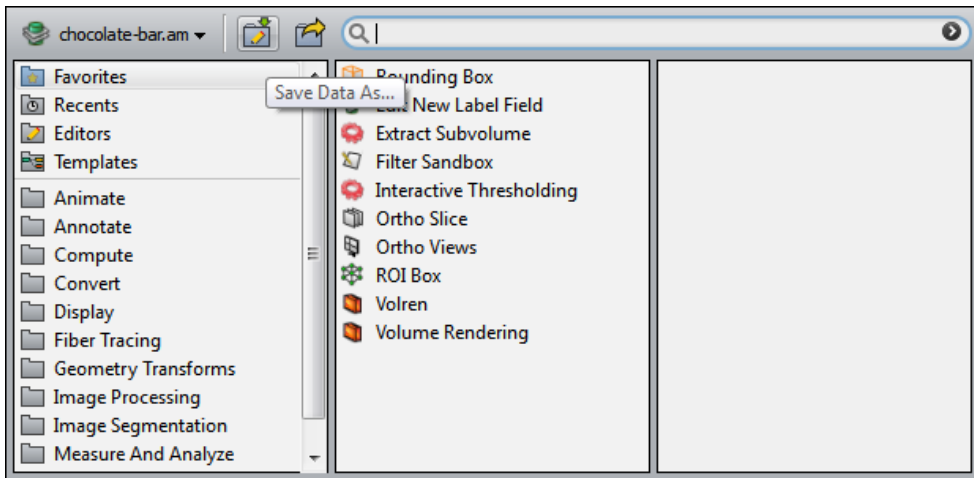
Figure 8.36: The Save and Export buttons.

8.1.23 Create Object Popup

The *Create Object* popup menu lets you create modules or data objects that cannot be accessed via the popup menu of any other object. The *Create Object* popup menu contains:

- An explorer used to browse the different categories of modules and data objects.
- A preview panel displaying information about the selected module or data object.
- A search field used to search an item within the categories tree.
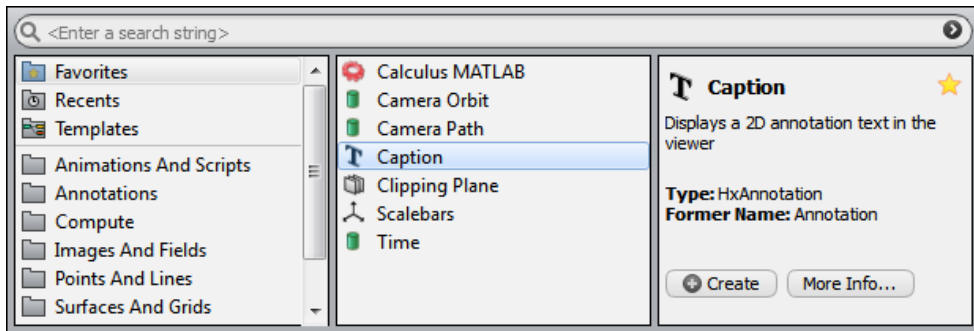
Figure 8.37: The Create Object popup menu.

This popup menu looks and works like the popup menu of objects within the Project View. For more details about the popup menu components, please refer to the *Object Popup* documentation.

To create an object like a *Caption*, select the *Annotations* category and, then, double-click (or press [Enter]) on the *Caption* item.
For a quicker access, it is also possible to use the search field and start to enter the "Caption" string. A search will be done on the categories tree to find the modules and data objects whose names contains the entered string and the search results are automatically displayed within a completion popup. You can instantiate the requested object by clicking on the associated completion result.

The icon of a newly created object usually will not be connected to any other object in the Project View. In order to establish connections later on, use the popup menu over the small white square on the left side of the object's icon.

You can also put in links to scripts in the *Create Object* popup menu. Details are defined in Section 9.4.6 (Configuring popup menus).

8.2   General Concepts

This section contains some general comments on how data objects are organized, classified and managed in Avizo. In particular, the following topics are discussed:

- *Avizo Class Structure*
- *Scalar and Vector Fields*
- *Coordinates and Grids*
- *Surface Data*
- *Vertex Set*
- *Transformations*
- *Data parameters*
- *Shadowing*
- *Units in Avizo*
- *Automatic Display in Avizo*
- *Workroom Concept*

8.2.1   Class Structure

In this section we discuss the object-oriented design of Avizo in a little more detail. You already know that data objects, e.g., gray level image data or vector field sets, appear as separate icons in the *Project View*. You also know that there are certain display modules which can be used to visualize the data objects. While some modules can be connected to many different data objects, e.g., the *Bounding Box* module, others cannot, e.g., the *Ortho Slice* module. The latter can only be connected to voxel data or to scalar distributions on voxel grids. The reason is that internally both are represented as a scalar field with uniform Cartesian coordinates. Consequently, the same visualization methods can be applied to both. On the other hand, for example, a volumetric tetrahedral grid model of the object of interest usually looks completely different. But since it is also a 3D data object, the same *Bounding Box* module can be connected to it.

In summary, there are Avizo data objects that might be conceived of different type, but with respect to mathematical structure, applicability of viewing and other processing modules, as well as programming interface design, have many common properties. Obeying principles of object-oriented design, the data types of Avizo are organized in class hierarchies where common properties are attributed to 'higher up' classes and inherited to 'derived' classes, as sub-classes of a class are commonly referred to. Conceptually, each object occurring in Avizo is an instance of a class and each of its predecessors in the hierarchy that the class belongs to. The classes and their hierarchies are defined within Avizo. As the user, you normally deal with instances of classes only. For instance, there is a class called "HxObject" with sub-classes "HxData" and "HxModule". "HxData" comprises the types of data associated with data objects used for modeling the objects of interest, e.g., volumetric tetrahedral grids or surfaces. "HxModule" comprises data types that have been assigned to display and other process-

ing modules, again in accordance with principles of object-oriented design. This is why Avizo's data objects and processing modules are commonly referred to as "objects".

There are also classes in Avizo that are not derived from "HxObject" and constitute other data types, and there are several independent class hierarchies. e.g., there is a class called "HxPort" from which all classes supporting the operation and display of interface control elements are derived (see section *Properties Area* and the *List of Ports* in the index section of the User's Guide).

A single class hierarchy is usually figured as an upside-down tree, i.e., with the root at the top. Thus the *data* class tree is the one to which the information as to which processing module is applicable to which data object is hooked. Its classes reflect the mathematical structure of the object models supported by Avizo. For example, scalar fields and vector fields are such structures and derived from a common "field" class which represents a mapping $R^3 \rightarrow R^n$. Deriving a sub-class from this base class requires a value to be specified for n.

At the same time, fields defined on Cartesian grids are distinguished from fields defined on tetrahedral grids, i.e., this distinction is part of the classification scheme that gives rise to branches in the *data* class subtree. You will learn more about the *data* class hierarchy in the next section of this chapter. In the second section, we discuss how some data types frequently used for various visualization tasks fit into it.

Internally, all class names begin with a prefix *Hx*. However, you don't have to remember these names unless you want to use the command shell to create objects. For example, a bounding box is usually created by choosing the *Bounding Box* item from the popup menu of a data object that is to be visualized, but you may also create it by typing `create HxBoundingBox` in the command window.

8.2.2 Scalar Field and Vector Fields

The most important fields in Avizo are three-dimensional ones. These fields are defined on a certain domain $\subseteq R^3$. A field can be evaluated at any point inside its domain. If the field is defined on a discrete grid, this usually involves some kind of interpolation.

8.2.2.1 Scalar Fields

A 3D scalar field is a mapping $R^3 \rightarrow R$. The base class of all 3D scalar fields in Avizo is *HxScalarField3*. Various sub-classes represent different ways of defining a scalar field. There are a number of visualization methods for them, for example pseudo-coloring on cutting planes, iso-surfacing, or volume rendering. However, many visualization modules in Avizo rely on a special field representation. Therefore, they can only operate on sub-classes of a general scalar field. Whenever a given geometry is to be pseudo-colored, any kind of scalar field can be used (cf. *Color Wash*, *Tetra Grid View*, *Isosurface*).

The class *HxTetraScalarField3* represents a field which is defined on a tetrahedral grid. On each grid vertex, a scalar value, e.g., a temperature, is defined. Values associated with points inside a tetrahedron are obtained from the four vertex values by linear interpolation. This class does not provide a copy of

the grid itself, instead a reference to the grid is provided. This is indicated in the *Project View* by a line which connects the grid icon and the field icon. As a consequence, a field defined on a tetrahedral grid cannot be loaded into the system if the grid itself is not already present.

The class *HxRegScalarField3* represents a field which is defined on a regular Cartesian grid. Such a grid is organized as a three-dimensional array of nodes. In the most simple case, these nodes are axis-aligned and have equal spacings. The coordinates of such a uniform grid can be obtained from a simple bounding box containing the origin vector and increments for all directions. Stacked coordinates are another example. Here the spacing in z-direction between subsequent slices may be different. In any case, scalar values inside a hexahedral grid cell are obtained from the eight vertex values using trilinear interpolation. While the *Ortho Slice* module can only be used to visualize scalar fields with uniform or stacked coordinates, other modules like *Slice* or *Isosurface* work for all scalar fields with regular coordinates.

Yet another example of a scalar field is the class *HxAnnaScalarField3*. It represents an analytically defined scalar field. To create such a field, select *Images And Fields / Analytic Scalar Field* from the *Project >Create Object...* menu of Avizo's main window. You have to specify a mathematical expression which is used to evaluate the field at each requested position. Up to three other fields can be connected to the object. These can be combined to a new scalar field, even if they are defined on different grids.

8.2.2.2   Vector Fields

As for scalar fields Avizo provides a number of vector field classes, derived from the base classes *HxVectorField3* and *HxComplexVectorField3*. While ordinary vector fields return a three-component vector at each position, complex vector fields return a six-component vector. Complex vector fields are used for encoding stationary electromagnetic wave pattern as required by some applications. Usually complex vector fields are visualized by projecting them into the space of reals using different phase offsets. The *Vectors Slice* module even allows you to animate the phase offset. In this way, a nice impression of the oscillating wave pattern is obtained.

8.2.3   Coordinates and Grids

Avizo, in its Avizo Lite Edition, currently supports two important grid types, namely grids with hexahedral structure (regular grids), and unstructured tetrahedral grids.

In Avizo XWind Extension, *unstructured mixed grids* are supported. For more information about unstructured grids in Avizo XWind Extension, please refer to the Chapter 11 Avizo XWind Extension User's Guide or to the *Unstructured Model* documentation.

Other types of grids like block-structured grids will be added in future releases of Avizo.

8.2.3.1 Regular Grids

A regular grid consists of a three-dimensional array of nodes. Each node may be addressed by an index triple *(i,j,k)*. Regular grids are further distinguished according to the kind of coordinates being used. The most simple case comprises *uniform* coordinates, where all cells are assumed to be rectangular and axis-aligned. Moreover, the grid spacing is constant along each axis. A grid with *stacked* coordinates may be imagined as a stack of uniform 2D slices. However, the distance between neighboring slices in z-direction may vary. In case of *rectilinear* coordinates, the cells are still aligned to the axes, but the grid spacing may vary from cell to cell. Finally, in the case of *curvilinear* coordinates each, node of the grid may have arbitrary coordinates. Grids with curvilinear coordinates are often used in fluid dynamics because they have a simple structure but still allow for accurate modeling of complex shapes like rotor blades or airfoils.

8.2.3.2 Tetrahedral Grids

The *Tetra Grid* class represents a volumetric grid composed of many tetrahedrons. Such grids can generally be used to perform finite-element simulations, e.g., E-field simulations.

A considerable amount of information is maintained in a *Tetra Grid*. For each vertex, a 3D coordinate vector is stored. For each tetrahedron, the indices of its four vertices are stored as well as a number indicating the segment the tetrahedron belongs to as obtained by a segmentation procedure. Beside this fundamental information, a number of additional variables are stored in order for the grid being displayed quickly. In particular, all triangles or faces are stored separately together with six face indices for each tetrahedron. In addition, for each face, pointers to the two tetrahedrons it belongs to are stored. This way the neighborhood information can be obtained efficiently.

When simulating E-fields using the finite-element method, the edges of a grid need to be stored explicitly, because vector or Whitney elements are used. These elements and their corresponding coefficients are defined on a per-edge basis. When a grid is selected, information on the number of its vertices, edges, faces, and tetrahedrons is displayed.

8.2.4 Surface Data

Avizo provides a special-purpose data class for representing triangular surfaces, called *Surface*. This class is documented in more detail in the index section of the user's guide. For the moment, we only mention that the class maintains connectivity information and that it may represent manifold as well as non-manifold topologies.

The surface class provides a rich set of Tcl commands. It is a good example of an Avizo data class that does not simply store information, but allows the user to query and manipulate the data by means of special-purpose methods and interfaces.

8.2.5 Vertex Set

Another example of data abstraction and inheritance is the *Vertex Set* class. Many data objects in Avizo are derived from this class, e.g., landmark sets, molecules, surfaces, or tetrahedral grids. All these objects provide a list of points with x-, y-, and z-coordinates. Other modules which require a list of points as input only need to access the *Vertex Set* base class, but don't need to know the actual type of the data object.

One such example of a generic module operating on *Vertex Set* objects is the *Vertex View* module. This module allows you to visualize vertex positions by drawing dots or little spheres at each point.

8.2.6 Transformations

Data objects in Avizo can be modified using an arbitrary affine transformation. For example, this makes it possible to align two different data objects so that they roughly match each other. Internally, affine transformations are represented by a 4x4 transformation matrix. In particular, a uniform scalar field remains a uniform scalar field, even if it is rotated or sheared. Display modules like *Ortho Slice* still can exploit the simple structure of the uniform field. The possible transformation is automatically applied to any geometry shown in the 3D viewer.

In order to interactively manipulate the transformation matrix, use the *Transform Editor* (documentation is contained in the index section of the user's guide).

Be careful when saving transformed data sets! Most file formats do not allow you to store affine transformations. In this case, you have to apply the current transformation to the data. This can be done using the *applyTransform* Tcl command. In the case of *vertex set objects*, the transformation is applied to all vertices. Old coordinates are replaced by new ones, and the transformation matrix is reset to identity afterwards. After a transformation has been applied to a data set, it cannot easily be unset.

If a transformation is applied to uniform fields, e.g., to 3D image data, the coordinate structure is not changed, i.e., the field remains a uniform one. Instead, the data values are resampled, i.e., the transformed field is evaluated at every vertex of the final regular grid. The bounding box of the resulting grid is modified so that it completely encloses the transformed original box.

8.2.7 Data parameters

An arbitrary number of additional parameters or attributes may be defined for any data object. Data parameters can be interactively added, deleted, or edited using the *Data Parameter Editor*. Parameters are useful, for example, to store certain parameters of a simulation or of an experiment. In this way, the history of a data object can be followed.

There are certain parameters which are interpreted by several Avizo modules. The meaning of these parameters is summarized in the following list:

- *Colormap name*
  This specifies the name of the default colormap used to visualize the data. Some modules automatically search the *Project View* for this colormap and, for example, use it for pseudocoloring.

- *DataWindow minVal maxVal*
  This indicates the preferred data range used for visualizing the data. The *Ortho Slice* module automatically maps values below `minVal` to black and values above `maxVal` to white.

- *LoadCmd cmd*
  This parameter is usually set by import filters when a data object is read. It is used when saving the current project into a file and it allows the object to be restored automatically. Internal use only.

Note that there are many file formats which do not allow parameters to be stored. Therefore, information might get lost when you export the data set in such a format. If in doubt, use the specific *Avizo Format.*

8.2.8 Shadowing

Real time shadow casting is enabled through the *Rendering* tab of the *Preferences Dialog* (accessible via the *Preferences* button in the *Standard Toolbar* or the *Edit > Preferences* menu entry).

Once enabled, most of display modules can cast or receive shadows.
Different modes are available, and switching from one to another is possible by clicking on the shadow icon of a display module

-
  No Shadows: the displayed shape neither casts or receives shadows

-
  Cast Shadows: the displayed shape only casts shadows

-
  Receive shadows: the displayed shape only receives shadows

-
  Cast & Receive shadows: the displayed shape both casts and receives shadows

Restrictions:

At least the *Multi-Texture* and *Texture Environment Combine* OpenGL®extensions must be supported by your graphics board. Otherwise, no shadows will be computed. These extensions are now standard in OpenGL®1.3 and later.

The *Shadow* and *Depth Texture* OpenGL®extensions (standard in OpenGL®1.4) are used, if they are available, and generally improve performance.

Some aliasing artifacts can appear if you zoom in very close to the scene. You can then increase the quality in the *Preferences Dialog*.

Transparent objects are treated as follows, depending on the transparency type:

- *Screen door:* fully compatible
- *Add, Blend:* transparent objects cast a shadow and are shadowed but the shadow intensity doesn't depend on the transparency value (the same shadow is displayed for full transparent shapes and opaque shapes).
- *All other modes:* incompatible with shadowing.

Shadowing may impact performance and is only fully supported by some display modules/modes (such as *Volume Rendering*). Setting the environment variable `AVIZO_FORCE_SHADOW_MAP` enables activating a less restrictive shadowing mode (i.e. more modules supports it but not the *Volume Rendering*)

8.2.9   Units in Avizo

This chapter contains a description of how Avizo can be configured to work with spatial data objects with associated coordinate unit information.

This chapter is organized as follows:

- *Presentation*:
  this section describes globally the unit management in the entire product.

- *How to associate a coordinate unit with a spatial data object*:
  this section describes how you can set a coordinate unit to any spatial data object.

- *How to modify the coordinate unit used for displaying information*:
  this section describes how you can change the coordinate unit which is displayed in the Avizo user interface (ports values, info tags...) and 3D viewers (measuring tools...).

- *Available options linked with unit management*:
  this section describes all the different preferences and options you will be able to modify in order to customize the unit management.

- *Avizo components working with the unit management*:
  this section describes all the Avizo components for which the unit management has been implemented.

8.2.9.1   Presentation

When activating the unit management in Avizo, you will be able to:

- associate a coordinate unit with each spatial data object, retrieving it directly from the data files (depends on readers and file formats) or setting it manually with a *Units Editor*.

- store coordinates values of all loaded spatial data object with the same coordinate units (e.g., meters). The coordinate units storage in Avizo and used internally is called **working units**. Working units could be specified by the user or automatically determined (see *Automatically determine or manually set the working coordinate units*).

- display values related to coordinates information in the Avizo user interface (such as bounding box, length...) in the coordinate units you want.
  The units used to display values in the Avizo user interface are called **display units**.

To know how to activate/deactivate the unit management in Avizo, you can refer to the following section: *Activate/desactivate unit management in Avizo*.

8.2.9.2   How to associate a coordinate unit to a spatial data object

When the unit management is activated in Avizo, a coordinate unit will be assigned to each spatial data object.
This can be done in 2 different ways:

- by the reader used to load the data
- or by a specific *Units Editor*

In the most possible cases, Avizo readers will try to extract the coordinate unit directly from the information stored in the data file. This is the case, for instance, with *Avizo* or MCAD readers (like *IGES*).

If the coordinate unit can't be determined by the reader (the information is missing or not supported by the file format), it will be specified via the *Units Editor*. This editor is launched at data loading but is also accessible after the data has been registered into the Project View, in the same way as the other editors.

Once a coordinate unit has been assigned to the spatial data object, its coordinates values can possibly be converted. This will happen if the specified coordinate unit for this data is not the same as the one used internally by Avizo to store coordinates values for all spatial data objects (i.e., named working

units). In this case, the coordinates values will be converted from the specified coordinate unit to the working one.

Note:

Manually setting a coordinate unit to a spatial data object (via the editor) will mark the object as modified. Indeed, this implies that the original file format of the data does not support coordinate unit information. After associating it a coordinate unit, the data has to be saved in a file format that can support this information. For convenience, you can easily save data in the *Avizo data* file format which saves the coordinate unit in the Parameters section and can be used to store many different data objects.

The information about coordinate units associated with a spatial data is accessible in several places:

- in the *Units Editor*, where you can see and modify the original coordinate unit of a spatial data (i.e., the one specified ad data loading) after that the data has been loaded
- in the *Parameter Editor* on the spatial data where the coordinate unit of the spatial data is specified under a *Units* bundle. In fact, under this bundle are displayed 2 informations:
  - the current (working) coordinate unit of the spatial data stored in the *Coordinates* parameter (the one in which are currently stored the coordinates in memory),
  - the original coordinate unit of the spatial data stored in the *OriginalCoordinates* parameter (the one specified via the *Units Editor* at data loading or edited furtherly).

8.2.9.3 How to modify the coordinate unit used for displaying information

Even if the coordinate unit used to internally store spatial data objects coordinates (i.e., named working unit) is fixed, values related to coordinates information that are displayed in the product user interface can be expressed in any coordinate units.

For example, you can load a spatial data object with coordinates stored in meters (i.e., working unit), connect it a *Bounding Box* module and freely modify the coordinate unit used to display the coordinates (i.e., named display unit) of the corners in the info tags of this module (by selecting millimeters, for instance).

In this case, the displayed coordinates values will be converted from the working unit (in which are internally stored coordinates values) to the display unit (in which are displayed these values in the Avizo user interface).

To specify what coordinate unit is used for displaying coordinates values in the user interface:

- Launch the **Preferences dialog** (menu **Edit** > **Preferences...**)
- Select the **Units** tab
- Select the **Display units** tab
- In the **Spatial information** section, select the display unit you want for coordinates information

The quickest way to modify this display unit is accessible on the main viewer toolbar between Measure button and snapshot button (see *Viewer toolbar description*).

The Display unit button is enable if the option "Lock display units on working units" is unchecked. With this button, you can select the display unit you want for coordinates information not only for the active viewer but for all units displayed in Avizo.

8.2.9.4 Available options linked with unit management

Several options linked with unit management are available and allow you to customize the way you are using units in the product.

These options are the following:

- Activate/deactivate the unit management in Avizo
- Activate/deactivate the units editor dialog when loading spatial data objects
- Automatically determine or manually set the working coordinate unit
- Lock the display coordinate unit on the working one

All these options can be modified via the Avizo Preferences dialog.

To access the preferences linked with unit management:

- Launch the **Preferences dialog** (menu **Edit** > **Preferences...**)
- Select the **Units** tab

Activate/deactivate unit management in Avizo

In order to be compatible with older versions of Avizo, you are not forced to use/set units information in spatial data objects so it's possible to deactivate the unit management in the entire product. In this case, spatial data objects won't contain any unit information and no conversion linked with units will be performed: Avizo will work as in its previous versions, in the same way you are used to.

To activate/deactivate the unit management in the entire product:

- In the **Unit management** section, select **None** to deactivate the unit management or **Spatial information** to activate it and use coordinates and angle units information

Automatically determine or manually set the working coordinate units

When the unit management is activated, some units conversions could occur if you are loading data whose coordinate unit is different from the working unit, used by Avizo to internally store coordinates values.

To limit as much as possible these conversions, we recommand you let Avizo automatically determine the working coordinate units.

In this case, when spatial data is loaded, the working coordinate unit will be automatically set to the

data one.

As you have understood, this behavior prevents the conversion of the coordinate unit of the first loaded spatial data object but conversions will still occur if the following loaded spatial data don't have the same coordinate unit. By contrast, no conversion will be done if you load only data with the same coordinate unit (i.e., equal to the current working unit).

Of course, you are free to activate/deactivate this behavior. If you deactivate it, you can specify yourself the working coordinate unit in which Avizo will store coordinates values. In this case, pay particular attention to units conversion especially when loading data.

Note:
If spatial data has been already loaded, you won't be able to select another working coordinate unit: selecting a custom working coordinate unit is possible only when the Project View does not contain any spatial data.

To automatically determine the working coordinate units:

- Select the **Options** tab
- Toggle on/off the **Automatically determine working units** checkbox

To specify a custom working unit:

- Select the **Working units** tab
- In the **Spatial information** section, select the working unit you want for coordinates or angle information

Lock the display coordinate unit on the working one

In some cases, it could be interesting to always display information related to coordinates values in the Avizo user interface in the same unit as the one used internally to store these coordinates values.

This behavior ensures that values that are displayed in the interface components are the same as the ones manipulated internally by Avizo.

To lock the display coordinate unit on the working one:

- Select the **Options** tab
- Toggle on/off the **Lock display units on working units** checkbox

Activate/deactivate the units editor dialog when loading spatial data objects

As explained before, when the unit management is activated, some readers can launch a specific dialog used to specify the coordinate unit of the loaded spatial data.

This dialog will be launched only if the reader has failed to automatically retrieve the coordinate unit from the information stored in the data file.

Note that you have the option to prevent the dialog from being displayed. In this case, the loaded spatial data object will be assumed to have the default coordinate unit specified in preferences.

Note:

Even if you choose to not automatically display this dialog when data are loading, you still have the possiblity to specify the coordinate units thanks to the units editor available when selecting the spatial data object in the Project View.

To activate the units editor dialog when loading spatial data objects:

- Select the **Options** tab
- Go to **When the coordinate units are unknown at data loading** groupbox
- Toggle on the **Show Units Editor dialog** checkbox

To deactivate the units editor dialog when loading spatial data objects:

- Select the **Options** tab
- Go to **When the coordinate units are unknown at data loading** groupbox
- Select your preferred coordinate units in **Use XXX as default coordinate units**
- Check the associated checkbox if it is not toggle on.

8.2.9.5   Avizo components working with the unit management

Unit management is not yet available for all the components of Avizo. The components (data types, files formats, modules) for which the unit management has been implemented are listed here.

Data types

When unit management is activated in the product, it is possible to assign coordinate units and potentially make units conversion on coordinate values for all data objects for which the type is listed in table 8.1 below.

For all other data types, coordinate values can't be converted so these are always stored internally using original values. In this case, the coordinate units are set to the working coordinate units.

Note:

Data fields (Hexa*Field, Tetra*Field, Reg*Field and Surface*Field) have coordinate units but they are assumed to always be the same as the one specified on the associated data (HexaGrid, TetraGrid, Lattice, Surface). Consequently, it's not possible to explicitly specify the coordinate units of such field data thanks to the *Units Editor.* However, setting or modifying the coordinate units of a data field will automatically update the coordinate units of all attached fields.

Files formats

Some file formats can provide information about the coordinate units of stored data. The following

Data type	Description
Hexa Grid	Unstructured finite-element hexahedral grid and associated fields
. HexaField3	
. HexaScalarField3	
. HexaVectorField3	
. HexaComplexScalarField3	
. HexaComplexVectorField3	
Tetra Grid	Unstructured finite-element tetrahedral grid and associated fields
. TetraField3	
. TetraScalarField3	
. TetraVectorField3	
. TetraComplexScalarField3	
. TetraComplexVectorField3	
Lattice, *LabelLattice3*	Regular 3D data array and associated fields
. RegField3	
. RegScalarField3	
. RegVectorField3	
. RegSym2TensorField3	
. RegComplexScalarField3	
. RegComplexVectorField3	
. RegColorField3	
Surface	Surface data and associated fields
. SurfaceField	
. SurfaceScalarField	
. SurfaceVectorField	
. SurfaceComplexScalarField	
. SurfaceComplexVectorField	
Iv Data	Open Inventor scene graph

Table 8.1: Data Types List

table lists all the file formats for which Avizo readers and writers have been updated to be able to retrieve/save coordinate unit information when reading/writing a file:

File format	Coordinate units retrieved by reader	Coordinate units saved by writer
Avizo data	Y	Y
Surface	Y	Y
CATIA5	Y	N/A (no CATIA5 writer available)
IGES	Y	N/A (no IGES writer available)
STEP	Y	N/A (no STEP writer available)

For all other file formats, Avizo readers and writers currently don't retrieve/save any coordinate unit information when reading/writing a file.

However, for all file formats that generate data of which type supports the unit management (refer to the *Data Types* section), it's possible to associate coordinate units to loaded data via the *Units Editor*.

Refer to the *How to associate coordinate units with a spatial data object* section for more information about how to set and save a coordinate unit information.

Modules

The following modules have been modified to support unit management in case it has been activated:

Module	Unit management support
Local Axes	axis tick values and labels
Bounding Box	coordinates of the lower left front and upper right back corners of the box
ROI Box	minimum and maximum port values
Spline Probe	point coordinates, length value and plot window
Line Probe	point coordinates, length value and plot window
Measurement	length and angle values
Scalebars	axis tick values and labels

8.2.10   Automatic Display in Avizo

This section describes the way to configure and to use the Automatic Display feature in Avizo. The aim of this feature is to automatically connect a display module when a new data is added in Avizo. For example, you can specify that an Ortho Slice is automatically connected to all new images and a Surface View to all new surfaces.

8.2.10.1   Manage and configure Automatic Display in Preferences

Several options linked with automatic display are available and allow you to customize the way you are using it in Avizo. These options are the following:

- Activate/deactivate the automatic display in Avizo
- Select the behavior of automatic display for loaded data and computed ones
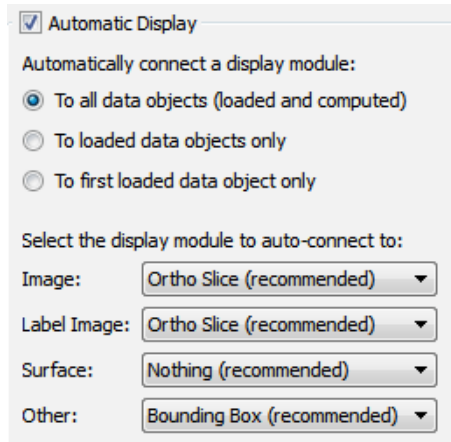
Figure 8.38: Automatic Display preferences
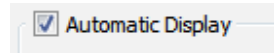
Figure 8.39: Automatic Display activation

- Choose the associations between data type and display module associated with

To reach the automatic display preferences:

- Launch the **Preferences dialog** (menu **Edit** > **Preferences...**)
- Select the **Auto-Display** tab

The contents of this preference tab looks like the following figure:

Activate/deactivate the automatic display in Avizo

The first available option is a global one that allows you to enable (if you check the box) or disable (if you uncheck the box) the automatic display mechanism in Avizo.

Select the functional mode for automatic display mechanism

Three functional modes are available allowing you to adapt the automatic display behavior to your working habits.

You can activate automatic display to:

- All data added to the project view, i.e., the loaded ones and the computed ones

General Concepts
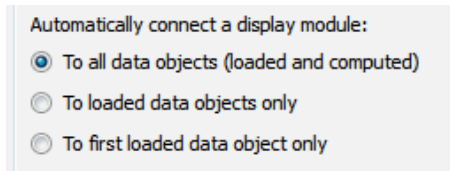
Figure 8.40: Automatic Display mode selection
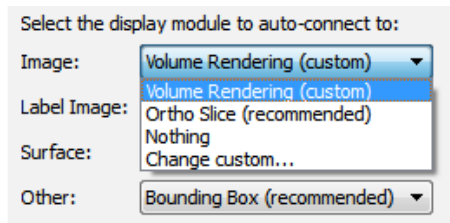
Figure 8.41: Automatic Display associated module selection

- Loaded data only, i.e., computed data will never be automatically connected to a display module
- First loaded data only, i.e., only the first data loaded after the Avizo opening will be connected automatically to a display module

Association management

The last section of the Automatic Display preference tab is the associations between data types and display modules to auto-connect with. Currently, the four main data types used in Avizo are available:

- **Image**
- **Label Image**
- **Surface**
- **Other**, i.e., all spatial data different from images, label images or surfaces.

To modify the predefined associations, you can click on the associated module to drop down a menu as follows:

This menu lets you choose between four options:

- A **custom** module: This user defined module is selected by the **Change Custom...** entry.
- A **recommended** module: This module is selected by default. It has been chosen as the answer to the most frequently usages and because of its low computation time.
- **Nothing**: This option allows you to use the automatic display, but not with all data type.
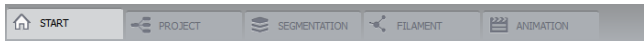
Figure 8.42: The Workroom Toolbar.

- A way to **change the custom choice**: Click on this menu entry to open the *object explorer* allowing you to change the custom choice.

8.2.10.2 Activate/Deactivate Automatic Display in the Project View

An easy way to enable/disable the automatic display mechanism is also available in the *Project View*. It consists in a toggle button with two states. If the button is down, automatic connection is enabled, disabled otherwise.

8.2.11 Workroom Concept

Beside visualization modules, the user interface enable an easier and faster swap between application areas. With the user-friendly "workroom" concept, several dedicated components are accessible as independent applications assembled together into the general Avizo platform. Each component provides its own specific interface, its dedicated tools and visualization options. Since switching between different components takes place through a simple toolbar, the user can explore and approach complex data sets using dedicated tools in the workrooms, and keep a clear overview in the general purpose framework. To access a workroom, a toolbar bar is available in the upper part of the main window.

The following components are currently workrooms:

- Start Page: Start a project, load data, learn about Avizo.
- *Project View*: Open data, connect modules, open and save projects. Pressing "Home" key directly opens this workroom.
- *Segmentation Editor*: Interactively segment 3D image data.
- *Filament Editor*: Automatically and interactively extract, analyze, and quantify filamentous structures in 3D image data (see tutorial: *Filament Editor*).
- *Animation Director*: Create animations from the current project and record a movie file (see tutorial: *The Animation Director*).
- *Metrology*: Test plan creation, geometry fitting, view alignment, and part dimensioning.
- *Recipes*: Recipes creation, customizable workflow automation.
- *Reporting*: Report generation, export of analysis to HTML and PDF.

Chapter 9

Automating, Customizing, Extending

9.1 Template Projects

This section describes the usage of *Template Projects*

9.1.1 Template Projects Description

Template projects can be used to ease repetitive tasks on sets of similar data. A template project is a copy of an original project that can be re-applied on other data of the same type.

9.1.1.1 How to save a template project

To create a template project, choose *Save Project As Template* from the *File* menu. An input selection dialog appears and lists all the possible template inputs (all the current data objects). A template input stands for a data set that must be supplied when the template is executed. You can change the label for each selected template input. This label should be general and meaningful since it will be displayed during template execution. The default label is the original data object name. Note: Unused data objects are filtered by default, but you can include them in the template project by selecting the *Include unused data* option.

If the template contains exactly one input, a dialog will ask if you wish to associate the template with data of this type. If you click OK, then the template will be available in the right-click menu (*Templates* submenu) for all data objects of the same data type.

Finally, a file dialog will appear to name the output file. The file name is also the name of the template, i.e., the name that will appear in the Templates menu. Built-in template projects are stored in the folder `share/templates`, but you may not have sufficient privileges to create new files in that directory.
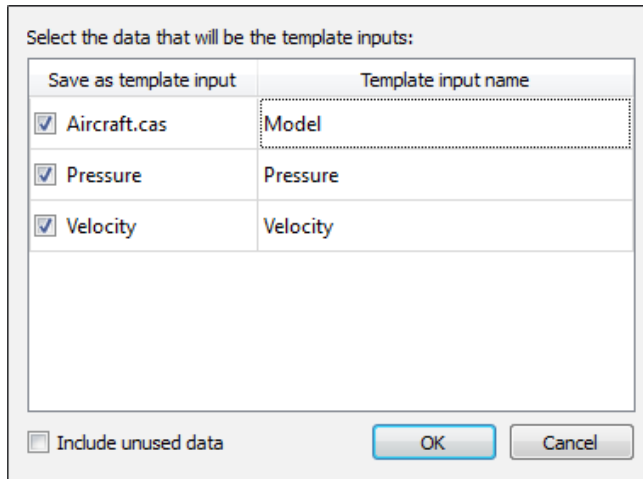
Figure 9.1: The template project save dialog.
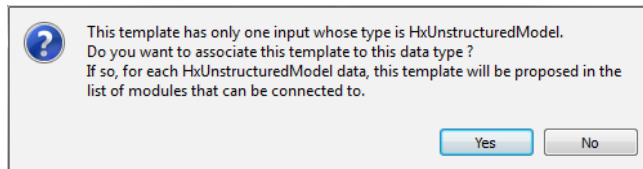
Figure 9.2: Data Type association is possible if template has only one input.

You can save custom templates in any directory. They will be automatically reloaded on each Avizo start-up.

9.1.1.2 How to use a template project

Built-in template projects and known custom user template projects are loaded automatically on Avizo start-up. Loading a template does not mean instantiating the template project. Template projects are only created on user demand, for example, using the *Project > Create Object...* menu. One exception: if user loads a template file via the *Open Data* dialog, the template resource is loaded, and then executed.

If the template is associated with a data type, you can create an instance using the right-click menu for a data object of that type. In this case, the template will be immediately created using the selected data object.

For other templates, you can create an instance from the *Templates* submenu of the *Project > Create Object...* menu. The template may also appear in the macro buttons list. If this case, the following

dialog appears on template execution:
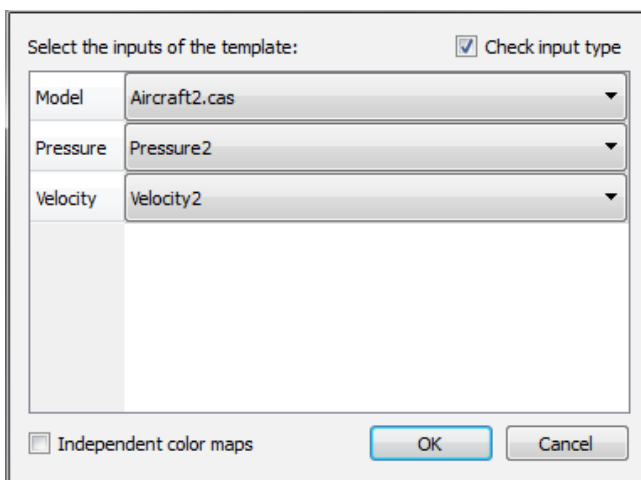
Figure 9.3: The template project run dialog.

Each template input is shown with its template input name and a combo box to select the data set to be used for that input. The candidates listed in each combo box are filtered according to their data type. You can disable this filter and display all data present in the Project View by unselecting the *Check input type* option. If there are no appropriate data objects, the combo box will be empty. You can always select the *"<load file...>"* item to display a file open dialog and choose a data file.

A special treatment for colormaps: by default, colormaps that are already in the Project View are re-used as is. This means, for instance, that objects in the template project may be affected by range changes. You can also choose not to share colormaps with existing objects by selecting the *Independant colormaps* option.

9.2 Recipes

The Avizo XRecipe Extension allows for the creation of user-defined *recipes* for automating a complex scenario, making use of multiple tools, and workspaces. These recipes define high-level workflows, such as extracting user-defined statistics from an image. For more information about the Avizo XRecipe Extension, refer to Chapter 15 - Recipes Workroom.

9.3 Avizo Start-Up

This section describes some options available for Avizo start-up:

- *Command Line Options*
- *Environment Variables*
- *Avizo start-up script*

9.3.1 Command Line Options

This section describes the command line options understood by Avizo. In general, on Unix systems, Avizo is started via the `start` script located in the subdirectory `bin`. Usually, this script will be linked to `/usr/local/bin/Avizo` or something similar. Alternatively, the user may define an alias `Avizo` pointing to `bin/start`.

On Windows systems, Avizo is usually started via the start menu or via a desktop icon. Nevertheless, the Avizo executable may also be invoked directly by calling `bin/arch-Win64VC12-Optimize/Avizo.exe`. In this case, the same command line options as on a Unix system are understood.

The syntax of Avizo is as follows:

```
Avizo [options] [files ...]
```

Data files specified in the command line will be loaded automatically. In addition to data files, script files can also be specified. These scripts will be executed when the program starts.

The following options are supported:

- `-help`
  Prints a short summary of command line options.
- `-version`
  Prints the version string of Avizo.
- `-no_stencils`
  Tells Avizo not to ask for a stencil buffer in its 3D graphics windows. This option can be set to exploit hardware acceleration on some low-end PC graphics boards.
- `-no_overlays`
  Tells Avizo not to use overlay planes in its 3D graphics windows. Use this option if you experience problems when redirecting Avizo on a remote display.
- `-no_gui`
  Starts up Avizo without opening any windows. This option is useful for executing a script in batch mode.
- `-logfile filename`
  Causes any messages printed in the *console window* also to be written into the specified log file. Useful especially in conjunction with the `-no_gui` option.
- `-depth number`
  This option is only supported on Linux systems. It specifies the preferred depth of the depth buffer. The default on Linux systems is 16-bit.

- -style={windows | motif | cde}
  This option sets the display style of Avizo's Qt user interface.
- -debug
  This options applies to the developer version only. It causes local packages to be executed in debug version. By default, optimized code will be used.
- -cmd command [-host hostname] [-port port]
  Send Tcl command to a running Avizo application. Optionally the host name and the port number can be specified. You must type app -listen in the console window of Avizo before commands can be received.
- -clusterdaemon
  Start as VR daemon, on a cluster slave node (Avizo XScreen Extension). This may be replaced by a service. See the online documentation for more information.
- -tclcmd command
  Executes the Tcl command in the starting application.
- -edition { LiteEdition | AvizoEdition | FEIEdition }
  Launches Avizo in a specific edition.

9.3.2 Environment Variables

No special environment settings are required in order to execute Avizo. On Unix systems, some environment variables like the shared library path or the AVIZO_ROOT directory are set automatically by the Avizo start script. Other environment variables may be set by the user in order to control certain features. These variables are listed below. On Unix systems, environment variables can be set using the shell commands setenv (csh or tcsh) or export (sh, bash, or ksh). On Windows, environment variables can be defined in the system properties dialog (Microsoft Windows).

- AVIZO_DATADIR
  A data directory path. This directory will be used as the default directory of the file dialog. Note that for quick access to several directories, one can use operating system, for instance, by adding directories to the list of Favorites places in the file dialog or by using a directory containing shortcuts or links to other directories.
- AVIZO_TEXMEM
  Specifies the amount of texture memory in megabytes. If this variable is not set, some heuristics are applied to determine the amount of texture memory available on a system. However, these heuristics may not always yield a correct value. In such cases, the performance of the *Volume Rendering* module might be improved using this variable.
- AVIZO_MULTISAMPLE
  On high-end graphics systems, a multi-sample visual is used by default. In this way, efficient scene anti-aliasing is achieved. If you want to disable this feature, set the environment variable AVIZO_MULTISAMPLE to 0. Note that on other systems, especially on PCs, anti-aliasing cannot be controlled by the application but has to be activated directly in the graphics driver.

- AVIZO_NO_LICENSE_MESSAGE

  By default, Avizo issues warning messages to the console when your Avizo license is about to expire. This allows you to take timely action so that your use of Avizo is not interrupted unexpectedly when the license expires. To disable these messages, set this variable to 1.

- AVIZO_NO_OVERLAYS

  If this variable is set, Avizo will not use overlay planes in its 3D graphics windows. The same effect can be obtained by means of the -no_overlays command line option. Turn off overlays if you experience problems with redirecting Avizo on a remote display, or if your X server does not support overlay visuals.

- AVIZO_NO_SPLASH_SCREEN

  If this variable is set, Avizo will not display a splash screen while it is initializing.

- AVIZO_LOCAL

  Specifies the location of the local Avizo directory containing user-defined modules. IO routines or modules defined in this directory replace the ones defined in the main Avizo directory. This environment variable overwrites the local Avizo directory set in the development wizard (see Avizo programmer's guide for details).

- AVIZO_SMALLFONT

  Unix systems only. If this variable is set, a small font will be used in all ports being displayed in the *Properties Area* even if the screen resolution is 1280x1024 or bigger. By default, the small font will be used only in case of smaller resolutions.

- AVIZO_XSHM

  Unix systems only. Set this variable to 0 if you want to suppress the use of the X shared memory extension in Avizo's *Segmentation Editor*.

- AVIZO_SPACEMOUSE (deprecated)

  This feature is deprecated.

  Spacemouse support is available by default if Avizo finds a connected device (see http://www.3dconnexion.com). If the driver is installed, a message is printed in the console window. With the spacemouse, you can navigate in the 3D viewer window. Two modes are supported, a rotate mode and a fly mode. You can switch between the two modes by pressing the spacemouse buttons 1 or 2. Further configuration options might be available in the Avizo.init file.

  3Dconnexion Spacemouse limitations:

  - Spacemouse support is not available on Mac OS X.
  - Spacemouse is recognized by Avizo/AvizoClue applications.
  - *Six degrees of freedom* motion is not fully supported yet.
  - Spacemouse can only control the first viewer.
  - It is not possible to translate the camera or move up and down in rotate mode.
  - It is not possible to rotate around the object or move up and down in fly mode.
  - By default, button 1 is used to open the "menu" and it must be reconfigured to "Button 1"

function. Press this button to set rotate mode.

- By default, button 2 is not set to "Button 2" function. Press this button to set fly mode.

- AVIZO_STEREO_ON_DEFAULT
  If this variable is set, the 3D viewer will be opened in OpenGL raw stereo mode by default. In this way, some screen flicker can be avoided which otherwise occurs when switching from mono to stereo mode. Currently, the variable is supported on Unix systems only.

- TMPDIR
  This variables specifies in which directory temporary data should be stored. If not set, such data will be created under /tmp. Among others, this variable is interpreted by Avizo's job queue.

9.3.3 User-defined start-up script

Avizo may be customized in certain ways by providing a user-defined start-up script. The default start-up script, called Avizo.init, is located in the subdirectory share/resources/Avizo of the Avizo installation directory. This script is read each time the program is started. Among other things, the start-up script is responsible for registering file formats, modules, and editors and for loading the default colormaps.

If a file called Avizo.init is found in the current working directory, this file is read instead of the default start-up script. If no such file is found, on Unix systems it is checked if there exists a start-up script called .Avizo in the user's home directory. Below an example of a user-defined start-up script is shown:

```
# Execute the default start-up script
source $AVIZO_ROOT/share/resources/Avizo/Avizo.init 0

# Set up a uniform black background
viewer 0 setBackgroundMode 0
viewer 0 setBackgroundColor black

# Choose non-default font size for the help browser
help setFontSize 12

# Restore camera setting by hitting [F3] key
proc onKeyF3 { } {
    viewer setCameraOrientation 1 0 0 3.14159
    viewer setCameraPosition 0 0 -2.50585
    viewer setCameraFocalDistance 2.50585
}
```

In this example, the system's default start-up script is executed first. This ensures that all Avizo objects are registered properly. Then some special settings are made. Finally, a hot-key procedure is defined

for the function key [F3]. You can define such a procedure for any other function key as well. In addition, procedures like onKeyShiftF3 or onKeyCtrlF3 can be defined. These procedures are executed when a function key is pressed with the [SHIFT] or the [CTRL] modifier key being pressed down.

9.4 TCL Scripting

This section describes how to use *TCL Scripting* in Avizo.

9.4.1 Scripting Introduction

This chapter is intended for advanced Avizo users only. If you do not know what scripting is, it is very likely that you will not need the features described in this chapter.

Beside the interactive control via the graphical user interface, most of the Avizo functionality can also be accessed using specific commands. This allows you to automate certain processes and to create scripts for managing routine tasks or for presenting demos. Avizo's scripting commands are based on Tcl, the *Tool Command Language*. This means you can write command scripts using Tcl with Avizo-specific extensions.

Avizo commands can be typed into the Avizo console window, as described in Section 8.1.13. Commands typed directly into the console window will be executed immediately. Alternatively, commands can be written into a text file, which can then be executed as a whole.

This chapter is organized as follows:

Section 9.4.2 (Introduction to Tcl) gives a short introduction to the Tcl scripting language. This section is not very Avizo specific.

Section 9.4.3 (Avizo Script Interface) explains Avizo-specific commands and concepts related to scripting. This includes a *reference of global commands*.

Section 9.4.5 (Avizo Script Files) explains the different ways of writing and executing script files including references to script objects, resource files, and function-key bound Tcl procedures.

Section 9.4.6 (Configuring Popup Menus) describes how the popup menu of an object can be configured using script commands, and how new entries causing a script to be executed can be created.

Section 9.4.7 (Registering pick callbacks) describes how script callbacks can be attached to objects or viewers and be invoked on user pick events.

Section 9.4.8 (File readers in Tcl) describes how to register a custom file reader implemented in Tcl.

Section 9.4.9 (Creating recipe-compliant script-object) describes how to create a script-object that can be used to create recipe.

Section 9.4.10 (Versioning of Script Objects and backward compatibility) describes backward compatibility considerations and lists all modules with deprecated ports.

The section *Data Type: Script Module* in the User's Reference Guide describes how to use Tcl scripts for defining custom modules that have their own graphical user interface and can be used like the built-in Avizo objects.

9.4.2 Introduction to Tcl

This chapter gives a brief introduction to the Tcl scripting language. If you are familiar with Tcl you can skip this section. However, please notice that instead of the `puts` command, you should use `echo` for output to the Avizo console.

This chapter is not intended to cover all details of the language. For a complete documentation or reference manual of the Tcl language, refer to a text book like *Tcl and the Tk Toolkit* by *John K. Ousterhout*, the creator of Tcl. Like many other books about Tcl, this also covers the Tk GUI toolkit. Note that Tk is not used in Avizo.

Alternatively, you can easily find Tcl documentation and reference manuals on the internet, e.g., at http://www.scriptics.com, or looking up keywords like `Tcl tutorial` or `Tcl documentation` with a search engine.

When you type Tcl commands into the Avizo console, they will be executed as soon as the return key is pressed. Use the completion and history functions provided by the Avizo console, as described in Section 8.1.13 (console window).

9.4.2.1 Tcl Lists, Commands, Comments

First, please note that Tcl is case sensitive: `set` and `Set` are not the same.

A Tcl command is a space-separated list of words. The first word represents the command name, all further words are treated as arguments to that command. As an example, try the Avizo-specific command `echo`, which will print all its arguments to the Avizo console. Try typing

```
echo Hello World
```

This will output the string *Hello World*. Note that Tcl commands can be separated by a semi-colon (;) or a newline character. If you want to execute two successive `echo` commands, you can do it this way:

```
echo Hello World ; echo Hello World2
```

or like this:

```
echo Hello World
echo Hello World2
```

Instead of a command, you can also place a comment in Tcl code. A comment starts with a hash character (#) and is ended by the next line break:

```
# this is a comment
echo Hello World
```

9.4.2.2 Tcl Variables

Variables can be used in Tcl. A variable represents a certain state or value. Using Tcl code, the value of the placeholder can be queried, defined, and modified. To define a variable use the command

```
set name value
```

e.g.,

```
set i 1
set myVar foobar
```

Note that in Tcl, internally all variables are of string type. Since the set command requires exactly one argument as the variable value, you have to quote values that contain spaces:

```
set Output "Hello World"
```

or

```
set Output {Hello World}
```

In order to substitute the value of a variable with name *varname*, a $ sign has to be put in front of that name. The expression $varname will be replaced by the value of the variable. After the above definitions,

```
echo $Output
```

would print

```
Hello World
```

in the console window, and

```
echo "$i.) $Output"
```

would yield the output *1.) Hello World*. Note that variable substitution is performed for strings quoted in ", while it is not done for strings enclosed in braces {}. Even newline characters are allowed in a { } enclosed string. Note however, that it is not possible to type in multi-line commands into the Avizo console.

9.4.2.3 Tcl Command Substitution

To do mathematical computations in Tcl, you can use the command `expr` which will evaluate its arguments and return the value of the expression. Examples are:

```
expr 5 / ( 7 + 3)
expr $i + 1
```

In order to use the result of a command like `expr` for further commands, an important Tcl mechanism has to be used: command substitution, denoted by brackets []. Any list enclosed in brackets [] will be executed as a separate command first, and the [...] construct will be replaced with the *result* of the command. This is similar to the '...' construct in Unix command shells. For example, in order to increase the value of the variable i by one you can use:

```
set i [expr $i + 1]
```

Of course, command expressions can be arbitrarily nested. The order of execution is always from the innermost bracket pair to the outermost one:

```
echo [expr 5 * [expr 7 + [expr 3+3]]]
```

9.4.2.4 Tcl Control Structures

Further important language elements are `if-else` constructs, `for` loops and `while` loops. These constructs typically are multi-line constructs and therefore can not be typed conveniently into the Avizo console. If you want to try the examples shown below, write them into a file like `C:\test.txt` by using a text editor of your choice, and execute the file by typing

```
source C:\test.txt
```

We start with the `if-then` mechanism. It is used to execute some code *conditionally*, only if a certain *expression* evaluates to "true" (meaning a value different from 0):

```
set a 7
set b 8
if {$a < $b} {
    echo "$a is smaller than $b"
} elseif {$a == $b} {
    echo "$a equals $b"
} else {
    echo "$a is greater than $b"
}
```

The `elseif` and `else` parts are optional. Multiple `elseif` parts can be used, but only a single `if` and `else` part.

Another important construct is the conditional loop. Like the `if` command, it is based on checking a conditional expression. In contrast to `if`, the conditional code is executed multiple times, as long as the expression evaluates to true:

```
for {set i 1} {$i < 100} {set i [expr $i*2]} {
    echo $i
}
```

In fact this code is identical to:

```
set i 1
while {$i < 100} {
    echo $i
    set i [expr $i * 2]
}
```

Both loops would produce the output *1, 2, 4, 8, 16, 32, 64*.

If you want to execute a loop for all elements of a list, there is another very convenient command for that:

```
foreach x {1 2 4 8 16 32 64} {
  echo $x
}
```

This will generate the same output as the previous example. Note that the expression enclosed in braces is a space-separated list of words.

9.4.2.5 User-Defined Tcl Procedures

A new function or procedure is defined in Tcl using the `proc` command. Proc takes two arguments: a list of argument names and the Tcl code to be executed. Once a procedure is defined, it can be used just like any other Tcl command:

```
proc computeAverageA {a b} {
    return [expr ($a+$b)/2.0]
}
proc computeAverageB {a b c} {
    return [expr ($a+$b+$c)/3.0]
}
echo "average of 2 and 3: [computeAverageA 2 3]"
echo "average of 2,3,4: [computeAverageB 2 3 4]"
```

As you can see in the example, the argument list defines the names for local variables that can be used in the body of the procedure (e.g. $a). The return command is used to define the result of the procedure. This result is the value that is used in the command bracket substitution [].

If you want to define a procedure with a flexible number of arguments, you must use the special argument name args. If the argument list contains just this word, the newly defined command will accept an arbitrary number of arguments, and these arguments are passed as a *list* called args:

```
proc computeAverage args {
    set result 0
    foreach x $args {
        set result [expr $result + $x]
    }
    return [expr $result / [llength $args]]
}
```

In this example, the llength command returns the number of elements contained in the args list.

Note that the variable result defined in the procedure has *local* scope, meaning that it will not be known outside the body of the procedure. Also, the value of globally defined variables is not known within a procedure unless that global variable is declared using the keyword global:

```
set x 3
proc printX {} {
    global x
    echo "the value of x is $x"
}
```

There is much more to be said about procedures, e.g., concerning argument passing, evaluation of commands in the context outside of the procedure, and so on. Please refer to a Tcl reference book for these advanced topics.

9.4.2.6 List and String Manipulation

Finally, at the end of this brief Tcl introduction, we come back to the concept of lists. Basically everything in Tcl is constructed using lists, so it is very important to know the most important list manipulation commands as well as to understand some subtle details.

Here is an example of how to take an input list of numbers and construct an output list in which each element is twice as big as the corresponding element in the input list:

```
set input [list 1 2 3 4 5]
set output [list]
```

```
foreach element $input {
  lappend output [expr $element * 2]
}
```

You can think of lists as simple strings in which the list elements are separated by spaces. This means that you can achieve the same result as in the previous example without using the list commands:

```
set input "1 2 3 4 5"
set output ""
foreach element $input {
  append output "[expr $element * 2] "
}
```

The *append* command is similar to *lappend*, but it just adds a string at the end of an existing string. List manipulation becomes much more involved when you start nesting lists. Nested lists are represented using nested pairs of braces, e.g.

```
set input {1 2 {3 4 5 {6 7} 8 } 9}
foreach x $input {
  echo $x
}
```

The result of this command will be

```
1
2
3 4 5 {6 7} 8
9
```

Please note that Tcl will automatically *quote* strings that are not single words when constructing a list. Here is an example:

```
set i [list 1 2 3]
lappend i "4 5 6"
echo $i
```

will yield the output

```
1 2 3 {4 5 6}
```

You can use the *lindex* command to access a single element of a list. *lindex* takes two arguments: the list and the index number of the desired element, starting with 0:

```
set i [list a b c d e]
echo [lindex $i 2]
```

will yield the result c.

9.4.3   Avizo Script Interface

Although the Tcl language is not intrinsically object oriented, the Avizo script interface is. There is one command for each object in the Avizo Project View. In addition there are several *global commands* associated with global objects in Avizo such as the *viewer* or the Avizo*main window*.

A command associated with an object in the Project View (e.g., an "Ortho Slice" module or an "Iso-surface" module) only exists while the object exists. These commands are identical to the name of the object as displayed in the Project View. Typically the script interface of a specific object contains many different functions. The general syntax for an Avizo object-related command is

```
<object-name> <command-word> <optional-arguments> ...
```

For example, if an object called "Global Axes" exists (choose View/Axis from the Avizo menu) then you can use commands like

```
"GlobalAxes" deselect
"GlobalAxes" select
"GlobalAxes" setIconPosition 100 100
```

Note: Modules are renamed in camel case. These three words are possible in commands: "Global Axes", "GlobalAxes" or GlobalAxes.

Remember to use the completion and history functions provided by the Avizo console, as described in Section 8.1.13 (console window) to save typing.

If you have already used Avizo, you have noticed that the parameters and the behavior of an Avizo module are controlled via its *ports*. The ports provide a user interface to change their values when the module is selected. All ports can also be controlled via the command interface. The general syntax for that is

```
<object-name> <port-name> <port-command> <optional-arguments> ...
```

For example, for the "Global Axes" you can type

```
"GlobalAxes" options setValue 1 1
"GlobalAxes" thickness setValue 1.5
"GlobalAxes" fire
```

When you type in these commands, you will notice that the values in the user interface change immediately. However, the module's compute method is not called until explicitly firing the module using the `fire` command. This allows you to first set values for multiple ports without a recomputation after each command. However, note that some modules automatically reset some of their ports, for example, when a new input object is connected. In such cases, you may need to call `fire` after setting the value of every single port.

Usually the name of a port is identical to the text label displayed in the graphical user interface, except that white spaces are removed and command names start with a lower case letter. To find out the names of all ports of a specific module, use the command

```
<object-name> allPorts
```

Almost all ports provide a `setValue` and a `getValue` command. The number of parameters and the syntax, of course, depend on the ports.

Commands of the type `<object-name> <port-name> setValue ...` make up more than 90% of a typical Avizo script. However, besides the port commands, many Avizo objects provide additional specific commands. The command interface for a particular module is described in the User's Reference Guide. You can quickly find the corresponding page by clicking the ? button in the Properties Area when the module has been selected.

As a quick help, entering an object's name without further options will display all commands available for that object. Note that this will also show undocumented, unreleased, and experimental commands. In order to get more information about a particular module or port command, you can type it into the console window without any arguments and then press the F1 key. This opens the help browser with a command description.

Avizo objects are part of a class hierarchy. Similar to the C++ programming interface, script commands are also inherited by derived classes from its base classes. This means that a particular object like the axis object will, beside its own specific commands, also provide all the commands available in its base classes. Links to the base class commands are given in a module's documentation.

9.4.3.1 Predefined Variables

Some variables in Avizo Tcl exist that are predefined and have a special meaning. These are

- `AVIZO_ROOT`: Avizo installation directory.
- `AVIZO_LOCAL`: Personal Avizo development directory (Avizo XPand Extension only).
- `SCRIPTFILE`: Tcl script file currently executed.
- `SCRIPTDIR`: Directory in which currently executed script resides.
- `hideNewModules`: If set to 1, icons of newly created modules will initially be hidden. Be careful to set this variable only when strictly necessary and restore it immediately after, in order to avoid to keep hiding accidentally created modules, for instance, in case of script interruption.

9.4.3.2 Object commands

The basic command interface of Avizo modules and data objects is described in the data type chapter of the reference part of the user's guide in the *Object* section. The basic syntax of object commands is

```
<object> <command> <arguments> ...
```

where `<object>` refers to the name of the object and `<command>` denotes the command to be executed. Each module or data object may define its own set of commands in addition to the commands defined by its base classes. The commands described in the *Object* section are provided by all modules and data objects.

Global commands are described in the following section.

9.4.4 Global Commands

This section lists Avizo-specific global Tcl commands. Some of these commands are associated with certain global objects in Avizo, such as the console window, the main window, or the viewer window. Other commands such as `load` or `echo` are not. These commands are described in one common subsection. In summary, the following command sections are provided:

- *viewer command options* (`viewer`)
- *main window command options* (`theMain`)
- *console command options* (`theMsg`)
- *common commands for top-level windows*
- *progress bar command options* (`workArea`)
- *application command options* (`app`)
- *other global commands*

9.4.4.1 Viewer command options

Commands to a viewer can be entered in the console window. The syntax is

```
viewer [<number>] command,
```

where `<number>` specifies the viewer being addressed. The value 0 refers to the main viewer and may be omitted for convenience.

Commands

```
viewer [<number>] snapshot [-offscreen [<width> <height>]]
[-stereo] [-alpha] [-tiled <nx> <ny>] <filename> [filename2]
```
This command takes a snapshot of the current scene and saves it under the specific filename.

The image format will be automatically determined by the extension of the file name. The list of available formats includes: TIFF (`.tif,.tiff`), SGI-RGB (`.rgb,.sgi,.bw`), JPEG (`.jpg,.jpeg`), PNM (`.pgm,.ppm`), BMP (`.bmp`), PNG (`.png`), and Encapsulated PostScript (`.eps`). If the viewer number is not given, the snapshot is taken from all viewers, if you have selected the 2 or 4 viewer layout from the *View menu*.

If the `-offscreen` option is specified, offscreen rendering with a maximum size of 2048x2048 is used. In this case, the viewer number is required even if viewer 0 is addressed. If the width and height is not specified explicitly, the size of the image is the current size of the viewer.

Caution: If you have more than one transparent object visible in the viewer and you want to use offscreen rendering, set the transparency mode to *Blend Delayed* and check to see if all objects are rendered properly prior to taking a snapshot.

If `-stereo` option is used, the stereo mode image is created. In this case, *filename2* file can be used to specify where the second image of the stereo image is stored.

If `-alpha` option is used, the snapshot image is created with transparent background.

If `-tiled nx ny` option is used, tiled rendering is used with *nx* number of tiles in horizontal direction, and *ny* number of tiles in vertical direction.

```
viewer [<number>] setPosition <x> <y>
```
(in top-level mode only) Sets the position of the viewer window relative to the upper left corner of the screen. If more than one viewer is shown in the same window, the position of the toplevel window is set.

```
viewer [<number>] getPosition
```
Returns the position of the viewer window. If more than one viewer is shown in the same window, the position of the toplevel window is returned.

```
viewer [<number>] setSize <width> <height>
```
(in top-level mode only) Sets the size of the viewer window. Width and height specify the size of the actual graphics area. The window size might be a little bit larger because of the viewer decoration and the window frame.

Caution: A warning message is printed in the console when a viewer is not resized with the requested size. When setting a new size to a viewer, it can happen that the new viewer size is not the requested one. This case can happen when:

- the viewer is in top level and the given size is too small (ex: (10, 10))
- the viewer is not in top level and the main window can't be resized to a smaller size (Example: a widget is blocking the main window resize like the unified title and tool bar on Mac or a dock widget with a minimal width).

```
viewer [<number>] getSize
```
Returns the size of the viewer window without decoration and window frame.

```
viewer [<number>] setCamera <camera-string>
```
Restores all camera settings. The camera string should be the output of a `getCamera` command.

```
viewer [<number>] getCamera
```
This command returns the current camera settings, i.e., position, orientation, focal distance, type, and height angle (for perspective cameras) or height (for orthographic cameras). The values are returned as Avizo commands, which can be executed in order to restore the camera settings. The complete command string may also be passed to `setCamera` at once.

```
viewer [<number>] setCameraPosition <x> <y> <z>
```
Defines the position of the camera in world coordinates.

```
viewer [<number>] getCameraPosition <x> <y> <z>
```
Returns the position of the camera in world coordinates.

```
viewer [<number>] setCameraNearDistance <value>
```
Defines the distance from the camera viewpoint to the near clipping plane.

```
viewer [<number>] setCameraFarDistance <value>
```
Defines the distance from the camera viewpoint to the far clipping plane.

```
viewer [<number>] setCameraOrientation <x> <y> <z> <a>
```
Defines the orientation of the camera. By default, the camera looks in negative z-direction with the y-axis pointing upwards. Any other orientation may be specified as a rotation relative to the default direction. The rotation is specified by a rotation axis *x y z* followed by a rotation angle *a* (in radians).

```
viewer [<number>] getCameraOrientation
```
Returns the current orientation of the camera in the same format used by `setCameraOrientation`.

```
viewer [<number>] setCameraFocalDistance <value>
```
Defines the camera's focal distance. The focal distance is used to compute the center around which the scene is rotated in interactive viewing mode.

```
viewer [<number>] getCameraFocalDistance
```
Returns the current focal distance of the camera.

```
viewer [<number>] setCameraHeightAngle <degrees>
```
Sets the height angle of a perspective camera in degrees. Making the angle smaller makes the field of view smaller, effectively "zooming in", as with a telephoto lens. Unless you specifically want to change the camera field of view, it is normally better to move the camera closer to an object (sometimes called "dolly in") to make the object appear larger. The command has no effect if the current camera is an orthographic one.

```
viewer [<number>] getCameraHeightAngle
```
Returns the height angle of a perspective camera.

```
viewer [<number>] setCameraHeight <height>
```
Sets the height of the view volume of an orthographic camera. The command has no effect if the camera is an perspective one.

```
viewer [<number>] getCameraHeight
```
Returns the height of an orthographic camera.

```
viewer [<number>] setCameraType <perspective|orthographic>
```
Sets the camera type.

```
viewer [<number>] getCameraType
```
Returns the camera type.

```
viewer [<number>] setTransparencyType <type>
```
This command defines the strategy used for rendering transparent objects. The argument *type* may be a number between 0 and 8, corresponding to the entries *Screen Door*, *Add*, *Add Delay*, *Add Sorted*, *Blend*, *Blend Delay*, *Blend Sorted*, *Sorted Layers* and *Sorted Layers Delayed* as described for the *View menu*.

Most accurate results are obtained using mode 8. Default is mode 6. In the default mode, some objects may not be recognized correctly as being transparent. In this case, you may switch them off and on again in order to force them to be rendered last. Also, problems may occur if lines are to be rendered on a transparent background. In this case, you may use transparency mode 4 and ensure the correct rendering order manually.

```
viewer [<number>] getTransparencyType
```
This command returns the current transparency type as a number, the meaning of this number is the same as in `setTransparencyType`.

```
viewer [<number>] setSortedLayersNumPasses <value>
```
Sets the number of rendering passes used when transparency type is *Sorted Layers* or *Sorted Layers Delayed*. Use more passes for more correct transparency. Usually four passes (which is the default value) gives good results.

```
viewer [<number>] getSortedLayersNumPasses
```
Returns the number of rendering passes used when transparency type is *Sorted Layers* or *Sorted Layers Delayed*.

```
viewer [<number>] setBackgroundColor <r> <g> <b>
```
This command sets the color of the background to a specific value. The color may be specified either as a triple of integer RGB values in the range 0...255, as a triple of rational RGB values in the range 0.0...1.0, or simply as plain text, e.g., *white*, where the list of allowed color names is defined in `/usr/lib/X11/rgb.txt`.

```
viewer [<number>] getBackgroundColor
```
Returns the primary background color as an RGB triple with values between 0 and 1.

`viewer [<number>] setBackgroundColor2 <r> <g> <b>`
Sets the secondary background color which is used by non-uniform background modes.

`viewer [<number>] getBackgroundColor2`
Returns the secondary background color as an RGB triple with values between 0 and 1.

`viewer setBackgroundMode <mode>`
Allows you to specify different background patterns. If mode is set to 0, a uniform background will be displayed. Mode 1 denotes a gradient background. Mode 2 which corresponds to checkerboard pattern has been removed. Finally, mode 3 draws an image previously defined with `setBackgroundImage` on the background.

`viewer getBackgroundMode`
Returns the current background mode.

`viewer setBackgroundImage <imagefile> [<imagefile2>] [-stereo]`
This command allows you to place an arbitrary raster image into the center of the viewer's background. The image must not be larger than the viewer window itself. Otherwise, it will be clipped. The format of the image file will be detected automatically by looking at the file name extension. All formats mentioned for the `snapshot` command are supported except for Encapsulated PostScript. If a second image file is specified, this file will be used as the right eye image in case of active stereo rendering. If the options `-stereo` is specified and only one image file is given, it it assumed that this file contains a left eye view and a right eye view composited side by side. These views then will be separated automatically.

`viewer getBackgroundImage`
This command returns the file name of the last background image file defined with `setBackgroundImage`. If a pair of stereo images was specified, two file names are returned. If the option `-stereo` was used in `setBackgroundImage`, this option will be returned too.

`viewer [<number>] setAutoRedraw <state>`
If *state* is 0, the auto redraw mode is switched off. In this case, the image displayed in the viewer window will not be updated, unless a redraw command is sent. If *state* is 1, the auto redraw mode is switched on again. In a script, it might be useful to disable the auto redraw mode temporarily.

`viewer [<number>] isAutoRedraw`
Returns true if auto redraw mode is on.

`viewer [<number>] redraw`
This command forces the current scene to be redrawn. An explicit *redraw* is only necessary if the auto redraw mode has been disabled.

`viewer [<number>] rotate <degrees> [x|y|z|m|u|v]`
Rotates the camera around an axis. The axis to be taken is specified by the second argument. The following choices are available:

- x: the x-axis (1,0,0)
- y: the y-axis (0,1,0)
- z: the z-axis (0,0,1)
- m: the most vertical axis of x, y, or z
- u: the viewer's up direction
- v: the view direction

The last option does the same as the rotate button of the user interface. In most cases the *m* option is most adequate. For backward-compatibility the default is *u*.

```
viewer [<number>] setDecoration <state>
```
Deprecated.

```
viewer [<number>] saveScene [-b] [-r] [-z] <filename>
```
Saves all of the geometry displayed in a viewer in Open Inventor 3D graphics format. Warning: Since many Avizo modules use custom Open Inventor nodes, the scene usually can not be displayed correctly in external programs like *ivview*. The following options are available:

- -b : The Open Inventor file is saved in binary format.
- -r : The geometry displayed in a viewer but also additional properties are saved in the Open Inventor file.
- -z : The Open Inventor file is be saved in compressed format (zip compression is used).

```
viewer [<number>] viewAll
```
Resets the camera so that the whole scene becomes visible. This method is called automatically for the first object being displayed in a viewer.

```
viewer [<number>] show
```
This command opens the specified viewer and ensures that the viewer window is displayed on top of all other windows on the screen.

```
viewer [<number>] hide
```
This command closes the specified viewer.

```
viewer [<number>] isVisible
```
This command indicates if the specified viewer is visible.

```
viewer [<number>] fogRange <min> <max>
```
Sets a range of attenuation for the fog affect that can be introduced into a viewer scene by the *View menu.* The default range is $[0, 1]$. Values within this range correspond to distances of scene points from the camera, such that points nearest to the camera have value zero and those farthest away have value one. Restricting the range of attenuation means that attenuation will start at points where the specified minimum is attained and reach its maximum at points where the specified maximum

is attained. Maximum attenuation by fog is equivalent to invisibility, thus all points beyond that maximum will appear as background.

`viewer [<number>] setVideoFormat pal|ntsc`
Sets the size of the viewer window according to PAL 601 or NTSC 601 resolution, i.e., 720x576 pixels or 720x486 pixels. The current setting of the decoration is taken into account.

`viewer [<number>] setVideoFrame <state>`
If *state* is 1, a frame is displayed in the overlay plane of the viewer. This frame depicts the area where images recorded to video are safely shown on video players. Sets the viewing *state*: 0 switches the frame off. Note: Objects displayed in the overlay planes are not saved to file with the `snapshot` command (see above).

`viewer [<number>] getViewerSpinAnimation`
Return 1 if viewer spin animation is turned on, otherwise return 0.

`viewer [<number>] setViewerSpinAnimation <state>`
If *state* is 1, a viewer spin animation is turned on. Otherwise, passing 0 as *state* will turn off viewer spin animation. Note: State of the viewer spin animation is saved as preference, so it will remain the same upon restartin Avizo.

`viewer [<number>] setViewing <state>`
Sets the viewing *state*: 0 switches the viewer to interaction mode, 1 switches it to viewing mode.

`viewer [<number>] getViewing`
Indicates the viewing state: 0 for interaction mode and 1 for viewing mode.

`viewer [<number>] linkViewer [<ID>...]`
This command is used for linking viewers. This command works exactly the same as the appropriate GUI action.

`viewer [<number>] unlinkViewer [<ID>...]  [all]`
This command is used to unlink linked viewers.

9.4.4.2   Main window command options

The command `theMain` allows you to access and control the Avizo main window. Besides the specific command options listed below, all sub-commands listed in Section 9.4.4.4 (Common commands for top-level windows) can also be used.

Commands

`theMain snapshot` *filename*
Creates and saves a snapshot image of the main window. The format of the image file is determined

from the file name extension. Any standard image file format supported by Avizo can be used, e.g., `.jpg`, `.tif`, `.png`, or `.bmp`.

`theMain setViewerTogglesOnIcons {0|1}`
Enables or disables the display of the orange viewer toggles on object icons in the Avizo Project View.

`theMain ignoreShow [0|1]`
Enables or disables the special purpose *no show flag*. If this flag is set, subsequent `mainWindow show` commands are ignored. This can be useful to run standard Avizo scripts in a Avizo XScreen Extension environment. Calling the command without an argument just returns the current value of the flag.

`theMain showConsole [0|1]`
Enables or disables display of console window in Avizo.

9.4.4.3 Console command options

The command `theMsg` allows you to access and control the Avizo console window. Besides the specific command options listed below, all sub-commands listed in Section 9.4.4.4 (Common commands for top-level windows) can also be used.

Commands

`theMsg error <message> [<btn0-text>] [<btn1-text>]`
`[<btn2-text>]`
Pops up an error dialog with the specified message. The dialog can be configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg warning <message> [<btn0-text>] [<btn1-text>]`
`[<btn2-text>]`
Pops up a warning dialog with the specified message. The dialog can be configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg question <message> [<btn0-text>] [<btn1-text>]`
`[<btn2-text>]`
Pops up a question dialog with the specified message. The dialog can be configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg overwrite <filename>`
Pops up a dialog asking the user if it is ok to overwrite the specified file. If the user clicks *Ok*, 1 is returned, otherwise 0.

9.4.4.4   Common commands for top-level windows

These commands are available for all Avizo objects which open a separate top-level window. In particular, these are the Avizo main window (`theMain`), the console window (`theMsg`), and the viewer window (`viewer 0`). For example, you can set or get the position of these windows using the corresponding global command followed by `setPosition` or `getPosition`.

Commands

`getFrameGeometry`
Returns the position and size of the window including the window frame. In total, four numbers are returned. The first two numbers indicate the position of the upper left corner of the window frame relative to the upper left corner of the desktop. The last two numbers indicate the window size in pixels.

`getGeometry`
Returns the position and size of the window without the window frame. In total, four numbers are returned. The first two numbers indicate the position of the upper left corner of the window relative to the upper left corner of the desktop. The last two numbers indicate the window size in pixels.

`getPosition`
Returns the position of the upper left corner of the window including the window frame. This is the same as the first two numbers returned by `getFrameGeometry`.

`getRelativeGeometry`
Returns the position and size of the window including the window frame in relative coordinates. The size of the desktop is (1,1). The position and size of the window is specified by fractional numbers between 0 and 1.

`getSize`
Returns the size of the window without the window frame. This is the same as the last two numbers returned by `getGeoemtry`.

`hide`
Hides the window.

`setCaption <text>`
Sets the window title displayed in the window frame.

`setFrameGeometry <x y width height>`
Sets the position and size of the window including the window frame. Four numbers need to be specified, the x- and y-positions, the window width and the window height.

`setGeometry <x y width height>`
Sets the position and size of the window without the window frame. Four numbers need to be specified, the x- and y-positions, the window width and the window height.

`setPosition <x y>`
Sets the position of the upper left corner of the window frame.

`setRelativeGeometry <x y width height>`
Sets the position and size of the window including the window frame in relative coordinates. The size of the desktop is (1,1). The position and size of the window is specified by fractional numbers between 0 and 1.

`setSize <width height>`
Sets the size of the window without the window frame.

`show`
Makes the window visible in normal state. Also raises the window.

`showMinimized`
Makes the window visible in iconified state.

`showMaximized`
Makes the window visible in maximized state.

9.4.4.5 Progress bar command options

The command `workArea` allows you to access the progress bar located in the lower part of the Avizo main window. You can print messages or check if the stop button was pressed.

Commands

`workArea setProgressInfo <text>`
Sets an info text to be displayed in the progress bar. The text can be used to describe the status during some computation.

`workArea setProgressValue <value>`
Sets the value of the progress bar. The argument must be a floating point number between 0 and 1. For example, a value of 0.8 indicates that 80% of the current task has been done.

`workArea startWorking [<message>]`
Activates the stop button. After calling this command, the Avizo stop button becomes active. In your script, you can check if the stop button was hit by calling `workArea wasInterrupted`. When the stop button is active, you can't interact with any other widget unless you call `workArea stopWorking` in your script. Therefore, you must not enter this command directly in the console window, but you should only use it in a script file or in a Tcl procedure.

`workArea stopWorking`
Deactivates the stop button. Call this command when the compute task started with `workArea`

`startWorking` is done or if the user pressed the stop button. This command also restores the progress info text which was shown before calling `startWorking`.

`workArea wasInterrupted`
Checks if the user pressed the stop button. You should only use this command between `workArea startWorking` and `workArea stopWorking`. If there are multiple nested compute tasks and the user presses the stop button, all subsequent calls to `wasInterrupted` return true until the first level is reached.

9.4.4.6 Application command options

The `app` command provides several options not related to a particular object or component in Avizo, but related to Avizo itself.

Commands

`app version`
Returns the current Avizo version.

`app uname`
Returns simplified name of operating system.

`app arch`
Returns Avizo architecture string, e.g., arch-Win32VC8-Optimize, arch-LinuxAMD64-Optimize.

`app hostid`
Returns host id needed to create an Avizo license key.

`app listen [port]`
Opens a socket to which Tcl commands can be sent. The TCP/IP port can be specified optionally. WARNING: This can create security holes. Do not use this unless behind a firewall and if you know what you are doing. In the *Network Preferences*, the port can be set and the listening port can be opened or closed.

`app close`
Closes the Avizo Tcl port.

`app port`
Returns port number of Avizo Tcl port. Returns -1 if socket has not been opened.

`app send <command> [<host> [<port>]]`
Sends a Tcl command to a listening Avizo. If no host or port are specified, the Avizo instance will send the command to host and port specified in the preferences. See section *Network Preferences*.

`app opengl`
Retrieves information about the used OpenGL driver including version number and supported extensions. This is useful information to send to the hotline if reporting rendering problems.

```
app cluster
```
Returns the current node status which can be "master" or "slave" if some cluster mode is active or simply "single" if is not the case.

```
app memTotal [-k | -m | -g]
```
Returns the physical memory of the system in bytes. Optional switches -k, -m, -g convert the output to kilo-, mega-, or gigabytes, respectively.

```
app memAvail [-k | -m | -g]
```
Returns the available memory of the system in bytes. Optional switches -k, -m, -g convert the output to kilo-, mega-, or gigabytes, respectively. Note that depending on the operating system, the output can deviate from that reported by other tools.

```
app config <key> [<val>]
```
Provides access to custom configuration settings that are permanently stored. `app config <key>` returns the current configuration value for `<key>`. `app config <key> <val>` sets a new value.

9.4.4.7   Other global commands

Commands

addTimeout *msec procedure* [*arg*]
Schedules a Tcl procedure for being called after *msec* milliseconds. If *arg* is specified, it will be passed to the procedure. The specified procedure will be called only once. If necessary, you can schedule it again in the time-out procedure. Example: `addTimeout 10000 echo {10 seconds are over.}`

all [-selected | -visible | -hidden] [*type*]
Returns a list of all Avizo objects currently in the Project View. If *type* is specified, only objects with that C++ class type (or derived objects) are returned. Search can be limited to selected, visible, or hidden objects, respectively. Example: `all -hidden HxColormap`.

aminfo [-a *outfile*|-b *outfile*] *Avizo-File*
If used with only a file name as argument, this command will open the file which has to be in Avizo data format and print header information. If used with the -a or -b option, the outputfile specified as argument *outfile* will be written in ASCII (-a) or binary (-b) format, respectively. Thus, *aminfo* can be used to convert binary Avizo data into ASCII and vice versa.

clear
Clears console window.

create *class name* [*instance name*]
Creates an instance of an Avizo object like a module or data object. Returns the instance name.

Note that data objects are normally not created this way but by loading them from a file. Example: `create HxOrthoSlice MySlice`.

`dso` *options*
Controls loading of dynamic libraries ("dynamic shared objects"). The following options are provided:

- `addPath path ...`: Adds a path to the list of directories to be searched when loading a dynamic library.
- `verbose {0|1}`: Switches on and off debug information related to dynamic libraries.
- `open <package>`: Tries to load the specified dynamic library. It is enough to specify the package name, e.g., `hxfield`. This name will be automatically converted into the platform dependent name, e.g., `libhxfield.so` on Linux or `hxfield.dll` on Windows.
- `unloadPackage <package>`: Unloads (if possible) the specified dynamic library.
- `execute <package> <function>`: Executes the function defined in the specified dynamic library.

`echo` *args*
Prints its arguments to the Avizo console. Use this rather than the native Tcl command `puts` which prints to stdout.

`help` *arguments*
Without arguments this opens the Avizo help browser.

`httpd [port]`
Start a built-in httpd server. The http server will deliver any document requested. If a requested document ends with `.hx`, Avizo will instead of delivering it execute the file as a Tcl script. This can be used to control Avizo from a web browser. WARNING: This command can create security holes. Do not use this unless behind a firewall and if you know what you are doing.

`limit {datasize | stacksize | coredumpsize}` *size*
Change process limits. Available on Unix platforms only. Use "unlimited" as size for no limit. The size has to be specified in bytes. Alternatively, you can use for example 1000k for 1000 kilobytes or 1m for one megabyte.

`load [fileformat]` *options files*
Load data from one or more files. Optionally a file format can be specified to override Avizo's automatic file format recognition. The file format is specified by the same label which is displayed in the file format combo box in the Avizo file dialog. The list of all file formats supported by Avizo can be obtained using the global command *fileFormats*. Remote files can be read by using `FTP` or `HTTP` protocol.

Additional options are:

- `-browse`: *Open Data* window is shown.

- `-avizoscript`: open Avizo script files.
- `-avizo`: Avizo's native general-purpose file format. It is used to load many different data objects like fields defined on regular or tetrahedral grids, segmentation results, colormaps, or vertex sets such as landmarks.
- `-dataOnly`: prevents importer from creating display modules, useful in hx files.

Options for raw data:

- `load -raw FileName Endianess IndexOrder`

- `DataType nDataVar dimX dimY dimZ`

- `xMin xMax yMin yMax zMin zMax`

- Options for DICOM data:
  `-nodialog`: prevents the Dicom dialog box from being shown. This option should be specified after the filename and does not work unless DICOM format is specified. See the example below:

  `load -dicom` *options* *files* `-nodialog`

`mem`
Prints out some memory statistics.

`quit`
Immediately quits Avizo.

`remove` {*objectname* | *-all* | *-selected*}
Removes objects from Project View.

- `objectname`: the specified Avizo object.
- `-all`: all objects.
- `-selected`: selected objects.

`removeTimeout` *procedure* [*arg*]
Unschedules a Tcl procedure previously scheduled with addTimeout.

`rename` *objectname* *newname*
Changes instance name of an object. Identical to *objectname* `setLabel` *newname*, except that it returns 1 if successful, and nothing if unsuccessful.

`sleep` *sec*

Waits for *sec* seconds. Avizo will not process events in that time.

`source` *filename*

Loads and executes Tcl commands from the specified file. If the script file contains the extension `.hx`, the `load` command may be used as well.

`system` *command*

Executes an external program. Do not use this unless you know what you are doing.

`saveProject`

Saves current project. If the project is not previously saved, then the project will be saved in the Avizo root dir as `Untitled.hx`.

`saveProjectAs [-forceAutoSave | -packAndGo]` *arg*

A copy of the current project will be saved as `arg` in Avizo root dir.(e.g., `saveProjectAs myProject`). When using a path, a full path needs to be specified and a `.hx` extension needs to be added on the project name (e.g., `saveProjectAs c:/work/myProject.hx`). Optionally, a forceAutoSave parameter can be specified to force auto saving of modified data without displaying a warning dialog. If parameter *packAndGo* is specified, in the same folder where the project file is saved, a new folder will be created and it will contain all data necessary for loading the saved project. Note: If a file exists it will not be overwritten.

`theObjectPool setSelectionOrder` {*first object*} {*second object*}...

This command reorders the selection so that it matches the given object order. Selected objects not contained inside this list will be moved at the end of the selection (their relative order will not be changed though).

`thePreferences [`*save* `|` *load*`]` *filename*

This command saves or loads preferences to/from the file specified as *filename*.

`theProperties [`*show* `|` *hide*`]`

This command shows or hides the *Properties panel in Avizo.*

`theProjectView [`*show* `|` *hide*`]`

This command shows or hides the Project View panel in Avizo.

`fileFormats`

Shows all file formats which can be used in Avizo.

9.4.5   Avizo Script Files

It is worth noticing that an Avizo project is simply a Tcl script that will regenerate the current Avizo state. Therefore, it is often efficient to interactively create an Avizo project, save it with "Save Project", and use this as a starting point for scripting.

The simplest way to execute Tcl commands in Avizo is to type them into the Avizo console window. This, however, is not practical for multi-line constructs, like loops or procedures. In this case, it is recommended to write the Tcl code into a file and execute the file with the command `source` *filename*. You can also use the `source` command inside a file in order to include the contents of a file into another file.

Alternatively one can also use the command `load` *filename* or the *Open Project...* menu entry from the *File* menu and the file browser. Then, however, in order to let Avizo recognize the file format, either the file name must end with `.hx`, or the file contents must start with the header line

```
# Avizo Script
```

There are some Tcl files that are loaded automatically when Avizo starts. At startup, the program looks for a file called `.Avizo` in the current directory or in the home directory (see Section 9.3.3 (Start-up script) for details). If no such user-defined start-up script is found, the default initialization script `Avizo.init` is loaded from the directory `$AVIZO_LOCAL/share/resources/Avizo` or `$AVIZO_ROOT/share/resources/Avizo`. This script then reads in all files ending with `.rc` from the `share/resources` subdirectory. The `.rc` files are needed to register modules and data types. Therefore, one can customize the startup behavior of Avizo by simply adding a new `.rc` file to that directory or by modifying the `Avizo.init` file.

Note:
These script files must be encoded in utf-8 in order to work with non ascii characters.

Another way of executing Tcl code is to define procedures that are associated with function keys. If predefined procedures with the names `onKeyF2`, `onKeyF3`, `...`, `onKeyShiftF2`, `...`, `onKeyCtrlF2`, `...`, `onKeyCtrlShiftF2`, `...` exist, these procedures will be automatically called when the respective key is pressed in the Avizo main window, console window, or viewer window. To define these procedures, write them into a file and `source` it or write them into `Avizo.init` or in one of the `.rc` files. An example is

```
proc onKeyF2 { } {
    echo "Key F2 was hit"
    viewer 0 viewAll
}
```

Note:
Some of these functions can be reserved for Avizo specific actions. For example, [F1] is always reserved for help and [F2] is reserved for object renaming when pressed in the Project View or the Tree View.

Finally, Tcl scripts can also be represented in the GUI and be combined with a user interface. In Avizo this is called a *Script Module*.

9.4.6 Configuring Popup Menus

In Avizo, all of the modules that can be attached to a data object are listed in the object's popup menu, which is activated by right-clicking on the object's icon. For some applications, it makes sense to customize new modules using Tcl commands after they have been created. Sometimes it also makes sense to add new entries to an object's popup menu, causing a particular script to be executed. This sections describes how to achieve these goals by modifying Avizo resource files or creating new ones.

Avizo resource files are located in the directory `$AVIZO_ROOT/share/resources`, where `$AVIZO_ROOT` denotes the directory where Avizo has been installed. Resource files are just ordinary script files, although they are identified by the suffix `.rc`. When Avizo is started, all resource files in the resources directory are read. In a resource file, modules, editors, and IO routines are registered using special Tcl commands. Registering a module means to specify its name as it should appear in the popup menu, the type of objects to which it can be attached, the name of the shared library or DLL in which the module is define, and so on. For example, the *Multi-Thresholding* module is registered by the following command in the file `hxlattice.rc`:

```
module -name "Multi-Thresholding" \
    -check { ![$PRIMARY hasInterface HxLabelLattice3] &&
            ([$PRIMARY primType]<3 ||
             [$PRIMARY primType]==7 ||
             [$PRIMARY primType]==8) } \
    -primary "HxUniformScalarField3 HxStackedScalarField3" \
    -category "{Image Segmentation}" \
    -class "HxLabelVoxel" \
    -dso "libhxlattice.so"
```

The different options of this command have the following meaning:

- The option `-name` specifies the name or label of the module as it will be printed in the popup menu.

- The option `-primary` says that this module can be attached to data objects of type `HxUniformScalarField3` or `HxStackedScalarField3`. This means that *Multi-Thresholding* will be included in the popup menu of such objects only.

- With `-check`, an additional Tcl expression is specified which is evaluated at run-time just before the menu is popped up. If the expression fails, the module is removed from the menu. In the case of the *Multi-Thresholding* module, it is checked if the input object provides a `HxLabelLattice3` interface, i.e., if the input itself is a label field. Although a label image can be regarded as a 3D image, it makes no sense to perform a threshold segmentation on it. Therefore, *Multi-Thresholding* is only provided for raw 3D images, but not for label fields. There is also a check on the primitive data type of the input (signed/unsigned integer, float, signed/unsigned short...). Here, the *Multi-Thresholding* module does not support float or double label images input.

- The option $-$category says that *Multi-Thresholding* should appear in the *Image Segmentation* submenu of the main popup menu. If a module should appear not in a submenu but in the popup menu itself, the category Main must be used.
- The option $-$class specifies the internal class name of the module. The internal class name of an object can be retrieved using the command getTypeId. It is this class name which has to be used for the $-$primary option described above, not the object's label defined by $-$name.
- Finally, the option $-$dso specifies in which shared library or DLL the module is defined. The option $-$package can be used instead, specifying in which package the module is defined (ex: -package "hxlattice").

Besides these standard options, additional Tcl commands to be executed after the module has been created can be specified using the additional option $-$proc. For example, imagine you are working in a medical project where you have to identify stereotactic markers in CT images of the head. Then it might be a good idea to add a customized version of the *Multi-Thresholding* module to the popup menu, which already defines appropriate material names and thresholds. This could be done by adding the following command either in a new resource file in $AVIZO_ROOT/share/resources or directly in hxlattice.rc:

```
module -name "Stereotaxy" \
    -primary "HxUniformScalarField3 HxStackedScalarField3" \
    -check { ![$PRIMARY hasInterface HxLabelLattice3] } \
    -category "{Image Segmentation}" \
    -class "HxLabelVoxel" \
    -package "hxlattice" \
    -proc { $this regions setValue "Exterior Bone Markers";
            $this fire;
            $this boundary01 setValue 150;
            $this boundary12 setvalue 300 }
```

The variable $this used in the Tcl code above refers to the newly created module, i.e., to the *Multi-Thresholding* module. Note that the commands are executed *before* the module is connected to the source object for which the popup menu was invoked. Some modules do some special initialization when they are connected to a new input object. These initializations may overwrite values set using Tcl commands defined by a custom $-$proc option. In such a case, you can explicitly connect the module to the input object via the command sequence

```
    $this data connect $PRIMARY;
    $this fire;
```

Here the Tcl variable $PRIMARY refers to the input object. The same variable is also used in Tcl expressions defined by a $-$check option, as described above.

Besides creating custom popup menu entries based on existing modules, it is also possible to define completely new entries which do nothing but execute Tcl commands. For example, we could add a new submenu *Edit* to the popup menu of every Avizo object and put in the *Hide*, *Remove*, and *Duplicate* commands here which are normally contained in the *Edit* menu of the Avizo main window. This can be achieved in the following way:

```
module -name "Remove" \
    -primary "HxObject" \
    -proc { remove $PRIMARY } \
    -category "Edit"

module -name "Hide" \
    -primary "HxObject" \
    -proc { $PRIMARY hideIcon } \
    -category "Edit"

module -name "Duplicate" \
    -primary "HxData" \
    -proc { $PRIMARY duplicate } \
    -category "Edit"
```

Of course, it is also possible to execute an ordinary Avizo script or even an Avizo script object with a `-proc` command.

9.4.7   Registering pick callbacks

A pick callback is a Tcl procedure attached to a module or a viewer. When a pick event occurs on this target, the callback is invoked. Such a callback can be registered by using the Tcl command `setPickCallback` on modules and viewers:

```
<module> setPickCallback <proc> [ <EventType> ]
viewer <n> setPickCallback <proc> [ <EventType> ]
```

Only one callback can be attached to a given module or viewer. In order to detach the callback, just call the register command with no parameter:

```
<module> setPickCallback
viewer <n> setPickCallback
```

The optional argument `<EventType>` refers to the kind of event that will invoke the callback. Other events will be ignored. This argument can take the following values:

- MouseButtonPress, MouseButtonRelease (any mouse button),
- VRButtonPress, VRButtonRelease (any 3D button),
- MouseButton1Press, MouseButton1Release, etc. (a specific mouse button),
- VRButton0Press, VRButton0Release, etc. (a specific 3D button).

The default value is MouseButton1Press.

The actual callback procedure <proc> is expected to take one argument, which is to be interpreted as an associative array and which encodes all the picking information. The following elements are defined in the argument array:

- *object*, the name of Avizo object the picked geometry belongs to,
- *x*, the x coordinate of picked point,
- *y*, the y coordinate of picked point,
- *z*, the z coordinate of picked point,
- *idx*, the index of picked element,
- *stateBefore*, the modifier state just before event occurs,
- *stateAfter*, the modifier state just after event occurs.

The procedure should return 0 if the picking event was not handled, in which case other callback procedures could be invoked. Here is an example:

```
proc pickCallback arg {
    array set a $arg
    echo "$a(object) : picked element $a(idx)"
    return 1
}
```

Note that any module is free to add specific information to this argument array. All elements can be displayed with:

```
proc pickCallback arg {
    echo "arg = { $arg }"
    return 1
}
```

Thus, some Avizo modules append additional data:

- *Vertex View: idx* is the picked point index.
- *Point Cloud View: idx* is the picked point index.
- *Line Set View: idx* is the picked line index, *pt0* and *pt1* the two points of the picked segment.

- *Surface View: idx* is the picked triangle index.
- *Hexa/Tetra Grid View: idx* is the picked triangle index, *tetra0* and *tetra1* the adjacent tetrahedra.
- *Grid Boundary: idx* is the picked triangle index, *originalIdx* the index in the grid, *tetra0* and *tetra1* the adjacent tetrahedra.

9.4.8   File readers in Tcl

This section describes how to register a custom file reader implemented in Tcl.

First the Tcl reader function must be declared in the global scope. It must accept a list as input parameters which will contain the list of files to load. The reader is expected to return the number of files successfully read.

```
proc myReaderInTcl {args} { echo "myReaderInTcl $args" ; ... ;
    return 1}
```

Then, the Tcl reader function must be registered into the wanted file format declaration with the following template:

```
dataFile -name "MyFormat" ... -package hxcore -load hxReadByTcl
    -loadArgs "-cmd myReaderInTcl"
```

You indicate the custom Tcl reader function with '-loadArgs'. The arguments '-package hxcore -load hxReadByTcl' must be filed as is, without change. This sets the internal wrapper that will call the Tcl interpreter.

You can declare your custom reader in a Tcl script, or include it in a resource file to be loaded when the application starts up.

9.4.9   How to Create Recipe-Compliant Script-Object

The Scripting Workroom allows for the creation of script-objects, which can be considered to be custom tools since they can take input data to produce results.

Usually, script-objects are not compliant with the recipe system since the recipe cannot be created from a result of a workflow using a script-object. Some rules need to be respected to create a script that can be used in the recipes.

Note: You can also create a custom module using the *Tcl Command* module. This module should be recipe-compliant unless you are in one of the advanced rules use cases described below, or if you need more than one input.

The script object should apply the following rules:

- *Having an Apply button*

- *Creating a result*

Some other rules should be applied if your create non-standard scripts:

- *Using inhibitors*
- *Modifying the input*

9.4.9.1 Having an Apply button

The script-object should have an Apply (i.e., portDoIt) button. The computation should be launched only if this button has been clicked.

In terms of code:

- Create the Apply (i.e., doIt) port in the script-object constructor
- At the beginning of compute method: abort if the Apply button has not been clicked.

```
"$this" proc constructor {} {
    "$this" select
    "$this" script hide

    # Apply button, mandatory to have a recipe-compliant script
    "$this" newPortDoIt doIt
}

"$this" proc compute {} {
    # If "Apply" button wasn't hit: nothing is done
    if { !["$this" doIt wasHit] } {
        return
    }
    ...
    ...
}
```

9.4.9.2 Creating a Result

The recipe system is based into the HistoryLog of a data. The HistoryLog is a record of each step or tool that has been applied to produce the data.

The script-object should create a result dataset, and set it as result by using the setResult command. The setResult command will correctly record the HistoryLog of the data.

In terms of code:

- Use "`$this` setResult <resultName>" at the end of the compute method.

```
$this proc compute {} {
```

```
    # If "Apply" button wasn't hit: nothing is done
    if { !["$this" doIt wasHit] } {
        return
    }

    # If nothing is connected to the tool: nothing is done
    if { ["$this" data source] == "" } {
        return;
    }

    # Do some things, and store the result into 'result' var.
    ...

    # set the result
    "$this" setResult $result
}
```

The sections below describe how to handle some particular scripts, which modify the script connections or the input data.

9.4.9.3 Advanced Use: Inhibit Internal Tools

In some particular case, the recipe created from the script result contains *"internal steps"* since the internal tools used in the script-object appear in the recipe.

This appends when the connection port of the script-object is modified by the script.

If such a use case occurs, and if there is no reason these tools should appear in the recipe, the internal tool recording should be inhibited using the startLogInhibitor and stopLogInhibitor TCL commands. The script setResult command should be called after the stopLogInhibitor command.

In terms of code:

- Start the inhibitor in the compute method before creating any other tool.
- Stop the inhibitor before setting the result.

```
$this proc compute {} {

    # If "Apply" button wasn't hit: nothing is done
    if { !["$this" doIt wasHit] } {
        return
    }

    # If nothing is connected to the tool: nothing is done
    if { ["$this" data source] == "" } {
        return;
    }

    # Make the script recipe-compliant: start the inhibitor
```

```
    startLogInhibitor

    # Do some things using internal tool, including modifying the data
    # connected to your script, then store the result into 'result' var.
    ...

    # Make the script recipe-compliant: stop the inhibitor
    stopLogInhibitor

    # set the result
    "$this" setResult $result
}
```

9.4.9.4   Advanced Use: Script Modifying the Data

If you want to modify the input of the script-object (e.g., by changing its transformation), the setResult rule cannot be applied since the result of a tool cannot be connected as an input.

In this use case, to make the script-object recipe-compliant, we need to manually record the recipe step, by using the recordInputUpdateInHistoryLog command.

In terms of code:

```
$this proc compute {} {

    # If "Apply" button wasn't hit: nothing is done
    if { !["$this" doIt wasHit] } {
        return
    }

    # If nothing is connected to the tool: nothing is done
    if { ["$this" data source] == "" } {
        return;
    }

    # Do some things which modify the input data
    ...

    # Make the script recipe-compliant: record the step
    "$this" recordInputUpdateInHistoryLog data
}
```

9.4.10   Versioning of Script Objects and backward compatibility

With the release of Avizo 9.2 many modules have been reworked with respect to their user interface which required changing names and types of ports. As a consequence, scripts and script objects written for earlier versions of Avizo than 9.2 would in many cases no longer work. To account for this a backward compatibility mode has been implemented that requires the script objects to have a version number. For script objects before 9.2 the version string was V3.0. Beginning with 9.2 a 3 digit number composed from the program version without separating dots will tell Avizo whether or not to use the

backward compatibility mode. Script objects with the line `# Avizo-Script-Object 940` in the first line will use the new interface while those with `# Avizo-Script-Object V3.0` will use the compatibility mode for some modules. For recent port list consult the module documentation. The modules using the compatibility mode are given in the table:

Module Name	Type ID
2D Mesh	Hx2DMesh
Absolute Permeability Experiment Simulation	HxApparentAbsolutePermeability
Absolute Permeability Tensor Calculation	HxEffectiveAbsolutePermeability
Align Molecules	HxAlignMolecules
Align Principal Axes	HxAlignPrincipalAxes
Atomic Molecular Density	HxCompMolecularDensity
Auto Skeleton	HxExtAutoSkeleton
Bar Chart Slice	HxCityPlot
Bond Angle View	HxBondAngle
Boundary Conditions	HxHexaBoundaryIds
Boundary View	HxUnstructuredBoundariesView
Centerline Tree	HxTEASAR
Clipping Plane	HxArbitraryCut
Compute Tensor	HxComputeTensor
Configuration Density	HxConfDensity
Cross Section	HxUnstructuredMeshCrossSection
Curved Slice	HxCurvedSlice
Cylinder Correlation	HxCylinderCorrelation
Cylinder Slice	HxCylinderSlice
Effective Formation Factor Calculation	HxEffectiveFormationFactor
Elastic Registration	HxElasticRegistration
Export To VRML	HxSurfaceToVRML
Formation Factor Experiment Simulation	HxApparentFormationFactor
Generate Molecular Interface	HxCompMolInterface
Generate Surface	HxGMC
GridVolume	HxPoMeshSkin
H Bond View	HxHBondView
Height Map Slice	HxHeightField
Hexa Grid View	HxHexaView
Isocontour Slice	HxIsolines
Isosurface	HxIsosurface
Isosurface	HxPoMeshLevelSurf
Isosurface (hexa)	HxIsoHexa
Isosurface (tetra)	HxIsoTetra
Line Probe	HxLineProbe
Line Set Probe	HxLineSetProbe
Local Axis, Global Axis	HxAxis
Merge Mosaic	HxMergeMosaic
Molecular Diffusivity Experiment Simulation	HxApparentMolecularDiffusivity
Molecular Diffusivity Tensor Calculation	HxEffectiveMolecularDiffusivity
Molecule Electrostatics	HxMolElectrostatics
Molecule Label	HxMolLabel
Molecule Surface View	HxMolSurfaceView
Molecule View	HxMolView
Observables	HxMolObservables

Ortho Slice	HxOrthoSlice
Ortho Slice (lattice)	HxLatticeOrthoSlice
Ortho Slice (LDM)	HxOrthoSliceLDM
Parametric Surface	HxParametricSurface
Plot In Viewer	HxPlot2Viewer
Point Cloud View	HxClusterView
Pseudo Electron Density	HxPseudoElectronDensity
Register Images	HxAffineRegistration
Remesh Surface	HxRemeshSurface
Scalebars	HxScale
Secondary Structure View	HxSecStructure
Slice	HxFilteredObliqueSlice
Slice (LDM)	HxObliqueSliceLDM
Spatial Graph View	HxSpatialGraphView
Spline Probe	HxSplineProbe
Surface Cross Contour	HxGeometryCutter
Surface Path Editor	SurfacePathEditor
Surface View	HxDisplaySurface
Tetra Grid To Surface	HxTetraToSurface
Tetra Grid View	HxGridVolume
Thermal Conductivity Experiment Simulation	HxApparentThermalConductivity
Thermal Conductivity Tensor Calculation	HxEffectiveThermalConductivity
Transform Sequence	HxObjectTransformAnimation
Thinner	HxExtThinner
Trace Correlation Lines	HxTraceCorrelationLines
Tube View	HxTubeView
Vector Field	HxUnstructuredMeshVectors
Vector Plane	HxUnstructuredMeshPlaneVectors
Vertex View	HxDisplayVertices
Volren	HxVolren
Volume Rendering Settings	HxVolumeRenderingSettings
Vortex Corelines	HxVortexCoreline

9.5 Python Scripting

This section describes how to use *Python Scripting* in Avizo.

9.5.1 Python Documentation

This chapter is organized as follows:
Section 9.5.1.1 Introduction to Python
Section 9.5.1.2 Python in Avizo
Section 9.5.1.3 Common Global Commands
Section 9.5.1.4 Modules Management
Section 9.5.1.5 Script Objects
Section 9.5.1.6 Python Package Manager: Enstaller
Section 9.5.1.7 List of Python Packages
Section 9.5.1.8 Python Integration Limitations

The Avizo Python API Documentation is available here

9.5.1.1 Introduction to Python

What is Python
Python is a high-level, object-oriented, interpreted language first implemented in 1989.
(https://www.python.org/dev/peps/pep-0020/)

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

What is Included
Avizo utilizes Python 2.7.11. Detailed information on how to use Python 2.X can be found here:
https://docs.python.org/2/tutorial/index.html. Many packages are included in the Python installation
(See the List of Python Packages *here*). Numpy and Scipy are two of the most popular Python
packages included in this installation.

Numpy is an extension for handling multi-dimensional array, which allows for elementwise operations, comparisons, logical operations, and statistics among others. The numpy array is implemented
in C allowing for faster computation. More information can be found here: http://www.numpy.org/.

Scipy is an extension that provides a toolbox for scientific computing such as interpolation,

integration, image processing, linear algebra, signal processing, and statistics. Creating additional GUIs for viewing plots is not currently supported. More information can be found here: http://www.scipy.org/.

9.5.1.1.1 Using Python

This section is not meant to cover all applications and details of the Python language. See the links in the previous section to learn more about Python, Scipy, and Numpy. To learn more about how Python interacts with Avizo, continue to chapter 9.5.1.2 Python in Avizo.

Python Console
The console can be accessed by going to Windows > Consoles (see Figure 9.4). The consoles panel has a tabbed interface with several different tools available (see Figures 9.5 and 9.6).
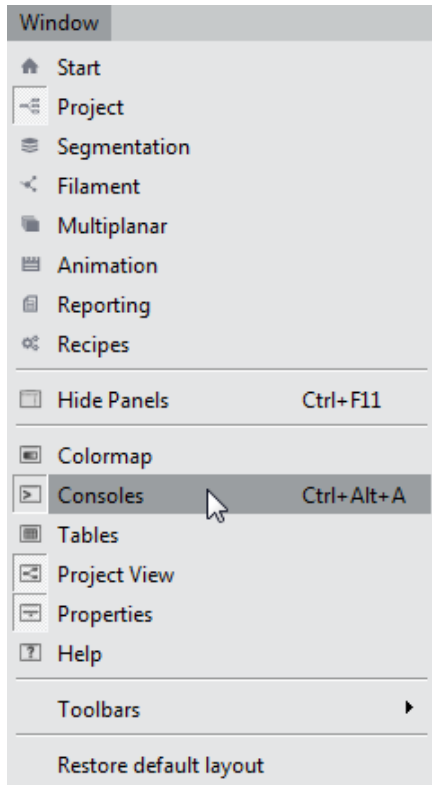
Figure 9.4: Accessing the Consoles
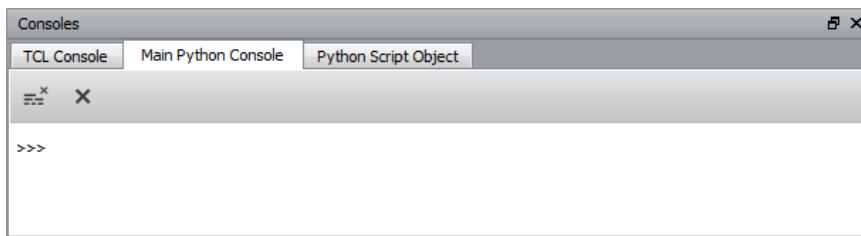
Figure 9.5: Main Python Console Interface
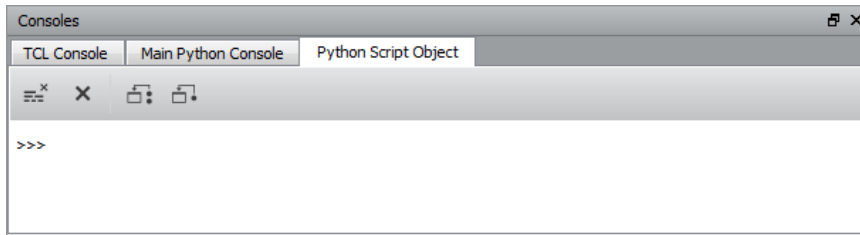
Figure 9.6: Python Script Object Console Interface

    Remove This content: Remove all content displayed in this interpreter

    Remove Pythonic Objects: This removes all python objects created within the console from Avizo's workspace

    Redirect All Output: This pushes all output from all interpreters to the main console

    Redirect This Output: This pushes all output from the current console to the main console

The console acts as a Python interpreter, so any written commands will be immediately executed after pressing **Enter**. The assignment, or return value, will be displayed after execution.

Hotkeys and Useful Commands

TAB
When nothing is written in the console, **TAB** will auto-complete the following command hx_project.get(module), which will create a Python handle for the object. When text exists in the console, **TAB** will attempt to auto-complete the current string from a list of available methods, attributes, and modules. The highlighted choice from the pulldown list will be completed.

UP or **DOWN**
These buttons cycle through your recent history. **UP** retrieves your previous command, while **DOWN** retrieves the subsequent command.

9.5.1.1.2   Notes on Python

Packages
Python is an object-oriented language, meaning that code is typically broken down into classes in which variables and methods can be inherited by objects at instantiation.

This allows you to avoid re-coding functionality that you use often. More information about OOP in the context of Python can be found here:

http://www.tutorialspoint.com/python/python_classes_objects.htm

Packages are collections of classes, methods, and variables that can be imported to a namespace in Python. For example, if the user wants to calculate $\sin(\pi/2)$, they first need to import the Numpy package into the namespace:

```
>>>import numpy
```

The Numpy package contains a `sin(x)` method that can then be accessed to calculate $\sin(\pi/2)$:

```
>>>numpy.sin(3.1415/2)
0.999999998927
```

Numpy already contains a global variable defining π that you can access in the same way:

```
>>>numpy.pi
3.14159265359
>>>numpy.sin(numpy.pi/2)
1.0
```

Packages can be assigned to variables when imported to the namespace to simplify code:

```
>>>import numpy as np
>>>np.sin(np.pi/2)
1.0
```

Syntax

Python supports several different object types:

Type	Description	Example
Number	Integers, floats, complex, booleans	`1, 1.05, 3j+2, 3>2 is True`
String	Sequence of characters	`"String of charaters"`
List	Container to group items that can be changed	`[1, 5, "Dragon", 948.5]`
Tuple	Container to group items that cannot be changed	`(948.5, "Dragon", 5, 1)`
Dictionary	Associated arrays with unique keys	`{'a':99, 'b':'red', 'c':'balloons'}`
Array	Vectorized numeric array optimized for C	`numpy.ones ((10,5))`

Some types are denoted using specific characters. For example, single or double quotes are used to create strings:

```
>>>a = 'This is a string.'
>>>b = "This is also a string."
```

While lists are assigned using brackets:

```
>>>c = [1, 5, "Dragon", 948.5]
```

Operations can be performed between objects using standard syntax:

+	addition	==	equal
-	subtraction	!=	not equal
*	multiplication	>	greater than
/	division	<	less than
**	exponent	<=	less than or equal to
%	modulus	>=	greater than or equal to

Some keywords are reserved for global variables or perform specific functions. Overwriting these keywords as variables should be avoided at all costs.

and	as	assert	break
class	continue	def	del
elif	else	except	exec
finally	for	from	global
if	import	in	is
lambda	not	or	pass
print	raise	return	try
while	with	yield	

A more thorough discourse on Python syntax and object types can be found in the official tutorial: https://docs.python.org/2/tutorial/index.html

9.5.1.2 Python in Avizo

9.5.1.2.1 Overview
Similar to TCL, Python has been implemented in Avizo using a Pythonic API. Commands specific to Avizo allow you to access information and functionalities contained within Avizo modules. There are two main ways to interact with Avizo using Python. The first way to use Python is through the Python console, which is separate from the TCL console. This is an interpreter. The basic functionality of this console, such as tab completion, is described in chapter 9.5.1.1 Introduction to Python.

The second way to use Python in Avizo is through script modules. Python script modules allow you to inherit properties from the pre-defined PyScriptObject class before you override them to create their own integrated extensions to Avizo. Script modules act like regular modules in Avizo and can be accessed in the *Object Popup* menu when accompanied by a resource file. The resource file still must be written in TCL. There are four methods included in the PyScriptObject class for your convenience:

- `__init__()`: This constructor method can contain code that is run when the script object is created in the Project View.
- `update()`: The update method can be invoked to update the script object's GUI in the *Properties Panel*.
- `compute()`: The compute method usually contains the bulk of your code and should be invoked when computation must occur.
- `__del__()`: This destructor method can contain code that helps clean up the namespace at module deletion.

9.5.1.2.2 Example: Interacting with Modules

Here we will go through an example of how the console can be used to interact with modules in the Project view by creating an Ortho Slice and changing its properties.

1. Ensure that the *Python Console* is shown in the workspace
2. Open $`AVIZO_ROOT`/data/tutorials/chocolate-bar.am

   - If Auto-View is enabled, an Ortho Slice is automatically created. Delete it.

3. To create an Ortho Slice, access the `create()` method in the HxProject class. The `create()` method requires us to pass the type ID of the object we want to create as a string argument.

   - If you are unsure what an object's type ID is, you can create the object in the GUI then use the `<module> getTypeId` command in the TCL console to learn its type.

     ```
     >>>hx_project.create('HxOrthoSlice')
     ```

4. Now we need to connect Ortho Slice to chocolate-bar.am, but we will first assign the Ortho Slice to a variable using the `get()` method to make it easier to access in the future. We will do the same for chocolate-bar.am:

     ```
     >>>ortho = hx_project.get('Ortho Slice')
     >>>input_data = hx_project.get('chocolate-bar.am')
     ```

5. To connect the Ortho Slice to chocolate-bar.am, we will need to see how to access the Data port of Ortho Slice. Expose a list of possible ports to interact with by printing the results of the portnames command to the console:

     ```
     >>>ortho.portnames
     ['data', 'origin', 'normal', 'frameSettings',
     'planeDef.', 'planeDefinition', 'planeNormal',
     ```

```
'planePoint1', 'planePoint2', 'planePoint3',
'planeVector1', 'planeVector2', 'planeDistance',
'sliceOrientation', 'options', 'depth',
'specularShininess', 'specularIntensity',
'mappingType', 'contrastLimit', 'colormap',
'sliceNumber', 'transparency', 'alpha', 'channels']
```

6. From the `portnames` command we see that the `'data'` port likely coincides with the "Data" connection exposed in the Ortho Slice's properties. Connect chocolate-bar.am to this port using the `connect()` method at the `ports` level, then apply the change with the `fire()` method:

```
>>>ortho.ports.data.connect(input_data)
>>>ortho.fire()
```

7. We can also change the slice location by setting the `sliceNumber` port's `value`:

```
>>>ortho.ports.sliceNumber.value = 100
>>>ortho.fire()
```
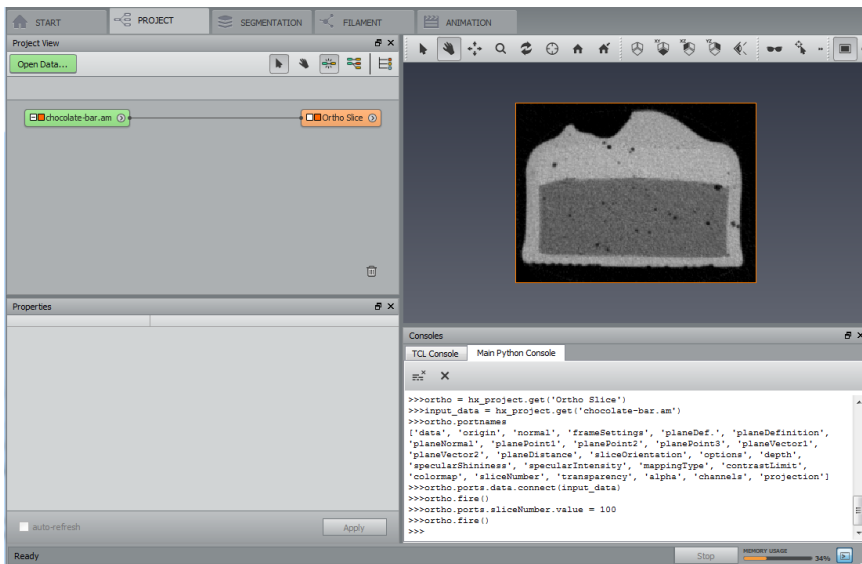

Figure 9.7: Interacting with Modules in Python Console

8. Experiment with the other ports in Ortho Slice to see if you are able to change the slice orientation and the colormap. For help accessing these ports, pass the access command to the `help()` method:

```
>>>help(ortho.ports.sliceOrientation)
```

9. Read further on the methods and classes available for your manipulation by passing them to the `help()` method:

```
>>>help(ortho.fire)
```

9.5.1.2.3 Example: Developing a Script

In this example, we will write a simple script to calculate the total volume of the bounding box of a dataset. This can be typed directly in the Python console in Avizo. You can also type the commands in a text editor and copy them to the Python console for execution.

1. Load $AVIZO_ROOT/data/tutorials/chocolate-bar.am.
2. Assign chocolate-bar.am to a variable to quickly refer to it in subsequent code.

```
>>>data = hx_project.get('chocolate-bar.am')
```

3. Create a Bounding Box and attach it to chocolate-bar.am.

   - You can assign the Bounding Box to a variable at the same time you create it, instead of creating it then using the `get()` method in a separate command later.

```
>>>bbox = hx_project.create('HxBoundingBox')
>>>bbox.ports.data.connect(data)
>>>bbox.fire()
```

4. Retrieve the extents of the bounding box in X, Y, and Z to calculate its volume. There is already a bounding_box command you can use to get this information, which is stored as a tuple defined as ((xmin, ymin, zmin),(xmax, ymax, zmax)).

```
>>>type(data.bounding_box)
<type 'tuple'>
>>>data.bounding_box
((0.0, 0.0, 0.0),
 (28.079999923706055,
  20.880001068115234,
  35.279998779296875))
```

5. Python also makes it easy to extract two variables at once from this two-list tuple.

```
>>>min_bounds, max_bounds = data.bounding_box
```

6. Access each of the indices from the minimum and maximum boundary lists using brackets []
   and plug this information into a formula to calculate the box's volume.

```
>>>x_extent = max_bounds[0] – min_bounds[0]
>>>y_extent = max_bounds[1] – min_bounds[1]
>>>z_extent = max_bounds[2] – min_bounds[2]
>>>volume = x_extent * y_extent * z_extent
```

7. Finally, print the information to the console in a refined format (See Figure 9.8).

```
>>>print('The volume of %s is %.3f. ' % (data.name, volume))
```
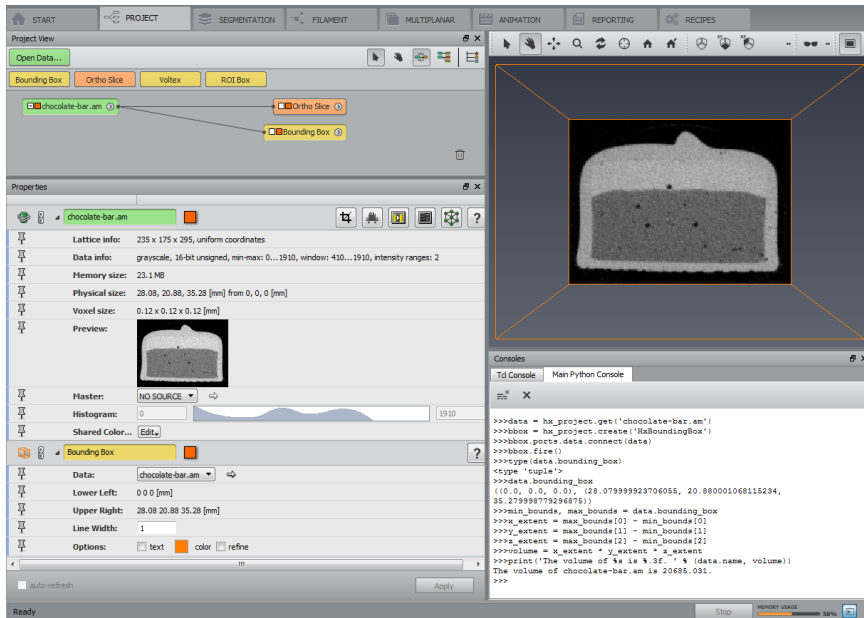

Figure 9.8: Object Information by Python Console

9.5.1.2.4 Example: Creating a Function

In this example, we wrap the bounding box volume calculator into a function that accepts our input
data as an argument and prints our answer back into the console. A few notes about functions and

Python syntax:

- If you do this directly in the Avizo console, press **SHIFT+ENTER** to enter a line break in your code without executing it.
- Python requires consistent indentation within the body of a function. The best practices convention is to indent the body of code by four spaces (not a tab).
- The Python console in Avizo does not indent lines for the user, and the user must control indentation themselves.

1. Load $AVIZO_ROOT/data/tutorials/chocolate-bar.am
2. Use the `def` keyword to define a function called `bbVol` that takes a single input dataset as an argument.

```
>>>def bbVol(data_arg): # Remember to SHIFT+ENTER here!
...
```

3. An ellipses is shown in the interpreter to show that it is waiting for more code before executing the current input. Manually type four spaces, then assign the minimum and maximum bounding box lists to variables as before.

```
...    min_bounds, max_bounds = data_arg.bounding_box
```

4. Pressing **SHIFT+ENTER** to make a new line, enter another four spaces, and proceed with the rest of the volume calculator.

```
...    x_extent = max_bounds[0] - min_bounds[0]
...    y_extent = max_bounds[1] - min_bounds[1]
...    z_extent = max_bounds[2] - min_bounds[2]
...    volume = x_extent * y_extent * z_extent
...    print('The volume of %s is %.3f. ' % (data_arg.name, volume))
```

5. Finally, include a `return` statement to send the volume variable back to the console. This allows you to set the result of the calculation to a variable.

```
...    return volume
```

6. Execute the function definition by pressing **ENTER**, then test the code with chocolate-bar.am. Test your ability to assign the calculation result to a variable as well (See Figure 9.9).

```
>>>bbVol(hx_project.get('chocolate-bar.am'))
The volume of chocolate-bar.am is 20685.031.
20685.031198228968
>>>chocolatebar_volume = bbVol(hx_project.get('chocolate-bar.am'))
```

Figure 9.9: Creating a Script Module

9.5.1.3   Common Global Commands

There are two mains functions that greatly help the user to explore the structure of Python within Avizo. The `dir()` function allows the user to see a list of attributes and methods that are available for a given object.

```
>>>ortho = hx_project.get('Ortho Slice')
>>>dir(ortho)
['__class__', '__delattr__', '__doc__', '__format__',
'__getattribute__', '__hash__', '__init__', '__new__',
'__pyx_vtable__','__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'_cpp_classname', '_cpp_package', '_tcl_interp',
'all_interfaces', 'can_be_renamed', 'clip_geometry',
'compute', 'create_doc_file', 'create_port_snaps',
'downstream_connections', 'duplicate', 'execute',
'fire', 'get_all_interface_names', 'icon_color',
'icon_position', 'icon_visible', 'is_geometry_clipped',
'is_same_object', 'name', 'need_saving', 'portnames',
'ports', 'removable', 'selected', 'unclip_geometry',
'update', 'viewer_mask']
```

This information is also available in a pulldown list when you begin typing a command (See Figure 9.10).
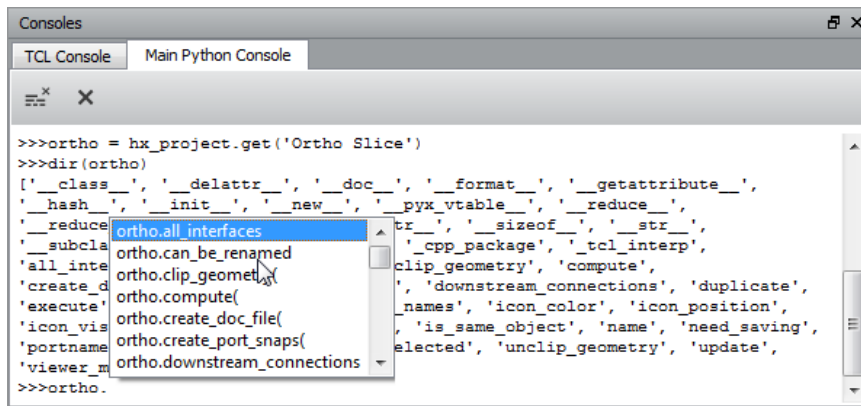
Figure 9.10: Pulldown List of Commands

The `help()` function has more detailed information about attributes and methods and examples of their use. Output from `help()` also contains information about related parent classes.

```
>>>help(ortho)
class HxPlanarModBase(HxModule)
| Method resolution order:
| HxPlanarModBase
| HxModule
| HxObject
| HxBase
| McInterface
| __builtin__.object
|
| Data and other attributes defined here:
|
...
...
...
| all_interfaces
| Attribute that contains all admissible interfaces as sub-members.
|
| Examples
```

```
| --------
|
| Retrieve an 'HxBase' interface on an orthoslice:
|
| >>> ortho = hx_project.create('HxOrthoSlice')
| >>> base = ortho.all_interfaces.HxBase
```

A list of some more specific global commands is provided below.

- `print()` is a native Python command that prints results to the Python console in Avizo.

    ```
    >>>x = 3
    >>>y = 2
    >>>print('The sum of %i + %i is %i.' % (x,y,x+y))
    The sum of 3 + 2 is 5.
    ```

- `import` is a native Python command that adds extended functionality to Python by loading extra packages.

    ```
    >>>x = 3
    >>>y = 2
    >>>import numpy
    >>>numpy.add(x,y)
    5
    ```

Some useful global functions for interacting with the Project view are contained in the `hx_project` method.

Method	Description
`hx_project.create()`	Creates an instance of a certain Avizo class and adds it to the Project view
`hx_project.remove()`	Removes an object from the Project view
`hx_project.add()`	Adds an object to the Project view
`hx_project.load()`	Loads data of a defined format when a filename is specified

Variables containing directory paths are also provided as attributes accessible via the `hx_paths` method.

Variable	Description
hx_paths.install_dir	Installation directory, also known as <$AVIZO_ROOT>
hx_paths.tutorials_dir	Tutorials data directory
hx_paths.python_modules_dir	Python modules directory that contains extra packages
hx_paths.python_script_objects_dir	Python script objects directory that contains user-created custom Python scripts
hx_paths.executable_dir	Directory containing Avizo.exe

```
>>>hx_paths.install_dir
'C:/Program Files/<INSTALL_DIR>'
>>>hx_paths.tutorials_dir
'C:/Program Files/<INSTALL_DIR>/data/tutorials'
>>>hx_paths.python_modules_dir
'C:/Program Files/<INSTALL_DIR>/share/python_modules'
>>>hx_paths.python_script_objects_dir
'C:/Program Files/<INSTALL_DIR>/share/python_script_objects'
>>>hx_paths.executable_dir
'C:\\Program Files\\<INSTALL_DIR>\\bin\\arch-Win64VC12-Optimize\\Avizo.exe'
```

9.5.1.4 Modules Management

9.5.1.4.1 Module Attributes

What is an Attribute
An attribute is a data field contained within a class. Some attributes may be read-only.

List of Common attributes
Here the variable 'a' refers to the python handle for your object.

a.**name**:
This is a string property that refers to the name of your Avizo object. A string may be assigned to change the display name of the object.

a.**portnames**:
This is a read-only list of all the port names that belong to the Avizo object.

a.**viewer_mask**:
This is an integer property that triggers the visibility in the viewer. There are 16 configurations that can be set from [0, 15]. If a number outside of that range is used, the number will be evaluated after the modulo 16 operation.

a.**bounding_box**:
This stores a pair of tuples that describe the spatial dimensions of your object. New dimensions could

be assigned in the format of $((x_{min}, y_{min}, z_{min}), (x_{max}, y_{max}, z_{max}))$.

a.**downstream connections[**x**]**:
This stores a read-only sequence of all objects that refer to it. Each connection is assigned an index 'x' which is an integer. To find the name of the object from the pointer, use the command a.**downstream connections[**x**].get owner().name**

a.**range**:
This stores a read-only tuple showing the intensity range of the data in the form (min, max).

a.**transform**:
This stores a tuple showing the 4x4 transformation matrix. A new transformation can be assigned.

9.5.1.4.2 Module Methods

What is a Method
A method is a function contained within a class. Many methods do not need an argument.

List of Common Methods
Here the variable 'a' refers to the python handle for your object.

a.**compute()**:
This method performs the computation of the object if a.**ports.doIt.was hit** = True. This simulates clicking **Apply** of the object when conditions permit.

a.**update()**:
This method updates the object GUI for the properties window.

a.**fire()**:
This method calls a.**update()** and a.**compute()**.

a.**execute()**:
This method combines all of the above methods to simulate a click of **Apply** and a refresh of the GUI.

a.**get array()**:
This method accesses the NumPy array of your object. Accessing the array will prevent deletion of the object.

a.**set array(...)**:
This method assigns a NumPy array to your object. Ownership of the array is not passed, so future changes to the array do not propagate unless reassigned.

9.5.1.5 Script Objects

9.5.1.5.1 What is a Python Script Object
A python script object is a compute module whose behavior is hardcoded in a Python class inherited from PyScriptObject. Python script objects are useful for creating custom tools whose functionality is accessed the same way that Avizo compute modules are accessed.

Script Structure
The following class methods are useful when defining your python script object as a class, but it is not required to define it in such a way. Any script will run just as if it was typed into the console. The variable `self` refers to its own class identity.

def __init__(`self`):
This method is called when the object is created. It is a useful area to define the structure of your script and setup the GUI.

def __del__(`self`):
This method is called when the object is restarted or deleted. This is a good place to cleanup any connections.

def update(`self`):
This method is called when the GUI needs to be refreshed.

def compute(`self`):
This method is called when **Apply** is clicked.

Creating Ports
The user interfaces with the object through ports. The ports can be initialized through simple assignment, and they belong to the `HxPort` class. A few useful ports are below:

- `HxConnection`
  This port class allows the user to connect modules. The type of module can also be restricted.
- `HxPortFilename`
  This port class allows the user to load or save files. The function of the port is defined with the **mode** attribute. The filename can either be input as text or accessed through a file browser.
- `HxPortIntSlider`
  This port class holds a range of integer values that can be accessed through a sliding scale. This port is used to define the slice number of the "Ortho Slice" module.
- `HxPortDoIt`
  This port class controls the auto-refresh box and handles the **Apply** button.
- `HxPortInfo`
  This port class is useful to provide instructions, notes, or warnings to the user. The text can be found in the module properties.

Bounding Box Script Example
In this example, we wrap the bounding box volume calculator (described in chapter : *"Python in Avizo "*) into a script module that can be loaded into the Avizo GUI via the Object Popup menu.

1. Open a text editor and create a new file with the following structure. Name the object BoundingBoxVolume.

    - Alternatively,   you   can   start   with   a   template   in

Python Scripting   **517**

```
class BoundingBoxVolume(PyScriptObject):

    def __init__(self):
        # Initialization code will go here

    def update(self):
        # Update code will go here

    def compute(self):
        # Computation code will go here
```

2. The script object needs to be able to connect to a data object to know which bounding box needs the volume calculated. By default, Python script objects inherit a data connection from the PyScriptObject class. Make sure it is visible in the __init__() method.

```
    def __init__(self):
        self.data.visible = True
```

3. In this example, use the pass keyword to skip updating the GUI in the update() method since there will be no ports to update.

```
    def update(self):
        pass
```

4. In the compute method, we want to halt the volume computation if there is no data object connected to the script module. Add an if statement that checks if the data connection is empty. If the data connection is empty, use a return statement to exit the compute() method.

- Python makes this easy with the None keyword (i.e., if <expression> is None)

```
    def compute(self):
        # Check if input data is connected
        if self.input.source() is None:
            return
```

5. Finally, if the logic check fails because a data object is attached, calculate the volume using the

bbVol script. Get the name of the connected data object using the `source()` method.

```python
def compute(self):
    # Check if input data is connected
    if self.input.source() is None:
        return

    data = self.input.source()
    min_bounds, max_bounds = data.bounding_box
    x_extent = max_bounds[0] - min_bounds[0]
    y_extent = max_bounds[1] - min_bounds[1]
    z_extent = max_bounds[2] - min_bounds[2]
    volume = x_extent * y_extent * z_extent
    print('The volume of %s is %.3f. ' % (data_arg.name, volume))
```

6. Once your module is complete, create a resource file in TCL that displays this module in the Object Popup menu as an option to attach to all data objects.

- Refer to *Configuring Popup Menus* to learn more about creating resource files.
- This file can be found in:
  $AVIZO_ROOT/share/resources/PythonBoundingBoxVolume.rc

```tcl
module -name "Bounding Box Volume" \
-primary "HxUniformScalarField3" \
-package "py_core" \
-category "{Measure And Analyze}" \
-proc {
    set this [[create HxPythonScriptObject] \
                setLabel "Bounding Box Volume"]
    "$this" startStop hideMaskIncrease
    "$this" filename hideMaskIncrease
    "$this" filename setValue \
            <PRODUCT_PATH>/share/python_script_objects \
            /PythonBoundingBoxVolume.pyscro
    "$this" startStop hit 0
    "$this" fire
    if { [exists $PRIMARY] } {
        $this data connect $PRIMARY
        $this fire
    }
}
```

9.5.1.5.2 Resource File

A resource file is a TCL script that is read and executed during the Avizo startup process. You can configure a resource file, so that your Python script object appears in the pulldown menu or as a macro button. The resource files are located in the $AVIZO_ROOT/share/resources directory.

Structure Pulldown Resource File

The script begins with the TCL command `<module>` followed by the following flags for a simple case. In the example below, $PRIMARY refers to the object that you originally right-clicked.

`-name`
This is the name of your module in the menu.

`-package`
This defines the package of your object.

`-primary`
This restricts the data type that is required for your script to appear.

`-category`
This defines which folder in the menu that your script will appear.

`-proc`
This is the body of the resource file where you can identify while *.pyscro* loads along with connecting your script to the object that you originally right-clicked.

Structure Macro Resource File

The script begins with the TCL command `<macroButton>` followed by the following flags for a simple case.

`-add`
This creates a name for your button.

`-color`
This controls the color of your button.

`-proc`
This is the body of the resource file where you can identify while *.pyscro* loads or the procedure runs.

9.5.1.6   Python Package Manager: Enstaller

9.5.1.6.1   The enpkg Tool
Context

This section explains how to list/install/update a new python package for Avizo in command line. The full description of Enthought enstaller's documentation can be found at http://enstaller.readthedocs.io/en/4.8.12/

Description

The Package manager command line `enpkg` can be used to install, remove, or upgrade Python packages available in three repositories:

- official fei/avizo
- enthought/free
- enthought/commercial

It provides several options for finding and installing available packages; for viewing, updating, and removing installed packages; and for reverting to previous states.

Prerequisites

To use the Package manager command line `enpkg`, it is necessary to have the root/admin read/write/execute permissions.

Avizo Package Manager Configuration File

The `$AVIZO_ROOT`/python/.enstaller4rc file contains reference to the three available repositories (`IndexedRepos`). Nonetheless, access to the Enthought free and Enthought commercial has been deactivated. You can grant access to those two additional repositories by uncommenting the following two lines:

```
'https://packages.enthought.com/repo/enthought/free/{PLATFORM}/',
'https://packages.enthought.com/repo/enthought/commercial/{PLATFORM}/',
```

By doing so, you will have access to merely the entire python ecosystem, but notice that only packages from the FEI repository have been tested as being fully compatible with Avizo. You may experience strange or incorrect behavior when using a package from another repository rather than the one from FEI.

9.5.1.6.2   The Command Line enpkg
Platform Directory of `enpkg`

On Windows, the `enpkg` tool is available in the `Scripts` relative path, but on the Linux/MacX platform, it is available in the `bin` relative path. The following examples are based on the Windows platform:

How to Use `enpkg`

For example, to know the version of `enpkg` in the $AVIZO_ROOT/python directory, open a bash or window console and execute the following command line:

```
./Scripts/enpkg --version
```

The `--help` option (or `-h`) shows all the available options:

```
./Scripts/enpkg -h
```

How to List the Current Installed Packages

To list all the current installed packages in Avizo, type:

```
./Scripts/enpkg -l
```

How to Search the Available Packages

The list of available packages will be displayed in your Avizo Package manager Configuration file (i.e., only for the FEI/Avizo repository or all available repositories). The following command line can be used:

```
./Scripts/enpkg -s
```

How to Search a Specific Package

To search the specific package version available of `numpy`, type:

```
./Scripts/enpkg -s numpy
```

How to Update/Install an Installed Package

To update with the last available version of an installed package `<package_name>` in Avizo use the following command:

```
./Scripts/enpkg <package_name>
```

And if you want a specific `<version>`:

```
./Scripts/enpkg <package_name> <version>
```

How to Reinitialize the Avizo Packages

To reinitialize Avizo packages to their original state (`<original_version>`), type:

```
./Scripts/enpkg avizo <original_version>
```

This may be useful if the FEI python distribution becomes non functional (for instance after installing unsupported packages).

9.5.1.7   List of Python Packages

List of packages already included in FEI Python:
enstaller 4.8.11-1
alabaster 0.7.7-1
babel 2.2.0-1
backports_abc 0.4-1
certifi 2015.11.20.1-1
configobj 5.0.6-1
cycler 0.9.0-2
cython 0.23.4-1
decorator 4.0.6-1
distribute_remove 1.0.0-3
docutils 0.12-1
freetype 2.5.3-4
h5py 2.5.0-4
hdf5 1.8.15.1-1
jdcal 1.2-1
jinja2 2.8-1
libjpeg 7.0-3
libopenjpeg 2.1.0-2
libpng 1.6.12-3
libxml2 2.9.2-2
libxslt 1.1.28-3
lxml 3.5.0-1
markupsafe 0.23-1
matplotlib 1.5.1-1
mkl 10.3-2
networkx 1.10-3
numexpr 2.4.0-3
numpy 1.9.2-2
numpydoc 0.5-7
openpyxl 2.3.1-3
pandas 0.17.1-7
patsy 0.4.1-1
pil_remove 1.0.0-2
pillow 3.1.0-1
pydicom 0.9.9-1
pygments 2.1-1
pyparsing 2.0.3-1
pytables 3.2.2-2
python_dateutil 2.4.2-2
pytz 2015.7-1

scikit_learn 0.17-1
scikits.image 0.11.3-12
scipy 0.16.1-1
seaborn 0.6.0-6
setuptools 19.4-1
singledispatch 3.4.0.3-1
six 1.10.0-1
snowballstemmer 1.2.1-1
sphinx 1.3.5-1
sphinx_rtd_theme 0.1.9-1
ssl_match_hostname 3.4.0.2-1
statsmodels 0.6.1-13
tornado 4.3-1
xlwt 1.0.0-1

9.5.1.8   Python Limitations

Context
This section explains the limitations that applies to Python integration in Avizo.

9.5.2   Future statement definitions

In Avizo embedded Python console, behavior of future statement definitions is sligthly different from the original python console. When typing in a regular Python console:

```
>>> from __future__ import division
>>> print 1 / 2
0.5
```

The division is automatically performed in floating point numbers because of the `import` statement. But in the Avizo Python console, the same code snippet will generate the following:

```
>>> from __future__ import division
>>> print 1 / 2
0
```

This is due to the fact that every line typed in the Python console is evaluated as a whole entity. (Regular Python console uses stream evaluation, which is not compatible with an embedded environment). Consequently, in order to be able to use the `__future__` statement in Avizo Python console, you must press SHIFT + Enter after typing your import statement to force the evaluation of the import statement with the following line, resulting in the following:

```
>>> from __future__ import division     # Press SHIFT + Enter
... print 1 / 2
0.5
```

Note
This limitation does not apply to an *.APL file (Avizo Python Listing) or to a Python Script Object :
the classic Python behavior prevail.

9.5.3   Python Tutorials

9.5.3.1   Python Tutorial - Using Tools from the Python Eco-System within Avizo

One of the advantages of integrating Python into Avizo is that Python tools can now be used to extend
Avizo functionality. This extension allows writing script objects that encapsulate Python functions to
make them available as modules in the Avizo graphical user interface. These Python tools can then be
controlled through standard Avizo ports. This tutorial demonstrates how to integrate the Fast Fourier
Transform (FFT) from scipy as an alternative to the FFT used in Avizo.

Copy *PythonScriptObjectTemplate.pyscro* from the $AVIZO_ROOT/share/python_script_objects direc-
tory to a location of your choice and rename it to *ScipyFFT.pyscro*.

1. Open the file in a text editor and give your new module a name by changing the first line to:

   ```
   class ScipyFFT(PyScriptObject):
   ```

2. In the initialization function, the default data input port will be used for the data connection to
   your module and the allowed connection types will be defined:

   ```
   self.data.valid_types = ['HxUniformScalarField3']
   ```

3. The final initialization function definition will look like this:

   ```
   def __init__(self):
       self.data.valid_types = ['HxUniformScalarField3']

       # Create an 'Apply' button.
       self.do_it = HxPortDoIt(self, 'apply', 'Apply')
   ```

4. Leave the update function as is:

   ```
   def update(self):
       pass
   ```

Python Scripting   **525**

5. The computation of the FFT will be done in the compute function. After checking if **Apply** was clicked and if an input dataset was selected, import a few packages from Python. In this case, we need the fftpack from scipy, numpy for some mathematical functions, and the time package to measure execution time:

```
from scipy import fftpack
import numpy
import time
```

6. As a first step in the computation, create a variable to store the result from the FFT computation as a 3D uniform scalar field.

```
result = hx_project.create('HxUniformScalarField3')
```

7. To measure execution time, you first need to take a time stamp at the beginning of the computation:

```
start_time = time.time()
```

8. Create a Python variable for the input data:

```
input = self.data.source()
```

9. To compute the absolute value of the FFT of our input data, execute the following three commands:

```
# Compute the discrete Fourier transform
F1 = fftpack.fftn(input.get_array())

# Shift the zero-frequency component to the center of the spectrum
F2 = fftpack.fftshift(F1)

# Take the magnitude of all coefficient
F3 = numpy.abs(F2)
```

10. After the computation has finished, assign the result array to the result variable that you created earlier:

```
result.set_array(F3)
```

11. Print out the total execution time of the FFT:

```
        print("--- %s seconds ---" % (time.time() - start_time))
```

12. The entire compute function should look as follows:

```
def compute(self):
    # Check if module's apply button has been touched by the user
    if not self.do_it.was_hit:
        return

    # Check if input data is connected to a valid object
    if self.data.source() is None:
        return

    # Import scipy package to perform fft
    from scipy import fftpack
    import numpy
    import time

    # Create the output field
    result = hx_project.create('HxUniformScalarField3')

    start_time = time.time()

    # Retrieve the input data
    input = self.data.source()

    # Compute the discrete Fourier transform
    F1 = fftpack.fftn(input.get_array())

    # Shift the zero-frequency component
    # to the center of the spectrum
    F2 = fftpack.fftshift(F1)

    # Take the magnitude of all coefficient
    F3 = numpy.abs(F2)

    # Affect to the output scalar field the resulting numpy array
    result.set_array(F3)

    # Show in console computation time
    print("--- %s seconds ---" % (time.time() - start_time))
```

13. To make this module available in the graphical user interface of Avizo, you will also need to write a resource file. Create a new file with your text editor and save it as *ScipyFFT.rc*. Resource files typically start with a comment line:

```
###########################################################
# .rc for pyscro Scipy FFT
###########################################################
```

14. Name the module:

```
module -name "Scipy FFT" \
```

15. Specify the data type we would like to be able to attach it to:

```
-primary "HxUniformScalarField3" \
```

16. Declare it as a Python script object:

```
-package "py_core" \
```

17. The next line defines the location where it will appear in Avizo's *Object Popup* menu:

```
-category "{Python Scripts}" \
```

18. Run a few TCL commands to initialize the module in Avizo's graphical user interface. The first command will create the script object and set a label for the module:

```
-proc {
    set this [[create HxPythonScriptObject] setLabel "Python FFT"]
```

19. Set the file name to find the Python script location for this module, where the <PRODUCT_PATH> is $AVIZO_ROOT:

```
"$this" startStop hideMaskIncrease
"$this" filename hideMaskIncrease
"$this" filename setValue \
<PRODUCT_PATH>/share/python_script_objects/ScipyFFT.pyscro
```

20. The script will run for the changes to take effect:

```
"$this" startStop hit 0
"$this" fire
```

21. Connect the dataset to the default input data port on which you right-clicked to create the module:

```
if { [exists $PRIMARY] } {
    $this data connect $PRIMARY
    $this fire
    }
}
```

22. The entire resource file will look like this at the end, where the <PRODUCT_PATH> is $AVIZO_ROOT:

```
##############################################################
# .rc for pyscro ScipyFFT
##############################################################

module -name "Scipy FFT" \
    -primary "HxUniformScalarField3" \
    -package "py_core" \
    -category "{Python Scripts}" \
    -proc {
        set this [[create HxPythonScriptObject] \
        setLabel "Python FFT"]
        "$this" startStop hideMaskIncrease
        "$this" filename hideMaskIncrease
        "$this" filename setValue \
        <PRODUCT_PATH>/share/python_script_objects/ScipyFFT.pyscro
        "$this" startStop hit 0
        "$this" fire
        if { [exists $PRIMARY] } {
            $this data connect $PRIMARY
            $this fire
        }
    }
```

23. To make this Python script object available to Avizo, you need to copy both files to $AVIZO_ROOT/share/python_script_objects/ and restart Avizo after changing the resource file.

24. To test this module, do the following:

   1. Start Avizo.
   2. Load $AVIZO_ROOT/data/tutorials/chocolate-bar.am

3. Right-click on the data object and select **Python Scripts/Scipy FFT** from the *Object popup*.

4. Click **Apply** and a new data object with the resulting FFT will display in the Project View.

9.6 Using MATLAB with Avizo

This section describes how to use *MATLAB Scripts* in Avizo.

9.6.1 Using MATLAB Scripts

In this tutorial, you will learn how to integrate complex calculus into Avizo using MATLAB (The MathWorks, Inc.) by the means of the *Calculus MATLAB* module.

To use the Calculus MATLAB module, MATLAB must be correctly installed on your computer. In addition, to allow this module to establish a connection with the MATLAB computational engine, you may need to register the MATLAB engine on Windows and set environmental variables (i.e., for including MATLAB libraries or programs in search paths) depending on your system. See the documentation for the *Calculus MATLAB* module for installation details and limitations.

This tutorial, available in the online documentation, covers the following topics through various examples:

- Loading and executing a MATLAB script.
- Passing various data types from Avizo to MATLAB and exporting them back.
- Using the *field* structures.
- Controlling the script variables with time sliders.
- Calling user MATLAB functions from a script.

Chapter 10

Avizo XFiber Extension User's Guide

Avizo XFiber Extension is an extension that provides specific support for analyzing fibers, filaments, tunnels, and other networks or tree-like structures. This option assists segmentation and analysis with automatic, semi-automatic, and interactive tools.

The extension provides tools for tracing centerlines in various modality acquisitions, such as X-ray microtomography, to detect fibers or tube-like structures.

The Avizo XFiber Extension includes many analysis tools to compute advanced fiber statistics. Various distributions and properties can be calculated and plotted. Various tools also allow for fibers to be filtered based on properties and attributes.

The Avizo XFiber Extension also includes the *Filament Editor* workroom that offers automatic and interactive tracing tools for segmenting the centerline of the fibers and measuring their thickness in fiber networks and other filamentous structures. This workroom also offers editing of spatial graphs obtained by skeletonization.

- *Introduction to the Filament Editor*
- *The detection workflow for the example of steel fibers in reinforced concrete*
- *Computing advanced statistics*

10.1   Introduction to the Filament Editor

This section provides a step-by-step introduction to the *Filament Editor*. To use the *Filament Editor*, an Avizo XFiber Extension license is required. The *Filament Editor* is a special purpose workroom in Avizo that is designed to extract complex three-dimensional networks of filamentous structures from image data and post-process data by editing and annotating the network with label scalar data. In this tutorial, we want to demonstrate the principles of the Editor by extracting the dendritic fiber network of

an invertebrate neuron. The dataset we will be working with depicts a neuron from the honeybee brain imaged with a confocal microscope and was kindly provided by Prof. R. Menzel, Free University of Berlin.

The following topics will be discussed in this section:

- Exploring the volume data
- Automatic extraction of the network
- Interactive tracing
- Labeling and visualizing the network

To access the *Filament Editor*, click the icon in the *Workrooms Toolbar*. We will begin our work by loading the 3D image dataset.

1. Load the file `neuron.am` located in the `data/tutorials/neuron` subdirectory

10.1.1   Exploration of the Volume Data

Once the image data has been loaded, the left-hand panel of the viewer shows a 2D slice of the volume, while the right-hand panel of the viewer shows the 3D objects. To have a clear idea of how the neuron spreads out in space, we will explore the volume through slicing, windowing, and rendering of the gray values.

Window Level:

1. Select the **Window Level** icon.
2. Click the 2D viewer.
3. Click the mouse button and hold.
4. Drag the mouse cursor left/right to change the window width, or up/down to change the window center.
5. Set the window level around **30−80**.
6. Click **3D** to visualize the selected voxels using a shaded volume rendering.

Browse Slice:
The browsing tool allows using the mouse to navigate through an image stack in the 2D viewer (or MPR viewer).

1. Select **Browse Slice**.
2. Click the 2D viewer.
3. Click the mouse button and hold.
4. Drag the mouse cursor up/down to scroll forward or backward through the image stack. If you are working with a wheel mouse you can use the wheel instead of **Browse Slice**.

5. Click the viewer and scroll the mouse wheel to scroll through the image stack.

Thick Slice:

The thickness of the slice can be set by sliding the *Thickness* slider. When displaying a thick slice, data values are computed as maximum intensity of the values of the original slices.

1. Set thickness to **25**.

2. Click **3D Thickslice** to visualize the thick slice in the 3D viewer.

Figure 10.1: The Filament Editor Immediately After the Image Data has been Loaded

10.1.2 Automatic Extraction of the Dendritic Tree

In a first step, you may want to automatically extract what will serve as a draft version of the neuron's skeleton. This can be achieved with the *Filament Editor*'s *Auto Skeleton* tool.

1. Set the *Window Level* to **37 137**.
2. Click **3D** beside the *Window Level* slider. The 3D viewer shows a shaded volume rendering of the neuron's main branches and gives a rough estimate of what the automatic tracing procedure will consider as part of a neuron.
3. Select the *Trace* tab in the **Tool Box** and click **Run** of the Auto Skeleton frame. After a few seconds you will see green lines and blue points in the 2D viewer. In the 3D viewer, gray spheres are displayed together with the preview rendering. If you do not see them, click **View All** or the space bar to center the 3D viewer on them.
4. Adjust the **Alpha** slider, which appeared when you activated the 3D rendering. You can see the generated graph through the rendering.

The *Auto Skeleton* tool traces connected regions according to a user-defined window level and converts the centerline of those regions into graphs composed of points, segments, and nodes that are the elements of the data class *Spatial Graph*. The result of the *skeletonization* is, therefore, a *Spatial Graph* object that is being visualized in the 3D viewer as balls and lines as well as blue points connected by green lines in the 2D viewer.

Depending on the quality of the data and on the selected window level, the result of skeletonization will typically show several disconnected elements and loops within the networks. Disconnected elements will break the single neuron into several subgraphs. Loops, on the other hand, are parts of a graph where segments connect onto itself. In some biological objects (e.g., neurons loops) this must not occur. To identify graphs and loops, the *Filament Editor* offers the following dedicated tools:

1. Click **Identify Graphs** in the *Edit* panel to automatically detect and label all graphs (i.e., all connected components) in the Spatial Graph object. This creates a new label group in the *Label Editor* named *Identified_Graphs* under which all identified graphs are listed as *Graph0, Graph1, ..., GraphN*.

2. To visually identify them, click on one of the items in the *Identified_Graphs* list. The selected item is highlighted with red in both viewers.

3. Under *Identified_Graphs*, select *Graph2* and press **SHIFT** while selecting *Graph25*, then remove them by clicking **Delete selected nodes, edges and points** in the toolbar.

4. Click **Identify Graphs** again. Now, you should have only two graphs.

5. Scale the nodes by moving the *Node Scaling Factor* slider in the *View* tab of the **Tool Box** (see details below).

At this point, we have only two graphs: Graph0 is the large tree, while Graph1 is just three segments and four nodes. Switch back to the *Edit* tab in the **Tool Box**.

1. Select **Graph1** in the Label Editor under *Identified_Graphs*.

2. Activate **Select Single** and press **CTRL** while clicking on the closest node of Graph0 in the 3D viewer.

3. Select **Connect Selected** to connect the graph.

4. Click **Identify Graphs** again. At this point, you should have only one graph.

5. Click **Identify Loops** to get another label group *Identified_Loops*.

6. Under *Identified_Loops*, select **Loop2** and remove it by clicking **Delete selected nodes, edges and points**. Note that a segment is defined by two nodes. Therefore, when you remove a node the associated segment will also be removed. Hence the *Identified_Loops* contains the looping segments, but not the nodes connecting them. However, this may produce isolated points that can be removed by **Delete selected nodes, edges and points**.

It should perhaps be noted that the *Auto Skeleton* tool is also available as a compute module in the *Object Popup* by selecting *Image Processing / Skeletonization / Auto Skeleton*. Also, there is a rich set of tools available in the *Segmentation Editor* to perform the threshold segmentation necessary for the

skeletonization process. Finally, in the case of a strict tree topology it may be advantageous to restrict the search to a tree. In this use case, you may want to use the *Centerline Tree* module to extract a graph with guaranteed tree topology.

Before closing this subsection, we should save the *Spatial Graph* data object we have created, it will be useful in further steps of this tutorial.

1. Switch back to the Project View, save the object called *neuron.Smt.SptGraph* in a directory of your choice, and then remove it from the Project Graph View.
2. Switch again to the *Filament Editor* to prepare the next step.
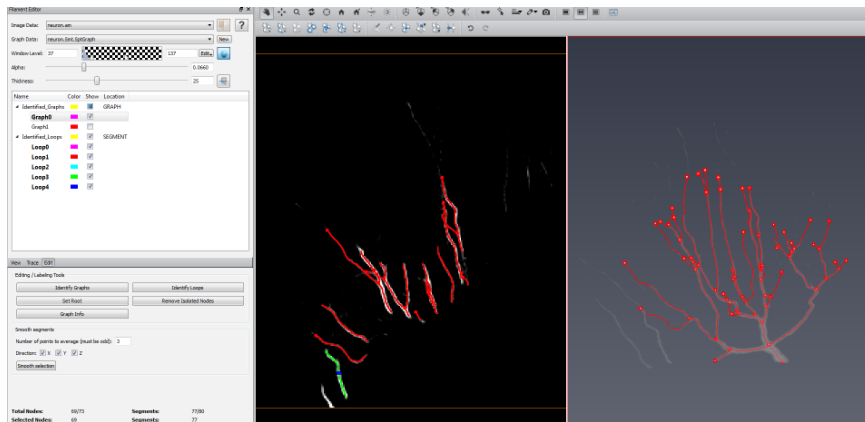
Figure 10.2: The Spatial Graph Object obtained with the *Auto Skeleton* Tool.

10.1.3 Filament Tracing

The *Interactive Tracer* tool enables you to trace the filaments directly on the gray values using an innovative tracing algorithm developed by us. The user sets the starting and ending points of a segment, and the *Interactive Tracer* finds the shortest line connecting the two points with respect to the user-defined data window. Now, we can use the *Interactive Tracer* to retrace the neuron segments.

To clearly understand how to use the tracer, we will try to retrace a small part of the neuron tree we extracted in the previous paragraph.

1. Click **New** at the *Graph Data* port to create a new *Spatial Graph* data object.
2. Click in the 2D viewer. Using the *Browse Slices* tool or the mouse wheel, slice through the volume until you reach the root of the neuron.
3. Set the thickness of the *Thick Slice* to 10 for a better visualization.

4. Activate the *Interactive Tracer* by clicking **Trace Filament** in the toolbar and make sure that the **Thick structures** option is selected in the *Trace* tab.

Note that the *Interactive Tracer* is always active while **Trace Filament** is highlighted and it can be triggered only if the 2D viewer is active.

If you now move the mouse over the 2D slice, the cursor will have a cross shape whenever it is over black regions and turns into a crosshair (i.e., cross within circle) when it is over voxels that are within the window level. The crosshair shape indicates that it is possible to set tracing key points. As you will see later, when the pointer is over a traced segment (e.g., green line) a small **P** appears in the crosshair. This indicates that a click would select the nearest point as a key point. Likewise, an **N** is shown in the cursor whenever the cursor is over a node and a click will select this node as a key point. Furthermore, the cursor turns red when the trace tool is in *Append* mode, meaning that the next click will trigger a tracing action between the previously selected and the current key point.

1. Start the tracing by clicking on a gray value of the tree root.
2. Slice ahead and set further points until you reach the first bifurcation, then click on it.
3. Deactivate the *Interactive Tracer* (e.g., by clicking the *Select Single* tool or by clicking again its icon).

If you click **Identify Graphs**, you will see only one graph. You can also access some statistical information by clicking **Graph Info** in the *Edit* tab. This opens a spreadsheet window containing a list of all segments in the graph. The spreadsheet is interactive (i.e., if you click on a line, the corresponding segment will be highlighted in the viewer).

Repeat the actions to trace the first branching sections from the root.

10.1.4 Visualize the Network

The *Spatial Graph* data object is able not only to store the spatial structure (i.e., the geometry of the network), but also to associate it with scalar and label data.

Labels can also be used to tag or annotate sub-graphs according to their topology. In this example, we want to tag the three main branches of the *Topology* as *Central*, *Left*, and *Right* of a neuron tree.

If you already extracted the graph as described in the second subsection, you should now load the file you previously saved, otherwise you should go two steps back and read the subsection *Automatic Extraction of the Dendritic Tree*.

1. Right-click **Label Editor** and select **Add graph label group / Empty label group**.
2. Right-click on the created label and rename it *Topology*.
3. Right-click on the *Topology* label and select **Add label**. Repeat this action to add three labels.
4. Select and right-click on each of the created labels and rename them *Central*, *Right*, and *Left*.
5. Select **Draw a line to select nodes, edges and points** and select-lasso the right-hand branch.

You can use the *Draw a line to select nodes, edges and points* tool in combination with the **Alt + Right-click** to deselect-lasso or with **CTRL** to add elements.

6. Right-click on label **Right** and select **Assign selection**.

7. Repeat the last two items to label *Central* and *Left* branches. Note that this is just an exercise, and it does not matter if you are exact when selecting the *Center*, *Right*, or *Left* segments. Now, select the **View** tab in the **Tool Box**.

8. Scale the nodes using the **Scaling Factor** slider.

9. Colorize the nodes according to the label by selecting **Topology** in *Node Coloring*.

10. Repeat the last item for the segments.

11. Click the color button of *Right* label and select a new color.

For an advanced network visualization, you can switch to the Project View. If you do so, you may attach a *Spatial Graph View* module to the *Spatial Graph* object you previously created.
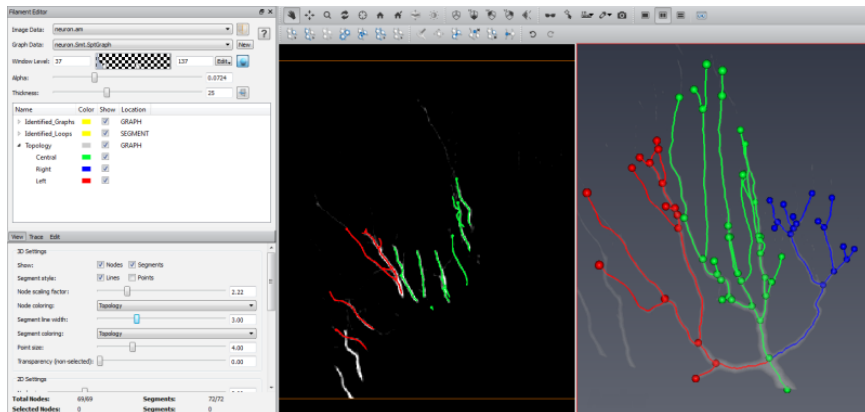
Figure 10.3: The Spatial Graph Obtained with the Auto Skeleton and Tracing Tools

10.1.5   Alternative Way to Extract a Network Using the Segmentation Editor

From the *Filament Editor*, switch to the *Segmentation Editor* by clicking **Segmentation Editor** in the *Workrooms Toolbar*. For an in-depth treatment of the *Segmentation Editor*, refer to the associated documentation and tutorial. For the tutorial try the following steps:

1. If the label *neuron.labels* is not available, create this new label image by clicking **New** in the *Label field* port located in the upper section of the user interface.

2. Create a new material by clicking **Add** under the *Material List* in the *Materials* port.

3. Right-click the new material in the *Material List*, select *Rename Material*, and enter **neuron**.

4. In the *Tools* tool box at the bottom section of the user interface, select **Threshold Tool**.

5. In the *Threshold Tool* area, set the threshold to **100−255**, select the **All Slices** option and click **Select Masked Voxels**.

6. In the selection, click **+** to assign the selected region to *neuron* material.

7. Switch back to the *Filament Editor* and select *neuron.labels* from the *Image Data* pulldown menu.

8. Skeletonize the label image by clicking **Run** in the *Auto Skeleton* tool.

10.2   Detecting Fibers or Tube-Like Structures

This tutorial requires the Avizo XFiber Extension, that provides specific support for tracing the centerline of fibers, filaments, or other tunnels that have a relatively fixed and constant diameter. The method allows for robust and accurate centerline tracing of individual fibers even in cases of noisy data and/or densely packed fibers, as long as the fibers are resolved, with at least a few voxels for the diameter.

This tutorial explains how to extract centerlines of tube-like structures from various modality acquisitions. It is organized in three steps:

1. *How to compute the normalized cross correlation to enhance fiber structures.*
2. *How to trace centerlines to create a spatial graph of the separated fibers.*
3. *How to derive statistics from the spatial graph of fibers.*

The dataset used in this tutorial is an Ultra High-Performance Fibre Reinforced Concrete (UH-PFRC), courtesy provided by Dr. Alenka Mauko and Dr. Aljosa Sajna from the Slovenian National Building and Civil Engineering Institute (www.zag.si; aljosa.sajna@zag.si), Log Cezsoski Bridge, Slovenia, WP7 project Assessment and Rehabilitation of Central European Highway Structures (ARCHES) http://arches.fehrl.org/ (see Figure 10.4).

The tutorial will be applied on a cropped volume of this dataset. To apply the workflow to your data, you might need to adapt these parameters as explained in more detail below.

The tutorial assumes that you are familiar with the basic concepts of Avizo. In particular, you should know how to load files, how to interact with the viewer, and how to connect modules to data objects. All these topics are covered in Chapter 2 - Getting Started of the Avizo User's Guide. Note that the modules presented in this tutorial require an NVIDIA graphics card supporting CUDA Compute Capability 1.3 or higher. To find your graphics card CUDA Compute Capability, see NVIDIA's list of CUDA GPUs (http://developer.nvidia.com/cuda-gpus/).

10.2.1   Computing Normalized Cross Correlation

The module *Cylinder Correlation* can be used to enhance fibers or tube-like structures in various modality acquisitions, such as X-ray micro tomography as illustrated in this tutorial. The X-ray micro
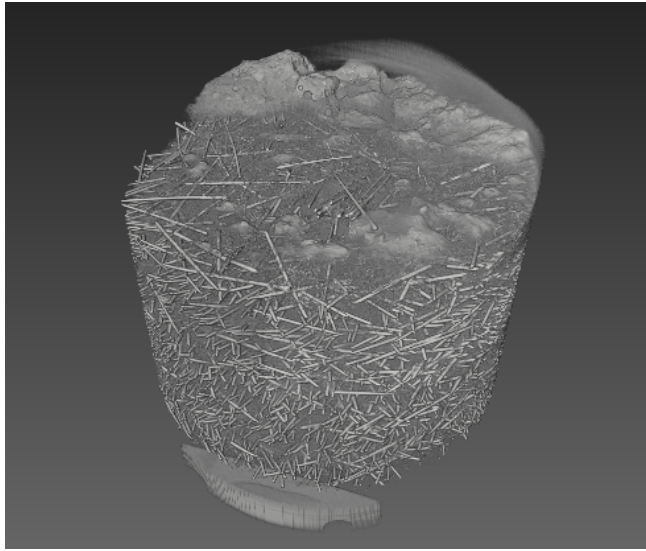
Figure 10.4: Volume Rendering of the Ultra High-Performance Fibre Reinforced Concrete

tomography is the first step before tracing the centerlines of these structures. The module computes the cross correlation of an image with a hollow or solid cylinder template in all 3D orientations. To apply the module to an example dataset, proceed as follows:

1. Load `AVIZO_ROOT/data/tutorials/fibertracing/concrete.am` into the Avizo workspace. A green data icon appears in the *Project View*.
2. Attach a *Cylinder Correlation* module (in category *Fiber Tracing*) to the data.
3. Set the parameters as shown in Figure 10.5.

Once all parameters have been configured, click **Apply**. For the example project, the computation of the correlation field takes several minutes. Note that this time consumption is not representative of what it would be for the whole dataset since the time consumption is not a linear function of the dataset size. The algorithm requires you to internally add a fixed size border to the dataset; which has an effect on time consumption for small datasets, but has a negligible effect on large datasets. Depending on the graphics card and the size of the input image data, the computation of the full image data might take up to several hours. For example, on a GeForce 8800 GT with 1GB video memory the computation of an image of size 2000 x 2000 x 200 takes approximately 20 hrs. For testing the algorithm, consider cropping the input image data using the *Crop Editor*.

As a result, two new objects, *concrete.CorrelationField* and *concrete.OrientationField*, will be connected to the *Cylinder Correlation* module. The *concrete.CorrelationField* data object stores the maximum cross correlation, and the *concretes.OrientationField* stores the orientation of the cylinder cor-
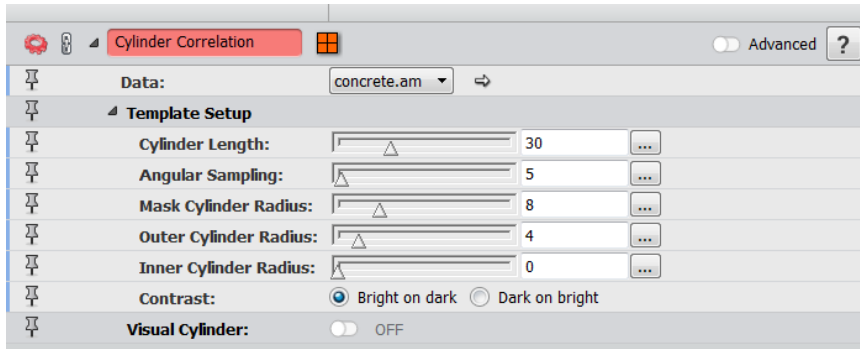
Figure 10.5: Default Parameter Settings of the *Cylinder Correlation* Module

responding to this correlation.

These two data objects also store information about the cylinder template that was used to compute the correlation in their respective data parameters. This information will then be used to correctly preset parameters of the *Trace Correlation Lines* module.

From these two fields, tube-like structures can be extracted as described in the next section.

But first, the parameters of *Cylinder Correlation* are explained in more detail.

10.2.1.1   GPU Computation

The cross correlation is computed on the GPU. Because the GPU memory is limited, the port *CUDA memory* allows you to limit the amount of GPU memory used for the computation. The preset value is estimated automatically depending on the memory available on the chosen *CUDA Device*. If you run other processes that consume GPU memory, computing the cross correlation might fail due to lack of memory. In this use case, you should set a reasonable limit. In practice, 80% of the free memory should work.

Computing the cross correlation is time consuming. To minimize computation time, consider cropping the image data to remove empty space. However, do not crop them too tightly (see section 10.2.1.3).

If you plan to process many datasets of similar characteristics, it is usually worth investing some time to determine the lowest reasonable resolution as a higher resolution does not substantially increase quality. The right voxel size is data dependent, but for solid cylindrical fibers as glass or carbon fibers, an apparent resolution with a fiber diameter of about 5 voxels should already yield good results.

10.2.1.2   Cylinder Template

The *Cylinder Correlation* module will try to correlate a cylinder template with the data in all orientations to mark each voxels with a score (i.e., maximum cross correlation). Before starting the

computation, this template needs to be well defined and to fit with a fiber structure of the data. This is done by specifying the length and the radius of the cylinder. A visual cylinder template is provided to easily set those parameters:

1. Enable the *Visual Cylinder* port (see Figure 10.6).
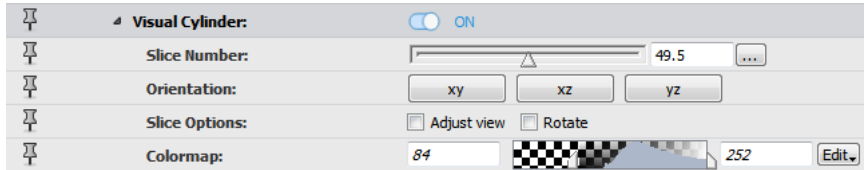
Figure 10.6: Visual Cylinder Port of *Cylinder Correlation* Module

2. A slice appears in the viewer to visualize the data. Translate and rotate the slice to best fit a fiber structure.
3. Once a fiber is visible on the slice along its main orientation, middle-click twice on the fiber to set the length of the cylinder with a start and end points. If the length is not satisfactory, repeat middle-clicking twice on the data (see Figure 10.7).
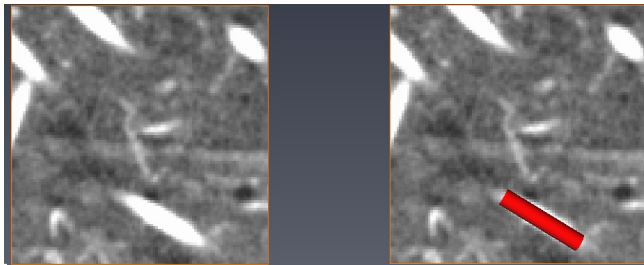
Figure 10.7: Visual Cylinder Length Setup

4. Set the radius of the cylinder template. The slice can now be changed to see the cross section of the fiber on which the visual template has been set. Left-click on the visual template and move the mouse to interactively change the cylinder radius until it fits the cross section on the slice (see Figure 10.8).

By using the visual cylinder, the *Outer Cylinder Radius*, the *Mask Cylinder Radius*, and the *Cylinder Length* ports are automatically set. Changing those ports while visualizing the template will interactively change the cylinder shape.

Alternatively, the values can be based on a priori information about the structures or on measurements taken from the image data. The measurement tool (i.e., *Measurement* in the viewer toolbar) is often useful to determine reasonable lengths and radii. Note that parameters are specified as radii; diameters

Detecting Fibers or Tube-Like Structures 541
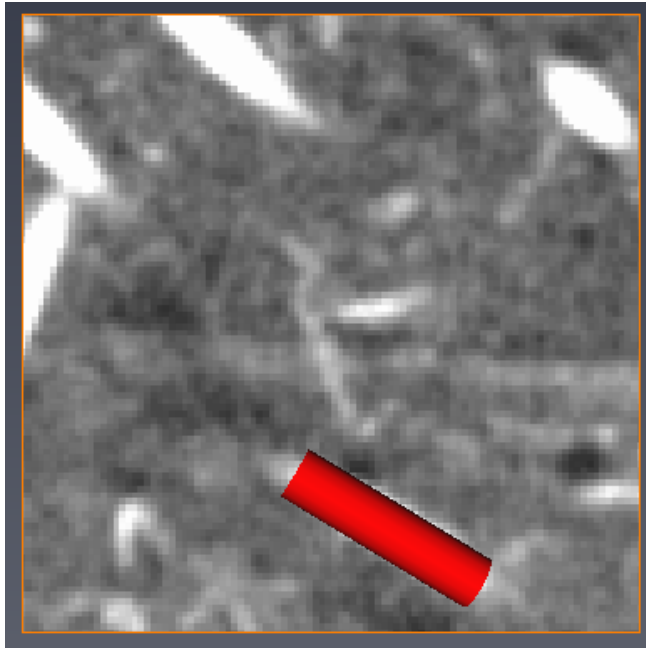
Figure 10.8: Visual Cylinder Radius Setup

need to be divided by 2. In the following sections, we use the concrete fibers as an example:

- Radius: For structures like steel fibers, a solid cylinder template is appropriate. Specify only the *Outer cylinder radius* and set the *Inner cylinder radius* to 0. Based on the measurements of the image data (see Figure 10.9), the fibers have a diameter of 8, so the *Outer cylinder radius* of 4 should be fine. Note that parameters are specified as radius, so diameters need to be divided by two. The *Mask cylinder radius* can be set to 8 for this example (i.e., diameter of 16). In the case of a hollow cylinder, a non-zero inner cylinder radius should be specified. The thickness of the cylinder wall is then equal to the *Outer cylinder radius - Inner cylinder radius*. Acquisition artifacts, like shrinking, might introduce a slight difference between the fiber real size and the size in the image.

- Length: The *Cylinder length* defines the template length in physical units. A longer template is more robust to noise, but it also reduces the sensitivity to curved structures and to the ends of structures. A longer template also requires you to stay further away from the image boundary (see section 10.2.1.3). A reasonable compromise depends on image quality and the structures to be extracted. In this example, a cylinder length of 30 is a good compromise. The tradeoff is different for other structures.

- Contrast: This specifies the contrast of the signal to be detected. Fibers, for example, are lighter
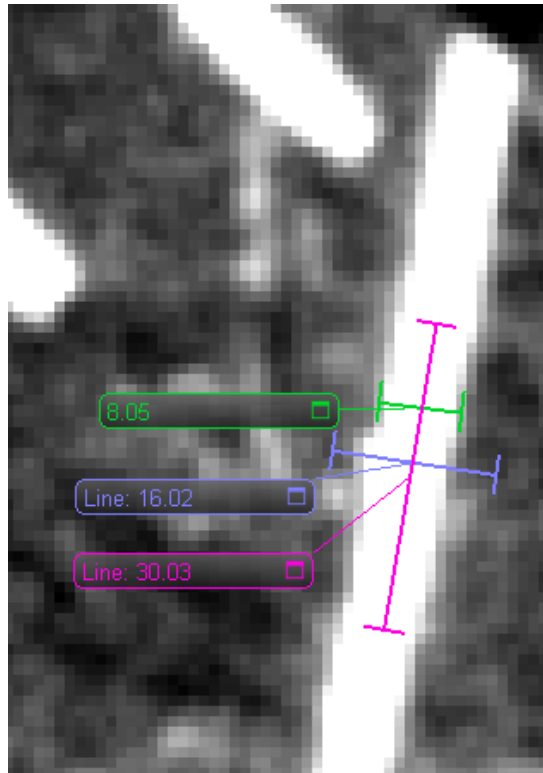
Figure 10.9: Length Measurements to determine *Cylinder Correlation* Parameters for Fibers. In magenta: *Cylinder length*, in green: twice the *Outer cylinder radius*, in blue: twice the *Mask cylinder radius*.

than the surrounding image, so you should select **Bright on dark**.

10.2.1.3 Limitations

Boundary: The correlation field may contain artifacts close to the boundary (i.e., within half the cylinder size). The reason is that the cross correlation is computed in Fourier space using the Discrete Fourier transformation (DFT) with periodic boundary conditions. The practical implication of this is that a template that is close to the boundary senses structures close to the opposing boundary of the image. Sharp edges in the image signal, like rapid foreground to background transition, may also create undesired effects. A general recommendation is:

- To acquire image data, so that the relevant structures are far enough away from the image boundary.
- To keep enough of the original image data when cropping, so that the relevant part of the corre-

lation field is away from the image boundary; yet, ideally have a reasonable image signal up to the boundary (i.e., no sharp transitions to black).

- Either crop the resulting correlation field to the section that is far enough away from the boundary and cut the orientation field to the same size before running *Trace Correlation Lines*, or clean up the resulting lines if they look unreasonable close to the boundary.

Template size: The template is sampled onto a cubic grid, which must fit into the image volume. For a large *Cylinder length*, the template may be too large for the image volume. In particular, if the image volume has few Z-slices, this might be a limitation. To successfully apply *Cylinder Correlation*, reduce the *Cylinder length* until the template fits. If this does not yield satisfactory results, consider increasing the volume by adding empty Z-slices using the *Crop Editor*. Keeping the template small is a good idea, in general, since the computation time increases with template size.

10.2.2 Tracing the Centerlines

The module *Trace Correlation Lines* traces centerlines of tube-like structures based on the *concrete.CorrelationField* and the *concrete.OrientationField*, which were computed in the previous step.

The following port will automatically be set based on the cylinder template parameters that were used to compute the correlation: min line distance (i.e., set to the cylinder diameter), min line length (i.e., set to the cylinder length), search cone (i.e., set to the cylinder length).

- Attach a *Trace Correlation Lines* module to *concrete.CorrelationField* (in category *Fiber Tracing*).
- Choose *concrete.OrientationField* as *Orientation field* input of *Trace Correlation Lines* module.
- Set parameters as shown in Figure 10.10.

Figure 10.10: Default Parameter Settings of the *Trace Correlation Lines* Module

Trace Correlation Lines traces lines that start at points whose correlation value is greater than the value of *Min seed correlation*. It is useful to inspect the correlation field to determine an initial value to refine the choice. If too few lines are extracted, consider lowering the value. If too many lines are extracted, consider increasing the value. The initial values should be chosen so that all points with a greater correlation value are clearly useful seed points. This can be done in various ways such as using an *Ortho Slice* (see Figure 10.11 and Figure 10.12 with same reasonable data range for *Min seed correlation* (max of range) and *Min continuation quality* (min of range)). Alternatively, use an *Isosurface* and adjust its *threshold* such that the isosurface contains only parts that are clearly useful as seed points. Then use this threshold as the *Min seed correlation*.

Figure 10.11: Histogram of Correlation Values
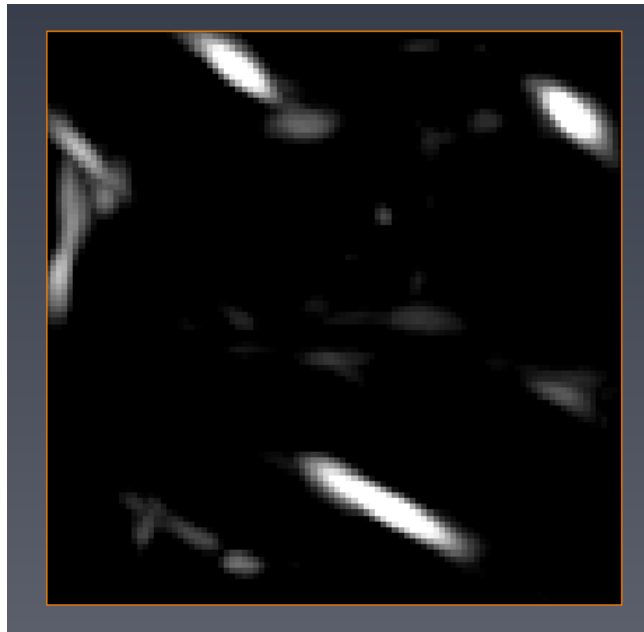
Figure 10.12: Slice through Correlation Values

It is not necessary that a given fiber looks continuous on the correlation score image, but a fiber cannot be traced if it has not at least 1 voxel above the *Min Seed Correlation* threshold.

The *Min continuation quality* can be determined in a similar way (see Figure 10.11). It controls where

line tracing stops. If lines are unexpectedly short, try a smaller value. If lines are too long, try a larger value.

The purpose of *Min Line Distance* is to avoid that additional lines are traced inside the diameter of other lines, which might happen if correlation values are not clearly separated. To determine a reasonable distance value, use the measurement tools (i.e., *Measurement* in the viewer toolbar).

Another data-specific parameter is the *Direction coefficient*. If the tube-like structures are more or less straight, a smaller value is appropriate. If the data also contain curved lines, a larger value might be appropriate.

You might also have to adjust all other parameters that are in physical units to the right scale for the relevant structures in your data. See the module's *Documentation* for details.

Once all parameters have been configured, click **Apply**. For the example network, the extraction of the centerlines takes only a few seconds. For a larger dataset, you should expect the computation to take several minutes.

The result of the module *Trace Correlation Lines* is of type *Spatial Graph*. To display this object, right click on the result's icon in the *Project View* and choose *Spatial Graph View* (i.e., in category *Display*) from the context menu. For further information on the data structure, click on the icon in the *Project View* and then click **Help** in the *Properties Area*.

10.2.3 Computing Basic Statistics for Tracing Results

Basic statistics are available from the *Spatial Graph* computed by the *Trace Correlation Lines* module. The module internally applies a *Spatial Graph Statistics* module to compute statistics of each centerline including:

- Chord Length: straight distance between start and end point of the segmentation
- Curved Length: curved segment length
- Tortuosity: Curved Length / Chord Length
- Orientation (Theta and Phi) of the segment between the start and end points

Those statistics are stored as attributes in the spatial graph. Those attributes can be displayed in the table view, with the *Show* button of the *Spatial Graph* module (i.e., *Table* port). The graph can also be displayed by using the *Spatial Graph View* module (see Figure 10.13).

It is also possible to use a *Spatial Graph Statistics* module on this attributed graph to output the statistics in a regular spreadsheet (see Figure 10.14).

1. Attach a *Spatial Graph Statistics* module to the generated SpatialGraph. To do so, right-click the green data object **CorrelationLines**. From the context menu, select **Measure And Analyze**. Then click **SpatialGraphStatistics**.
2. In the *Properties Area* of the *SpatialGraphStatistics*, click **Apply**. A *concrete.statistics* spreadsheet, as shown in Figure 10.14, is generated and displayed in the *Project View*.

3. Select the *concrete.statistics* spreadsheet and then click **Show**. A new dialog box will be displayed that presents the results in a table.
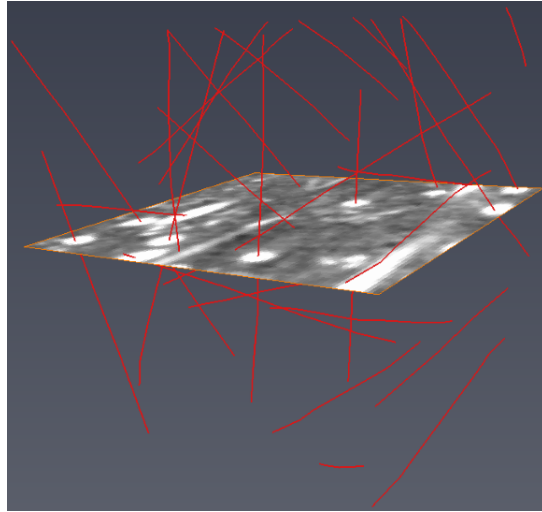
Figure 10.13: Display Centerline Orientations Using the *Spatial Graph View* Module

You can also load the project concrete_tutorial.hx to complete these tutorial steps. The result of *Cylinder Correlation* and *Trace Correlation Lines* applied on the whole example dataset, is shown in Figure 10.15.

10.2.4   References

1 Weber, B., Greenan, G., Prohaska, S., Baum, D., Hege, H.-C., Mueller-Reichert, T., Hyman, A. A., and Verbavatz, J.-M. (2012). Automated tracing of microtubules in electron tomograms of plastic embedded samples of caenorhabditis elegans embryos. Journal of Structural Biology, 178(2):129-138.

2 Rigort, A., Guenther, D., Hegerl, R., Baum, D., Weber, B., Prohaska, S., Medalia, O., Baumeister, W., and Hege, H.-C. (2012). Automated segmentation of electron tomograms for a quantitative description of actin filament networks. *Journal of Structural Biology*, 177:135-144.

10.3   Computing Advanced Statistics

This tutorial requires the Avizo XFiber Extension, which provides specific support for analyzing fibers, filaments, tunnels, and other networks or tree-like structures. This tutorial does not explain how to

Figure 10.14: Output Spreadsheet of the *Spatial Graph Statistics* Module
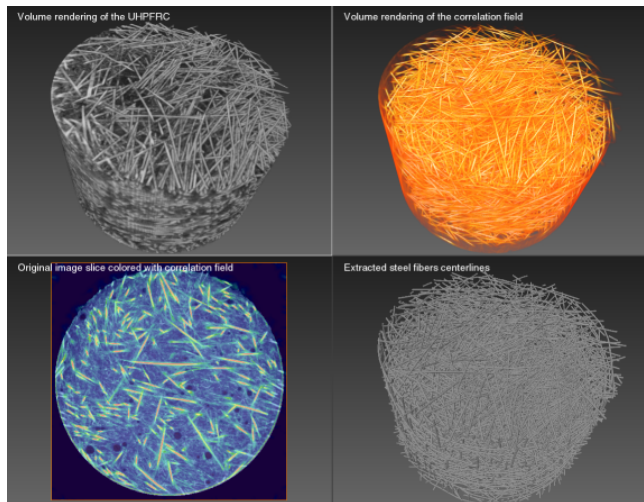
Figure 10.15: *Cylinder Correlation* and *Trace Correlation Lines* Applied on the UHPFRC Figure 10.4

extract fiber centerlines from the greyscale data (see *How to Trace Centerlines to Create a Spatial Graph of the Separated Fibers* for more details).

This tutorial explains how to quantify and analyze fibrous materials:

- *How to analyze contact points*
- *How to extract statistics about the fiber's shape*
- *How to calculate and plot the distribution of fibers based on a given property*
- *How to remove fibers that are touching the bounding box*
- *How to filter fibers based on properties*
- *How to extract and visualize local statistics*
- *How to plot orientations in 3D*

The dataset used in this tutorial is an Ultra High-Performance Fibre Reinforced Concrete (UH-PFRC), courtesy provided by Dr. Alenka Mauko and Dr. Aljosa Sajna from the Slovenian National Building and Civil Engineering Institute (www.zag.si; aljosa.sajna@zag.si), Log Cezsoski Bridge, Slovenia, WP7 project Assessment and Rehabilitation of Central European Highway Structures (ARCHES) http://arches.fehrl.org/.

The tutorial will be applied on a cropped volume of this dataset. To apply the workflow to your data, you might need to adapt these parameters as explained below.

The tutorial assumes that you are familiar with the basic concepts of Avizo. In particular, you should know how to load files, how to interact with the viewer, and how to connect modules to data objects. All these topics are covered in Chapter 2 - Getting Started of the Avizo User's Guide. Note that the modules presented in this tutorial require an NVIDIA graphics card supporting CUDA Compute Capability 1.3 or higher. To find your graphics card CUDA Compute Capability, see NVIDIA's list of CUDA GPUs (http://developer.nvidia.com/cuda-gpus/).

10.3.1   Analyzing Contact Points

This tutorial will show how to convert a *Spatial Graph* to a binary dataset to access quantification modules available from images. As an example, the fibers' contact areas will be identified and quantified:

1. Load data/tutorials/fibertracing/concrete_tutorial.hx from AVIZO_ROOTdirectory into the Avizo workspace.
2. Attach a *Convert Geometry To Label* module in category *Convert* to the output of the *Trace Correlation Lines* module (*concrete.CorrelationLines*).
3. Set the other parameters as shown in Figure 10.16.
4. Once all parameters have been configured, click **Apply**. This module will create a label data, in which the fibers' centerlines are individually labeled. There is a direct correspondence between the label Id and the graph Id.
5. Attach a *voxelized rendering* module to the converted dataset to visualize it inside the original

Figure 10.16: Default Parameter Settings of the *Convert Geometry To Label* Module

greyscale dataset (see Figure 10.17).
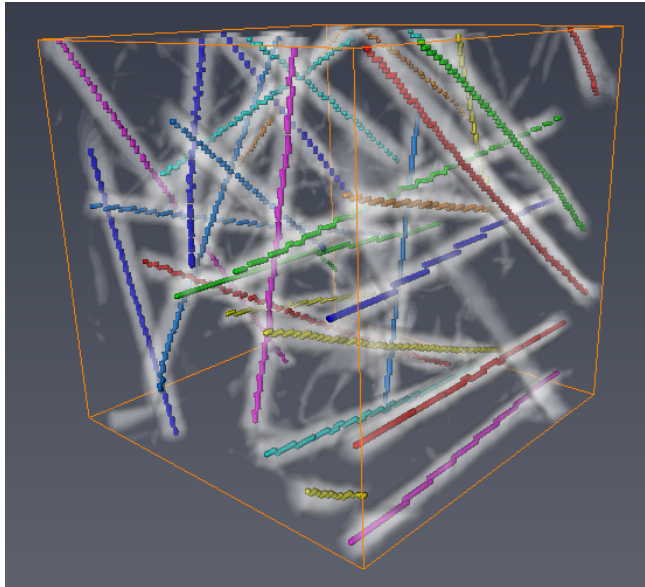
Figure 10.17: Voxelized Rendering of the Correlation Lines

6. To dilate the centerlines back to fit the original data, attach a *Ball Dilation* module to *concrete.converted* and set the parameters as shown in Figure 10.18:

7. Once all parameters have been configured, click **Apply**. Attach the *voxelized rendering* module to the *Ball Dilation* result. The centerlines are dilated back to their original size as shown in Figure 10.19.

8. Finally, contact points can be identified using a *Label Interfaces*. Attach a *Label Interfaces* module to the *convert.dilated* data and set the parameters as shown in Figure 10.20.

9. Click **Apply**.

10. Contact points are now identified. Attach the *voxelized rendering* module to *concrete.interfaces* (see Figure 10.21).
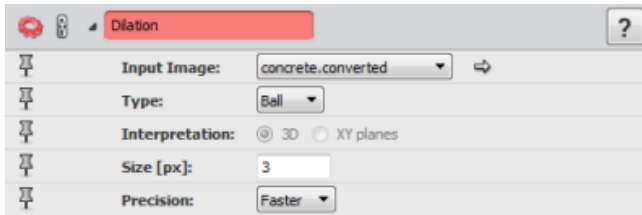
Figure 10.18: Default parameter settings of the *Ball Dilation* module
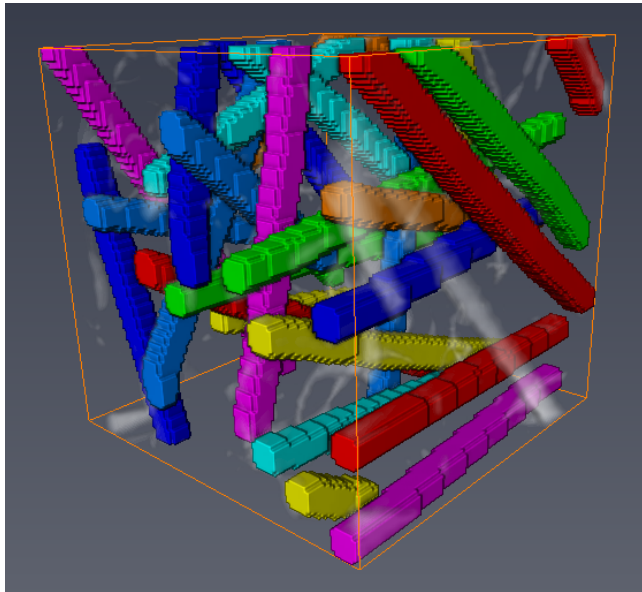
Figure 10.19: Dilated Centerlines
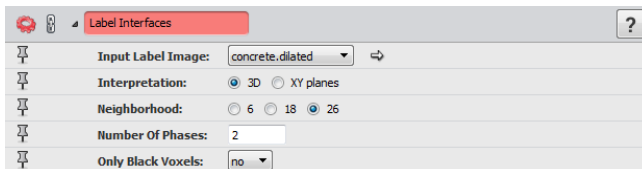
Figure 10.20: Default Parameter Settings of the *Label Interfaces* Module

11. Those contact points can be quantified. Label the contact point using a *Labeling* module while leaving the default values and then click **Apply**.

12. Attach a *Label Analysis* to the labeled contact points *concrete.labels* to get basic statistics. Keep
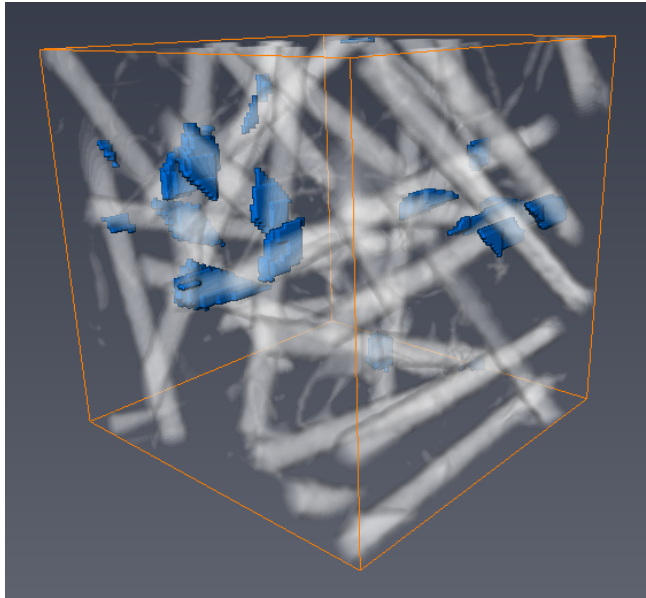
Figure 10.21: Identified Contact Points

the default values and then click **Apply** (see Figure 10.22).

Volume3d	Area3d	BaryCenterX	BaryCenterY	BaryCenterZ
132	231.281	480.894	733.917	493.765
248	306.75	508.274	733.637	496.444
480	618.802	497.398	734.129	502.833
189	237.518	471.143	725.608	508.365
231	259.572	505.234	736.329	510.238

Figure 10.22: Basic Statistics on Contact Points

This tutorial is available here: data/tutorials/fibertracing/concrete_contact_tutorial.hx

10.3.2   Extracting Statistics about the Fibers' Shape

In this step, statistics about the fibers' shape are computed such as: *Curved length*, *Diameter*, *Cross section area*, *Area*, and *Cross section perimeter*.

1. Load data/tutorials/fibertracing/concrete.am and data/tutorials/fibertracing/concrete.CorrelationLines.am from `AVIZO_ROOT`directory into the Avizo workspace.

2. Attach an *Interactive Thresholding* module to *concrete.am* with default parameters and click **Apply**.

3. Attach a *Fiber Shape Statistics* module to *concrete.CorrelationLines.am* and set the parameters as shown in Figure 10.23 and click **Apply**.
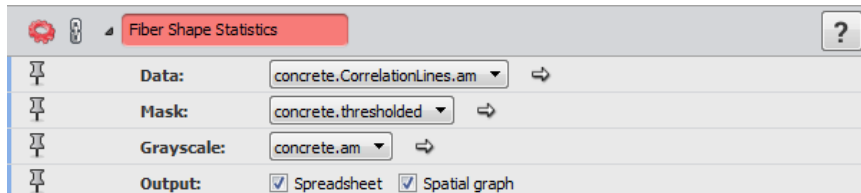
Figure 10.23: Parameter Settings of the *Fiber Shape Statistics* Module

4. The module will output the following statistics in a spreadsheet (see Figure 10.24).

	GraphID	Curved Length	Diameter	Area	Volume	Cross Section Area	Cross Section Perimeter
1	0	102.34913	7.2007351	2743.625	4168	40.723354	26.80653
2	1	76.228447	7.1643667	1864.4454	3073	40.313034	24.458656
3	2	105.9229	5.8366013	2059.6348	2834	26.755308	19.444658
4	3	99.179001	4.6413164	1511.0873	1678	16.918903	15.23596
5	4	68.367668	6.323328	1441.151	2147	31.403732	21.079424

Graph Summary

Figure 10.24: Shape Statistics on Fibers

This tutorial is available here: data/tutorials/fibertracing/concrete_shape_stat_tutorial.hx

10.3.3 Plotting the Distribution of Fibers

Avizo provides ways of effectively plotting the distribution of samples of a given property within a volume. For example, to know the diameter of the fibers that take the most space in a fibrous dataset, it is possible to plot the fibers' diameter versus the cumulated volume area:

1. Load data/tutorials/fibertracing/concrete_shape_stat_tutorial.hx from `AVIZO_ROOT`directory into the Avizo workspace.

2. Attach a *Distribution Analysis* module to the output of the *Fiber Shape Statistics* module, set the parameters as shown in Figure 10.25, and click **Apply**.

3. Attach a *Plot SpreadSheet* module to the output and set the parameters as shown in Figure 10.26, and click **Show** in the *Plot* port (see Figure 10.27).
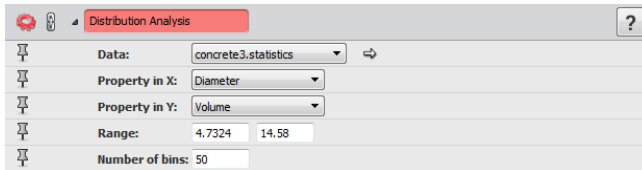
Figure 10.25: Parameter Settings of the *Distribution Analysis* Module
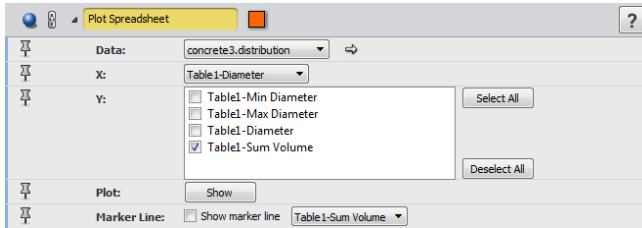
Figure 10.26: Parameter Settings of the *Plot SpreadSheet* Module
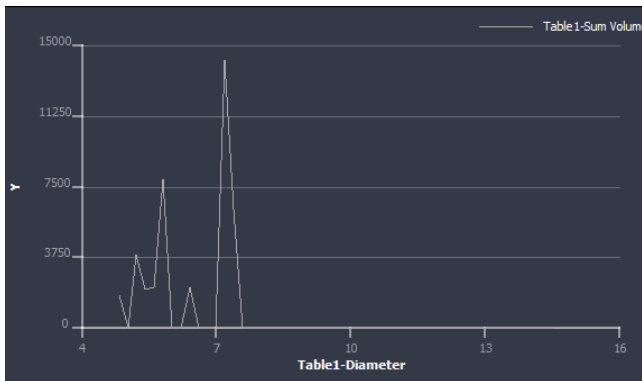
Figure 10.27: Fibers' Distribution Plot

10.3.4   Removing Fibers that are touching the Bounding Box

In some use cases, it is better to remove the fibers that are not completely inside the bounding box.Statistics of such fibers might be biased. For example, *Fiber length distribution* would be wrong because length is calculated on fibers that are not entirely inside the sample. To remove such fibers, attach a *Filter Spatial Graph By BBox Contact* module to the *Spatial Graph* representing the fibers and click **Apply**. This step can be added to the preceding example to make sure statistics are calculated on full fibers only. The Figures 10.28 and 10.29 show respectively the display before and after removing the fibers touching the bounding box.
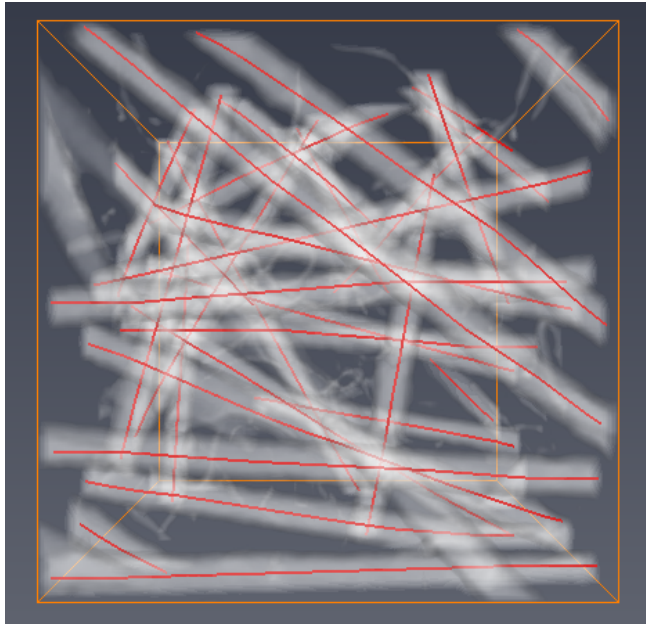
Figure 10.28: Display Before Removing Fibers

10.3.5 Filtering Fibers

In addition to removing fibers that are not touching the bounding box, it may be useful to filter out fibers that do and do not match a user-defined criterion. The *Spatial Graph Filter* module allows filtering samples based on user-defined equation using property values set as inputs of the module.

1. Load data/tutorials/fibertracing/concrete_shape_stat_tutorial.hx from AVIZO_ROOT directory into the Avizo workspace.
2. Attach a *Spatial Graph Filter* module to the spatial graph result of the *Fiber Shape Statistics* module and set the parameters as shown in Figure 10.30, and click **Apply**.

The data is filtered, so that only fibers matching the user's equation remain. The Figures 10.31 and 10.32 show respectively the display before and after filtering.

10.3.6 Calculating and Visualizing Local Statistics

Avizo provides ways of computing statistics in local areas of the fibrous sample rather than per individual fibers, which can be done using *Spatial Graph Statistics*. Such computation can help better assessing the sample's heterogeneity. For example, density maps allow you to identify where the
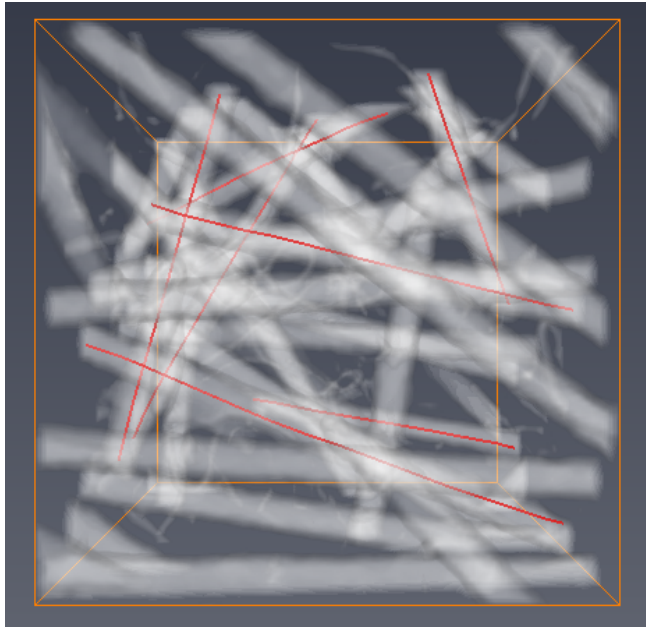
Figure 10.29: Display After Removing Fibers
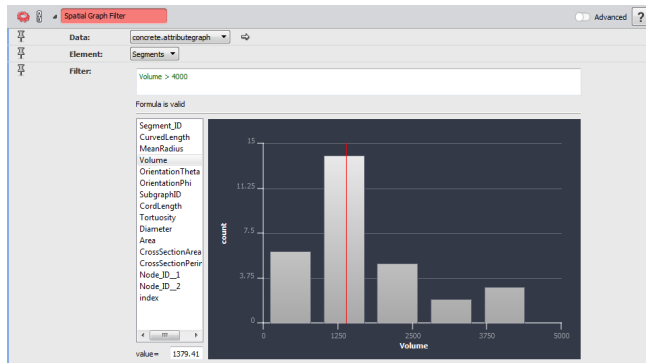
Figure 10.30: Parameters Settings of the *Spatial Graph Filter* Module

majority of fibers lay in the sample.

Areas of calculation are defined by blocks larger than the sample's grid, and blocks may overlap. The following modules compute local statistics:

- *Spatial Graph Local Statistics* computes local statistics from a *Spatial Graph* representing the
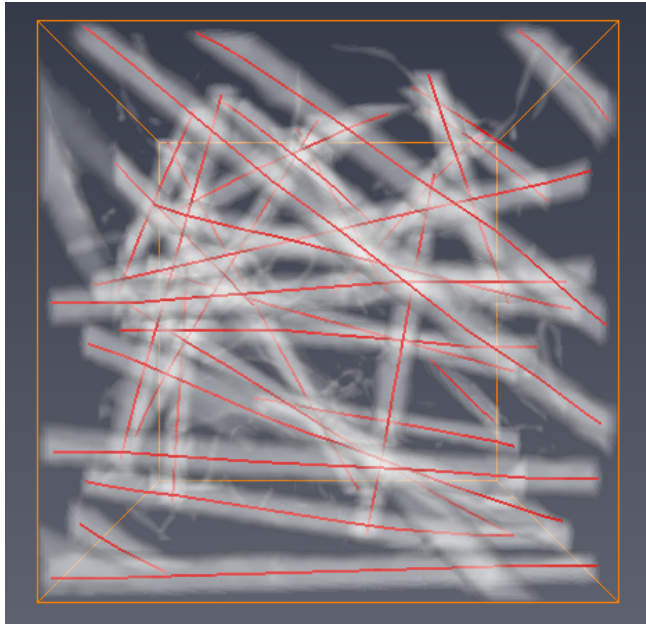
Figure 10.31: Before Volume Filtering

fiber's centerlines (generated from the *Trace Correlation Lines* module). Use this module when the fibrous sample can be separated. Statistics include:

- Volume Density
- Surface Density
- Tensor
- Orientation (i.e., vector field corresponding to the main eigen vector of the tensor)

- *Local Orientation* computes local statistics directly from a greyscale volume. Use this module when the fiber's centerlines cannot be traced. Statistics include:

  - Confidence
  - Orientation (Theta, Phi)
  - Eigen vectors
  - Moment of inertia

As an example, local statistics from the concrete dataset are extracted, using the *Spatial Graph Local Statistics* module:
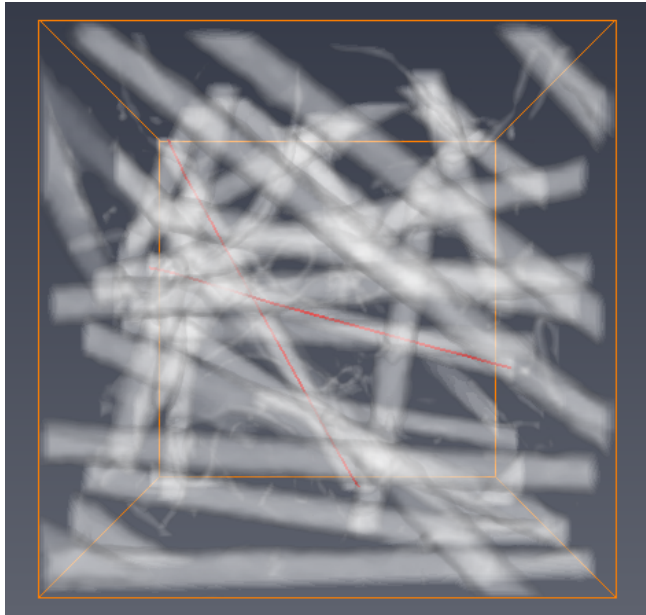
Figure 10.32: After Volume Filtering

- Load data/tutorials/fibertracing/concrete.am and data/tutorials/fibertracing/concrete.CorrelationLines.am from AVIZO_ROOT directory into the Avizo workspace.
- Attach a *Spatial Graph Local Statistics* to *concrete.CorrelationLines*.
- Set the parameters as shown in Figure 10.33:
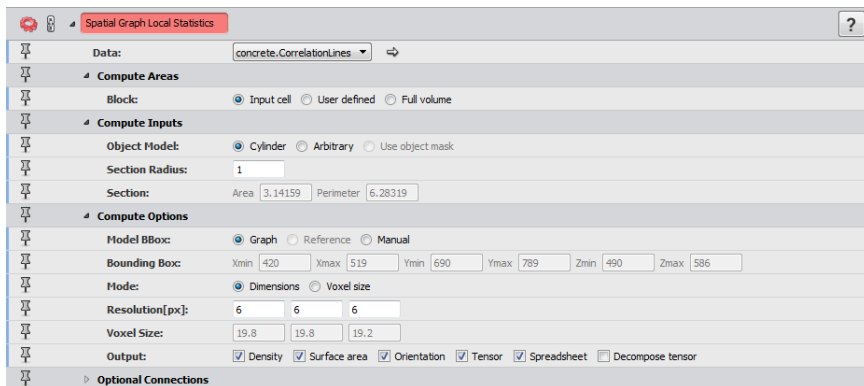
Figure 10.33: Parameters of the *Spatial Graph Local Statistics* Module

The *Data* of the module are the centerlines of the fibers. From this, local statistics relative to the volume fraction, surface area, and orientation tensors, can be computed. The major orientation corresponding to the main eigenvector of the tensor can also be extracted. When a mask of *Object Mask* is provided, the volume fraction is computed from the number of voxels, and the surface area is derived from a triangle mesh generated from the mask. In the present use case, no mask is provided and the calculation of those statistics is based on a perfect cylindrical fiber model with a known section radius. To determine a reasonable section radius, the measurement tool may be used (i.e., *Measurement* in the viewer toolbar). The third optional input is a mask to limit the local statistics computation to a given area in the sample. The local statistics are integrated within blocks, or subvolumes, regularly distributed in the bounding box of the volume. The optional *Reference* may be used to define a different bounding box for the data. The number of blocks and/or the distance between the centers of adjacent blocks is specified through the ports *Resolution* and *Voxel Size(units)*. The block size is, by default, such that there is no overlap between adjacent blocks. However, this can be configured through the port *Block* if necessary.

- Once all parameters have been configured, click **Apply**. The spreadsheet output lists all statistics for each block. Select the spreadsheet output and click **Show** to view the statistics in the table view. Note: This is not recommended, the number of blocks is very large.

The statistics can also be generated in the form of scalar images (i.e., volume fraction and surface area), vector field (i.e., major orientation) and tensor fields (i.e., orientation tensors), useful for visualization and data export. For instance, attach an *Ortho Slice* to *concrete.am*, then create a colorwash to visualize volume density on top of the data (see Figure 10.34).
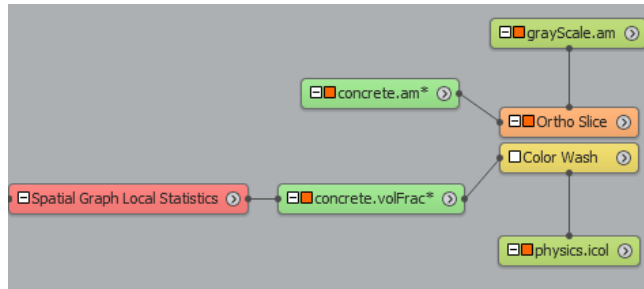
Figure 10.34: Density Visualization on Top of the Data

The same setup can be done with the area density map to compare zone of influence of the fibers' volume and the fibers' area (see Figure 10.35).

Orientation tensors may be visualized as ellipsoids using the *Tensor View* module.

- Attach a *Tensor View* module to the tensor output of the *Spatial Graph Local Statistics*.
- You may set the *Module* port of the *Tensor View* to a different plane than the default clipping plane. For example, use the *ortho slice* on which the density map is visualized. Set the parame-
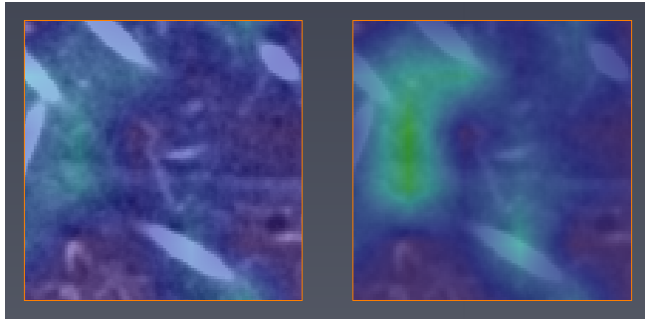
Figure 10.35: Comparison Between the Volume Fraction and Surface Area Densities

ters as shown in Figure 10.36.
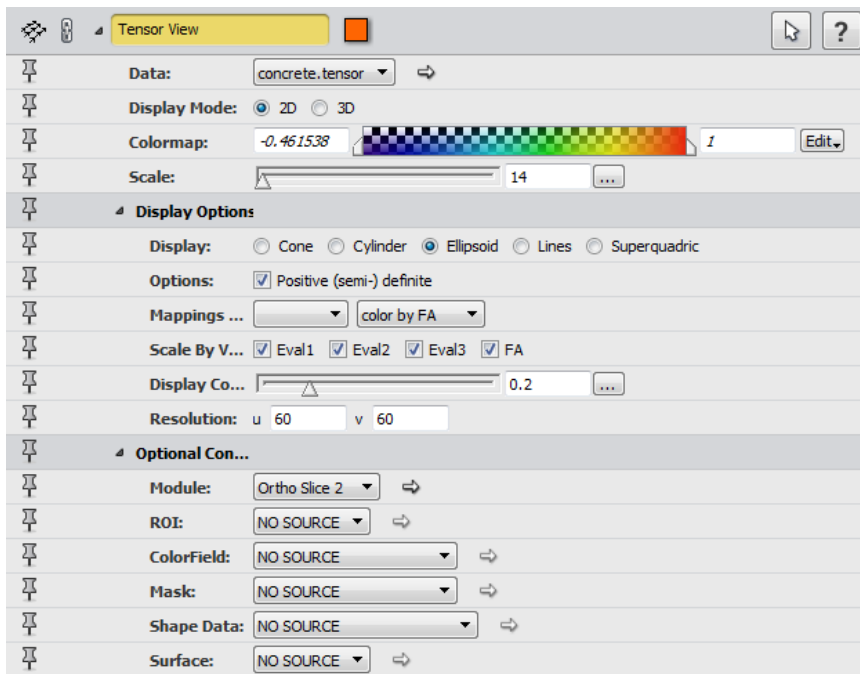
Figure 10.36: Tensor View Parameters

Tensors are now mapped on top of the density map as shown in Figure 10.37.

Note that you have to click **Apply** or select *Auto-refresh* to update the *Tensor View* module display.
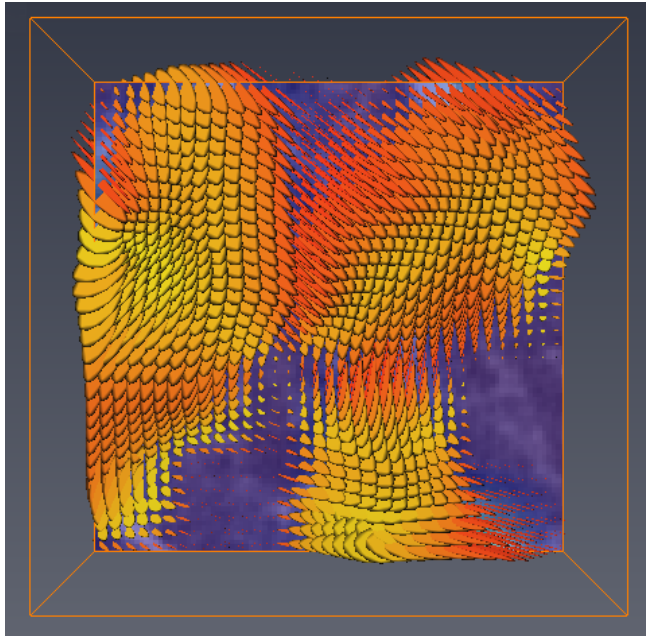
Figure 10.37: Tensors Mapped on Top of the Density Map

The resolution and scale in particular need to be adjusted. Please refer to the documentation of *Tensor View* for more details.

Finally, it is also possible to visualize orientations using a *Volume Rendering* module by converting the orientation output to RGB data using the *Vector To RGB* module.

10.3.7   Plotting Orientations in 3D

The *Plot 3D Orientation* module allows for understanding visually the orientation distribution on a 3D plot represented by a half or full sphere. The module also allows mapping a property on height or color, or both at the same time.

1. Load   data/tutorials/fibertracing/concrete_shape_stat_tutorial.hx   from   `AVIZO_ROOT`directory into the Avizo workspace.
2. Attach a *Plot 3D Orientation* module to the spatial graph output of the *Fiber Shape Statistics* module.
3. Set the parameters as shown in Figure 10.38.

The external viewer should display the orientation in 3D as shown in Figure 10.39.
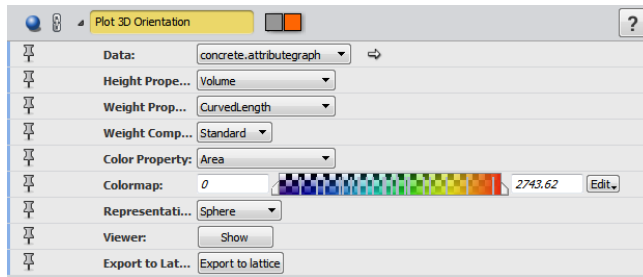
Figure 10.38: Parameter Settings of the *Plot 3D Orientation* Module
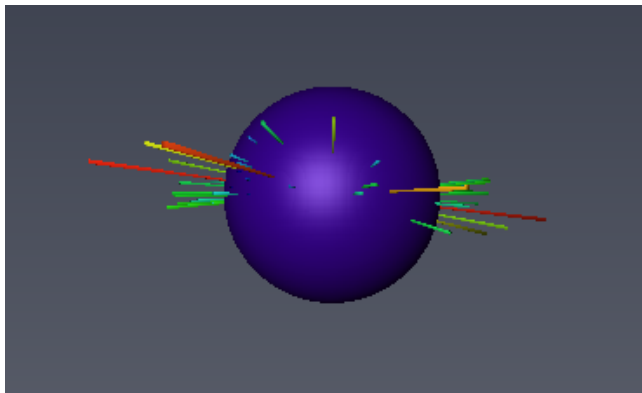
Figure 10.39: Fibers Orientation

The following angular representation convention is used:

- Theta represents the angle formed with the Z-axis and varies in the range [0-90] degrees.
- Phi is the angle in the XY plane, measured from the X-axis towards the Y-axis in the range [0-360[degrees.

There are two weighting computation methods:

- *Standard*: A sample of small length should not account as much as a sample of high length on height mapping.
- *Averaged*: You can calculate the average by adding the weight of each sample with the same orientation and dividing the total weight by the number of samples. For example, if the weight property is length and there are 5 samples of a given orientation, the weight assigned should be based on total length/5.

Using the *Export to lattice* button, the module outputs the plotted data as a regular 2D scalar field that can be visualized using the *Bar Chart Slice* or the *Height Map Slice* module (see Figures 10.40 and 10.41).
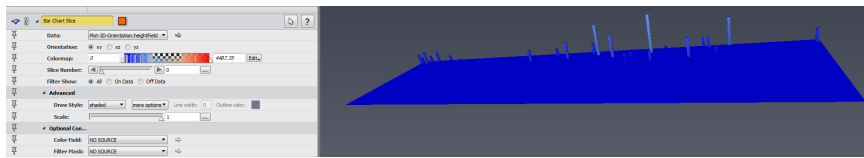
Figure 10.40: Plotted Data Displayed Using a *Bar Chart Slice* Module
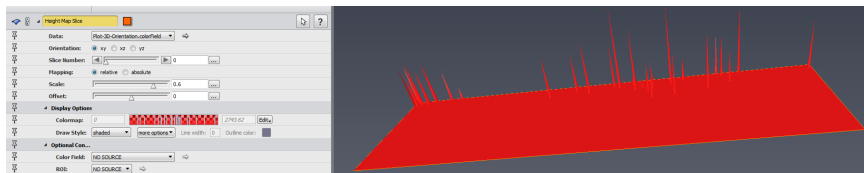
Figure 10.41: Plotted Data Displayed Using a *Height Map Slice* Module

Chapter 11

Avizo XWind Extension User's Guide

The Avizo XWind Extension provides tools for creating, analyzing, visualizing, and presenting numerical solutions from CAE and CFD simulations.

The Avizo XWind Extension provides advanced support for the following:

- Defining simulation inputs
    - 3D mesh generation for numerical simulation applications such as flow, stress, or thermal analysis
    - Boundary conditions
    - Export of surfaces or 3D meshes to numerical solvers
- post-processing of result data coming from solvers
    - Powerful visualization and analysis of scalar, vector, and tensor fields coming from either simulation or measurements
    - Import from major CAE file formats with low memory footprint and fast management of model data
    - Flexible probing and measurements
    - Extensive computation of derived variables and statistics on simulation results
    - Advanced feature extraction tools such as vortex core lines or critical points

The Avizo XWind Extension applies to the following:

- exploration, analysis, and comparison of data coming from simulation or measurements
- modeling from 3D images for Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD), Computer Aided Design (CAD), and Rapid Prototyping

- high-quality presentation and communication, from movie generation to remote collaboration, immersive displays, or virtual reality (requires *Avizo XScreen Extension*)

The Avizo Lite Edition provides base post-processing support limited mostly to visualization. Avizo XWind Extension extends the Avizo Lite Edition feature set by adding advanced Delaunay-based mesh generation and post-processing computation, and analysis of derived variables, statistics and feature extraction. Avizo XWind Extension brings also the robust support for unstructured mixed meshes made up of tetrahedra, hexahedra, pyramids, and wedges. Most CAE file formats are supported in Avizo XWind Extension only. See the *File Formats Index* of the User's Guide for a complete list of supported file formats.

By following the provided tutorials, you will learn how to use these modules on your own datasets. To start you do not have to read the *tips* sections.

The data used for this tutorial is normally installed in the directory `data/tutorials/cfd-fea-advanced` under your `AVIZO_ROOT` directory.

Defining simulation inputs:

- *Avizo XWind Extension Meshing Workroom*
- *Avizo XWind Extension Meshing Tutorial*

Post-processing solvers' results:

- *Getting Started with Reading and Visualizing CAE/CFD Data*
- *Avizo XWind Extension Model Information and Display*
- *Avizo XWind Extension Scalar Field Display*
- *Avizo XWind Extension Vector Field Display*
- *Avizo XWind Extension Statistical and Arithmetic Computations*
- *Avizo XWind Extension Vorticity Identification*
- *Avizo XWind Extension Measurements*

11.1   Meshing Workroom

This workroom is only available on Microsoft Windows platform.

The Meshing Workroom is a pre-processing tool for creating simulation inputs. It allows computing a *tetrahedral* mesh with boundary conditions. The mesh can then be exported to various solvers' file formats such as Fluent, COMSOL Multiphysics (c), etc... see *XWind file formats*.

The Meshing Workroom does not use the legacy meshing engine available in Avizo, which is based on advancing front technique. The meshing engine used in the workroom is based on Delaunay refinement and is available with the XWind license. Currently, the Meshing Workroom is only

available on the Windows platform.

The Meshing Workroom takes a *label field* (currently, only 8-bit Label Fields are supported) as input to generate a mesh. A *label field* may be created with the *Segmentation Workroom* or other tools such as *Interactive Thresholding*, *Auto Threshold*, etc...

No other inputs are required. Optionally, various *uniform scalar fields* may be used to control the mesh density (see *Advanced mode* (Meshing Tab)).

For a complete workflow description, please refer to the *Meshing Workroom Tutorial*.

The documentation of the Meshing Workroom is separated into the following parts:

- *First steps*
- *Inspecting the generated mesh*
- *Fast meshing versus precise meshing*
- *Controlling the refinement of the mesh*
- *Advanced meshing mode*
- *Troubleshooting*
- *Assigning boundary conditions to the mesh*
- *Exporting the mesh*
- *Scripting the room: TCL command list*
- *Progress bar indications*

All illustrations using the corn kernel dataset are courtesy of, Pawan S Takhar, University of Illinois at Urbana-Champaign.

11.1.1 First Steps

To activate the Meshing Workroom, press the Meshing Workroom icon in the workroom toolbar. The main parts of this workroom include:

- **A general area**: This area of the workroom lets you see or select the mesh currently set in the workroom. It also lets you select the solver type that is targeted in the workflow. An Export button is available whenever a mesh is created. It will export the mesh in the format selected from the type field (figure 11.1).
- **A Meshing tab**: This is where you build the mesh. A label input can be set using the label data field. Once you set the Create button so that it becomes accessible, you may generated a mesh (figure 11.2).
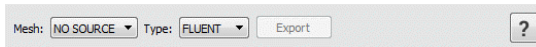
Figure 11.1: The General Area

Figure 11.2: The Meshing Tab

- **A Boundary tab**: This is where boundary conditions are assigned to the mesh. Various tools let you select the elements (triangles) which are part of a given boundary condition group (given name and ID). This tab is available only if the selected file format allows exporting boundary conditions (Figure 11.3).

The very basic workflow consists of the following:

1. Create a *label field* (see for example *Segmentation Workroom*) with as little noise (small dots, misalignment, etc...) as possible.
2. Set the export format in the general section located above the tabs. The Boundary tab will be available if the export format allows it.
3. Set the label into the Meshing tab's label data input of the Meshing Workroom.
4. Set the meshing options and create the mesh by clicking **Create**. Once created, the mesh will be displayed in the main viewer of the room.
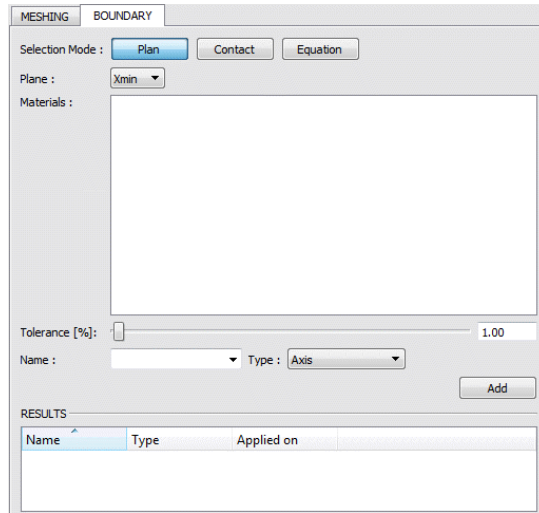
Figure 11.3: The Boundary Tab

5. (Option) Assign the boundary conditions to the mesh if the export format allows it.
6. Finally, export the mesh into the selected format by clicking **Export** in the general section located above the tabs.

Various mesh quality and optimization settings are available and are described in the next sections of the Meshing Workroom documentation. You can either create a new version of a mesh for a given *label field* using *Create*, or override the current mesh set in the Mesh input using **Replace**.

After the mesh creation, the mesh is automatically visualized in the main viewer (Figure 11.4).

The generated mesh is automatically set in the Mesh input of the Meshing Workroom.
The mesh properties (number of nodes, triangles, tetrahedron) are available from the Project tab in the Properties area (Figure 11.5) when selecting the green module corresponding to the mesh:

The console outputs some statistics about the mesh generation.

11.1.2   Inspecting the Generated Mesh

The materials' visibility of the generated mesh can be turned on and off to inspect the different parts of the grid.
The visibility of each individual material can be accessed from the Meshing tab in the materials section (Figures 11.6 and 11.7).
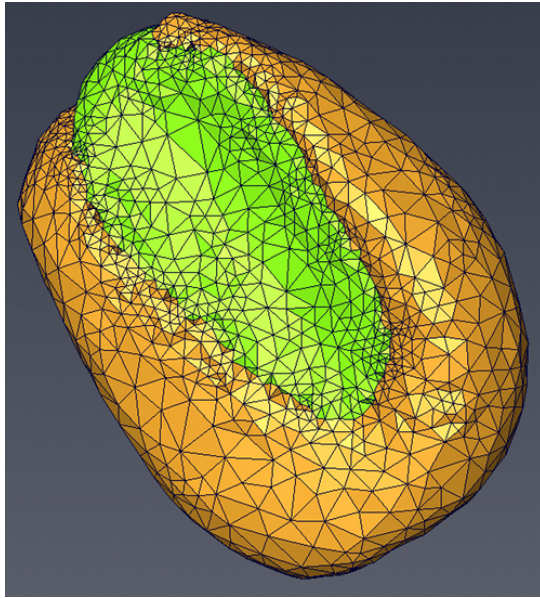
Figure 11.4: View of The Mesh
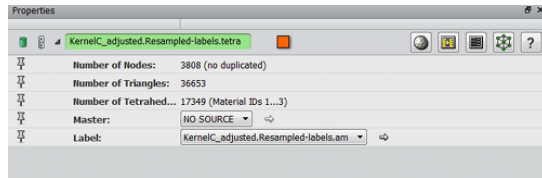
Figure 11.5: Mesh Properties
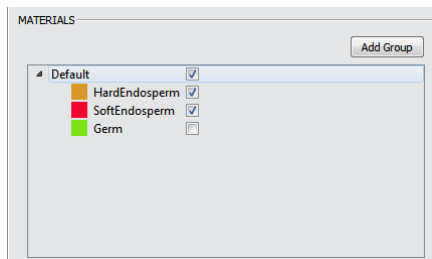
Figure 11.6: Material Visibility
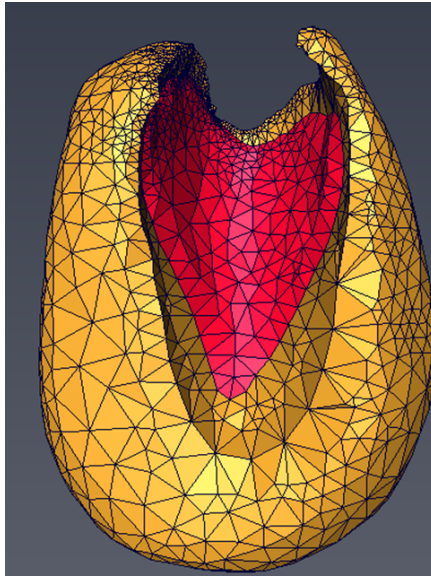
Figure 11.7: Mesh Without One Material

To inspect interior elements, a clipping plane dedicated to the visualization of tetrahedral mesh is available from the main toolbar. To activate clipping, switch **Clip** button on in the main toolbar. See result on Figure 11.8

The clipping plane automatically clips any tetrahedron that does not intersect the plane and is on the clipped side. Various options are available to manipulate the clipping planes:

- Select the plane with the left mouse button and move the mouse while keeping the left button pressed. This will translate the plane along its normal.
- Set the plane perpendicular to the main axis (X, Y, Z) by clicking **XY**, **XZ**, or **YZ** in the main toolbar.
- Click **Rotate** in order to show a dedicated manipulator located at the center of the plane (Figure 11.9). Select the manipulator with the left mouse button and rotate by moving the mouse while keeping the left mouse button pressed.

- Click **Clipping plane side selector** in main toolbar in order to change clipping side.
- Turn **Interactive Mode** off if the interaction with the clipping plane is slow
  In this case, the clipping is not enabled while you interact with the software, and is only applied when the left mouse button is released.
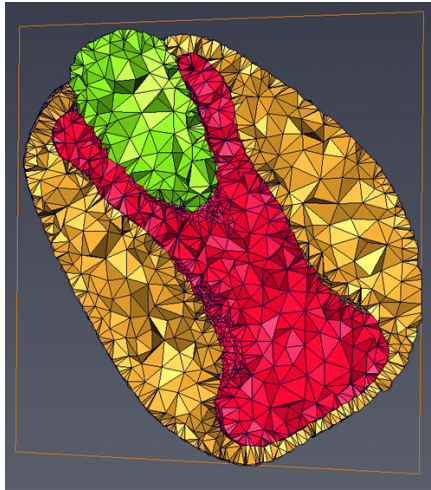
Figure 11.8: Clipped Mesh
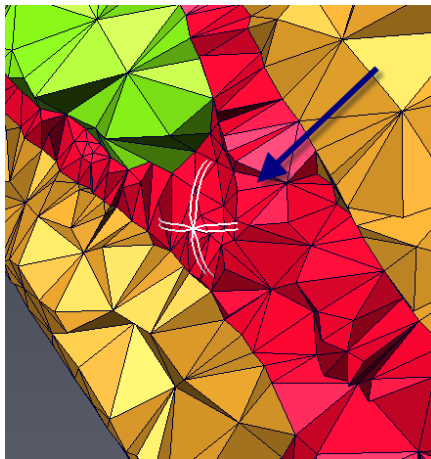
Figure 11.9: Clipping Plane Rotation Manipulator

11.1.3 Fast Meshing versus Precise Meshing

By default, the fast meshing mode is activated in the Meshing tab. This mode allows for fast prototyping the meshing of a given label. It is fast, because the meshing is not constrained by any geometry. The resulting mesh is fast to build, but it is missing precision at the boundaries between labels. Select the **Fast Meshing** checkbox to disable the resulting mesh.

When the fast meshing option is not used, the meshing engine is constrained to respective interfaces at boundaries between labels. Those interfaces are defined by surfaces that are generated using the *Generate Surface* module. The module is used with the constrained mode and the *Fit To Edge* option. Unless the label has weights (assigned from the Segmentation Editor or other methods), in which case the module uses the *Existing Weight Smoothing* option.

It is recommended that you input a smoothed label with weights, so there is no or little staircase areas that would need to be topologically preserved by the mesher (those areas will need to be very refined). The label should also be as clean as possible, with all unnecessary small materials areas cleaned out because such areas might create unnecessary over refinement. For example, in the figure 11.10, the small orange material dot in the middle of the brown material will be preserved so the mesher will over refine around that area:
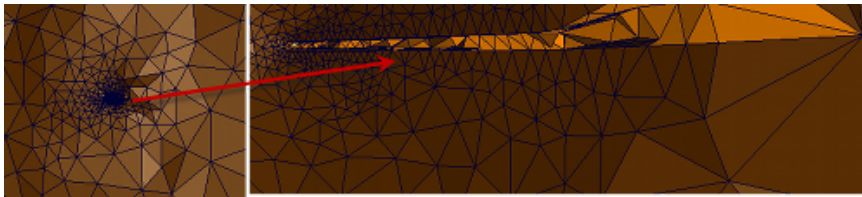
Figure 11.10: Not clean label image example

It is possible to specify a different surface than the one generated internally, using a hidden port.

To unhide it, you can type *"theMeshingTabController" enableSurfaceSelection 1* in the console. A *Surface Data* port appears in the tab. Set your surface there.

This port allows you to use a custom surface (e.g., smoothed or simplified by the user).

Using its own surface might have several advantages:

- First, if you set a surface to the port, the internal surface generation will not happen so you can generate your surface and then test various settings with fast meshing mode deactivated. Otherwise at each create or replace, the surface will be generated internally.
- Second, if the internal surface is dense, the meshing might take a long time. You can control the size of your surface (through simplification) to speed up the meshing. However, the input surface must respect the following criteria:
    - Watertight surface
    - No autointersection between the different materials (this can be checked with the autoin-

tersection test of the surface editor)

11.1.4 Optimization

In both modes (fast meshing or not), it is possible to optimize the tetrahedron node locations to reduce the amount of sliver in the mesh. A sliver tetrahedron is formed by placing its four vertices near the equator of its circumsphere (e.g., flat tetrahedron). Slivers may slow down the convergence of numerical simulation. The less sliver the better.

The **Optimize Mesh** option aims at reducing the amount of sliver by perturbing them through random vertex relocation. The mesh generation may be a bit longer if the option is activated.

More optimization options are available in the *tetra mesh module*.

11.1.5 Controlling the Refinement of the Mesh

Many options are available to fine tune the refinement of your mesh.

11.1.6 Slider Quality and Grouping

The **Mesh Quality** slider provides five quality levels to mesh the input *label field*, from low to high. Each quality level automatically sets a number of meshing options. You can check the effect of the different quality levels on the meshing engine settings by looking how the values within the Parameters Group are set in the greyed out area (Figure 11.11).
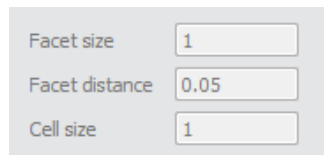
Figure 11.11: Parameters Group

Changing the quality will affect the constant there. Each quality level sets a factor that is multiplied by the voxel size (for isotropic voxel): the higher the quality, the lower the factor. The average voxel size along X, Y and Z is used for anisotropic voxels. Note that the mesh quality does not ensure that every cell size will have the same dimension, because the highest constraint of the mesher is to preserve the topology of the input label. It will, however, try to fit a given area with the indicated parameters if possible. By default, the quality set by the slider is applied on the default group. The selected group is written in the meshing tab.

New groups may be added to assign various qualities to various materials. A new group is created by clicking **Add Group**. It is then possible to simply drag and drop any material in the newly created group. When selecting the group or any material in the group, the mesh quality slider is then applied to this specific group. It is possible to create multiple groups with various qualities assigned. When a group is removed, all materials which were present in the group go back to the default one that cannot be removed. See an example in Figure 11.12.
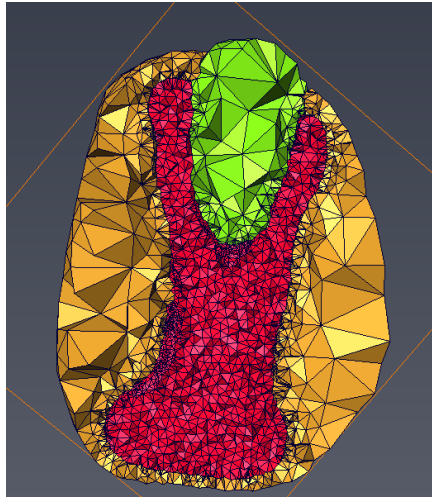
Figure 11.12: Material Grouped

11.1.7   Preserve Thin Structures

The Preserve Thin Structures option makes sure that thin structures are preserved: despite the mesh quality, which is set from the Mesh Quality slider. Practically, this option automatically sets the facet distance (approximation error of boundary and subdivision surfaces) to 0.25xvoxel size. This ensures that even the smallest structures are preserved. It will generate fine elements on boundaries, but still will try to respect the cell size internally. In addition, with **Fast Meshing** mode deactivated, the surface is internally generated with no smoothing to make sure no element is lost by smoothing operations. This mode is only available in non-advanced mode. If you need to get similar results in the advanced mode, you will need to use your own unsmoothed surface (with Fast Meshing mode deactivated)

Figure 11.13 and 11.14 show the difference between using this mode or not using it: Figure 11.13 shows mesh with Preserve Thin Structures option deactivated. A structure (pointed by the left arrow) is lost. Activating the mode will preserve the structures.
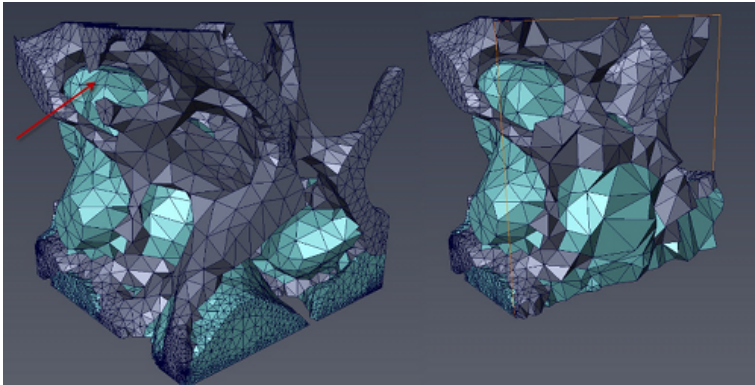
Figure 11.13: Without Preserve Thin Structures option
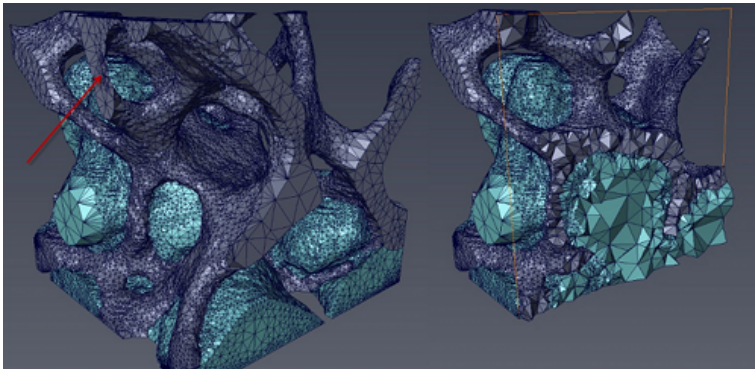
Figure 11.14: With Preserve Thin Structures option

11.1.8   Boundary Layer

The **Boundary Layer** option is set to have smaller tetrahedron on the boundary of the regions. The parameters are set by groups. The parameter thickness defines the thickness of the layer. Then the cell size defines the maximum size of the cells inside this layer.

11.1.9   Advanced Meshing Mode

By default, advanced parameters are not accessible. They are greyed out, and you can see how they are set when the **Mesh Quality** slider is changed (see *Slider Quality and Grouping* for more information).

To be able to customize the mesh quality (i.e., not using the presets of the Mesh Quality slider), enable
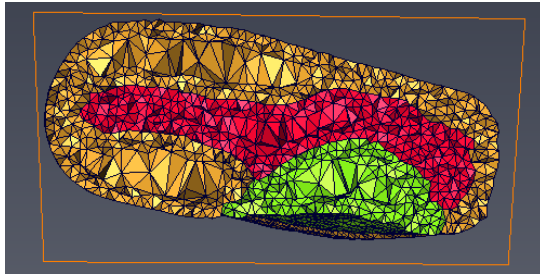
Figure 11.15: a boundary layer per material example

the advanced mode of the Meshing Workroom:

This will grey out the Mesh Quality slider and enable the advanced parameters (Figure 11.16).

Figure 11.16: Mesh Quality Customization

The following parameters are available:

- *Facet Size*: This parameter controls the size of surface facets (size of the triangles of surface facet). It provides an upper bound for the radii of a ball circumscribing the surface facet and centered on the surface patch.
- *Facet distance*: This parameter controls the approximation error of boundary and subdivision surfaces. The meshing engine will create small elements close to curved surfaces and large elements away from the surfaces.
- *Cell size*: This parameter controls the size of mesh tetrahedra. It provides an upperbound on the circumradii of the mesh tetrahedral.

When the advanced mode is turned on, optional scalar field inputs also become available (Figure 11.17).

Each individual advanced parameter can be set as a constant value or as a varying scalar field. If set as a scalar field, the advanced parameter constant is not available anymore and is reported as being managed by a varying field (Figure 11.18).

See the *Meshing Workroom Tutorial* to know how to create a scalar field that can control an advanced
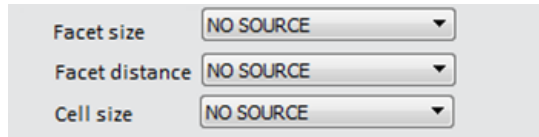
Figure 11.17: Field Parameters
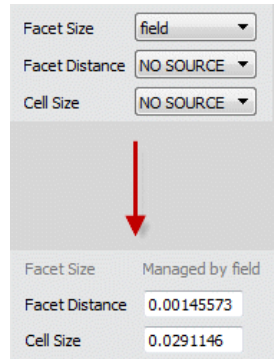
Figure 11.18: Field Selected

parameter. When a scalar field is used, it controls the parameter globally for the whole mesh, and not for a given group. You can mix scalar field that control a parameter globally and constant values that control other parameters individually per group.

11.1.10   Troubleshooting

With parameters between low and medium, you might get a mesh containing holes, like the following example created by meshing data/tutorials/motor.labels.am in fast meshing mode and medium quality.

This is because the object is thin and the approximation error tolerated by such parameters does not ensure its full preservation. In this case, you should select the thin structures option or lower the facet-distance setting in advanced mode. Check out the *Thin Structures section* for deeper understanding of this issue.

Sometimes you may get an inhomogeneous quality with some elongated triangles such as the following example created by meshing data/tutorials/motor.labels.am with a low quality, fast-meshing disabled and with thin structures ON :

This occurs because of a parametrization issue where the facet size parameter set by the user is not consistent with the facet size imposed by the engine to respect the geometry constraints: In some areas the mesh engine is forced to refine the facet to a smaller size than the specified facet size (typically when needing to respect thin structure). To fix the varying quality, from the project room, display the
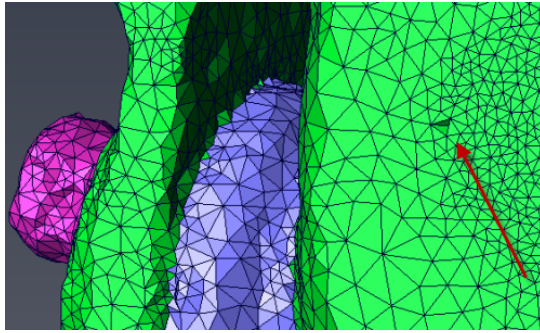
Figure 11.19: Mesh containing holes
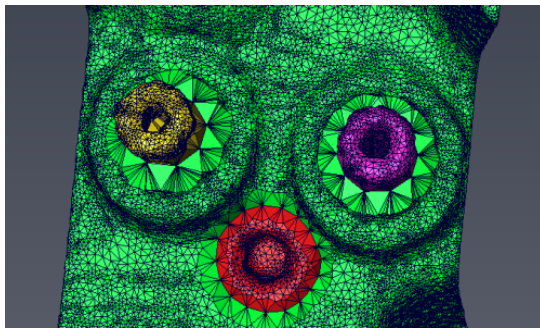
Figure 11.20: Inhomogeneous Mesh

mesh using a Tetra Grid View module, and measure the size of the facet outside the bad quality area using *Measure Tools*

In the meshing workroom in advanced mode, set the facet size parameter to the measured parameter (here, 0.005). Mesh again by hitting the create button.

11.1.11   Assigning Boundary Conditions to the Mesh

Boundaries may be assigned to the generated mesh from the Boundary tab.

The Boundary tab allows you to select triangle elements using three methods that can be enabled from the selection mode (Figure 11.23).

Selected triangles will be highlighted in red on top of the mesh in the main viewer (Figure 11.24).

- *Plan*: This mode allows you to select the triangles of a given material which lie close to the bounding box plan. The plan can be selected from the plan combo box. The material is selected
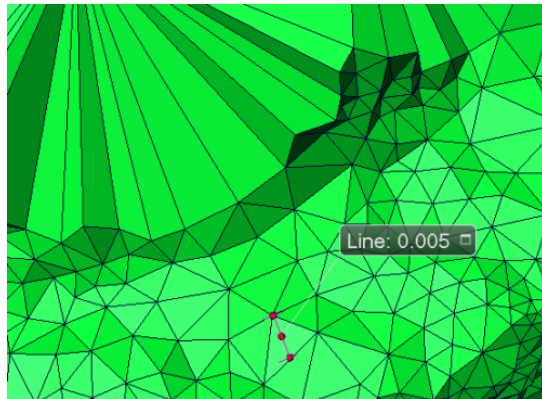
Figure 11.21: Facet Measure
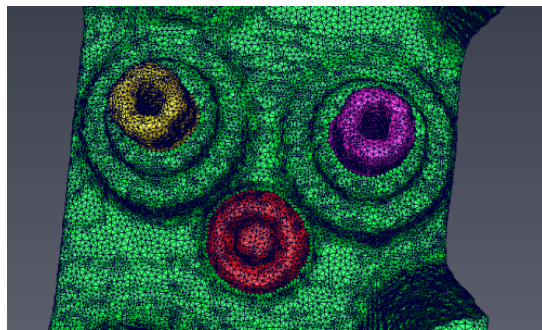
Figure 11.22: Homogeneous Mesh

Figure 11.23: Selection Mode

from the materials list (Figure 11.25). When the elements are not exactly in the bounding box plan, or follow a curved wall, it is possible to select them using a tolerance through a dedicated slider, that increases the area of selection away from the selected plan.

- *Contact*: The contact mode allows selecting interfaces between two materials. Select the two materials from the materials list (Figure 11.26).
- *Equation*: The equation mode lets you select any element of a given material based on a given equation (Figure 11.27). The vertex components are accessed through X, Y, and Z variables. A triangle is selected if all three vertice positions match the equation condition. For example, the
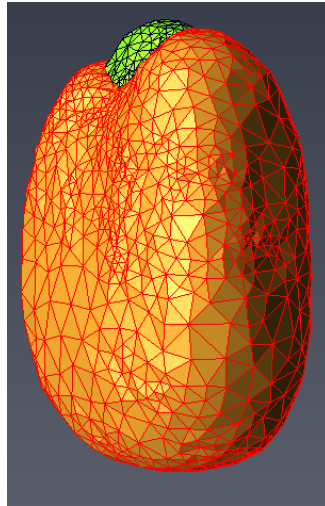
Figure 11.24: Selection Mode
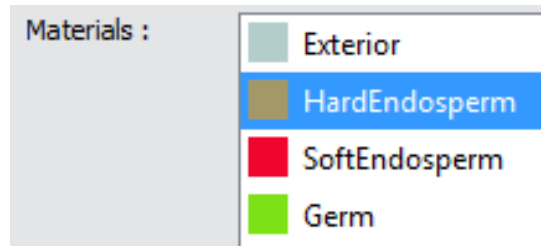
Figure 11.25: Material Selection
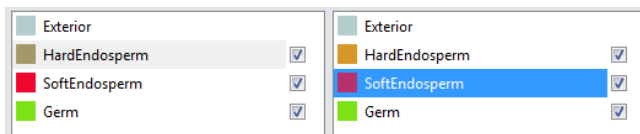
Figure 11.26: Contact Mode

"x <5" will select all triangles in which the X component of all three vertices is less than 5.

Once a selection is made, the boundary condition is added by selecting an identifier and a name. Once you click **Add**, the corresponding boundary condition is added in the results section (Figure 11.28).

Note that some characters are forbidden in the boundary name. If one of those characters are part of
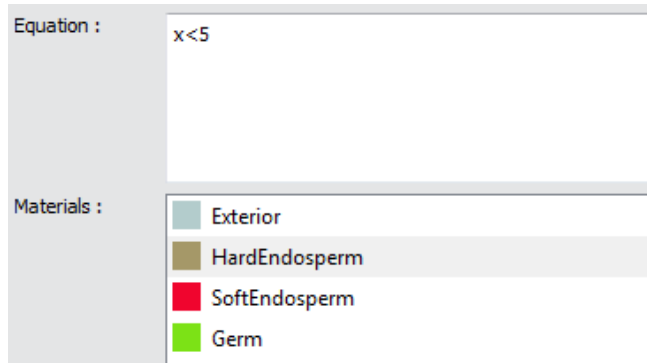
Figure 11.27: Equation Mode

Figure 11.28: Results Section

the name, **Add** will have no effect. Allowed characters include: a-z, A-Z, 0-9, - and _.
Selecting a boundary from the results section will result in the highlighting of the elements in green in the main viewer (Figure 11.29).

Highlight of a boundary condition can be disabled by clicking again on it in result section.

The names that are valid will be available from the Name combo box. It allows you to edit an existing boundary to add more elements.

11.1.12 Exporting the Mesh

When a mesh is generated, it is possible to save it to the target format by clicking **Export** in the general section of the meshing room.

11.1.13 Scripting the Room: TCL Command List

The following TCL commands are available to access the room from the command line or within a script:
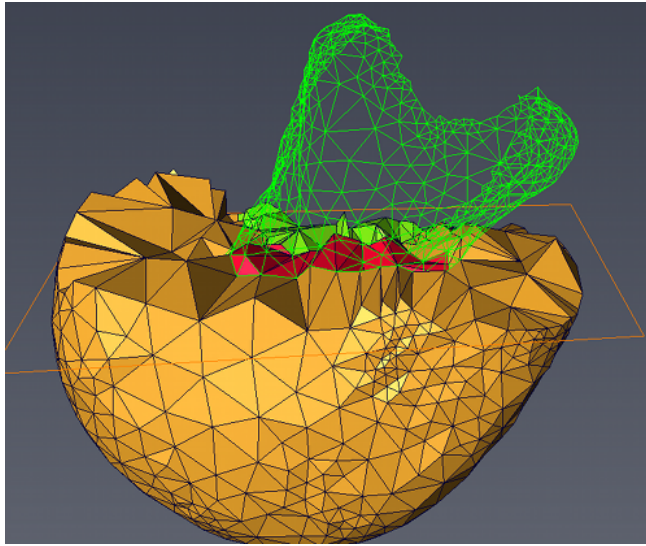
Figure 11.29: Result Section

- "theMeshingTabController" followed by the commands below can be used to script the meshing of the label data

  - `setLabelData <lableName>`
    set the label data into the meshing tab

  - `create`
    create the mesh

  - `update`
    replace the current mesh

  - `setMeshOptimizationEnabled <0|1>`
    activate/deactivate optimization

  - `isMeshOptimizationEnabled`
    optimization state

  - `setAdvancedMode <0|1>`
    activate/deactivate advanced mode of the meshing tab

  - `isAdvancedMode`
    advanced mode state

  - `setMeshQuality <groupId> <0|1|2|3|4>`
    set the mesh quality on a given group. Default group ID is -1.

- getMeshQuality <groupId>
  return the mesh quality of the given group ID

- setParameterValue <groupId> <0|1|2|3> <value>
  set the parameter value on a given group. (0 = edge size, 1 facet size, 2 facet distance, 3 cell size)

- getParameterValue <groupId> <0|1|2|3>
  get the parameter value of a given group. (0 = edge size, 1 facet size, 2 facet distance, 3 cell size)

- setParameterField <0|1|2|3> <scalarField>
  set the field for a parameter (0 = edge size, 1 facet size, 2 facet distance, 3 cell size)

- getParameterField <groupId> <0|1|2|3>
  get the name of the field of a parameter (0 = edge size, 1 facet size, 2 facet distance, 3 cell size)

- setSurfaceData <surface data name>
  set the surface to mesh if the fast meshing is not activated

- enableSurfaceSelection <0: disabled | 1: enabled>
  show/hide the port surface.

- setLayerThickness <groupId> <value>
  set the layer thickness value for a given group for the boundary layer

- setLayerCellSize <groupId> <value>
  set the cell size value for the boundary layer for a given group

- getLayerThickness <groupId>
  get the value of the boundary layer thickness for a given group

- getLayerCellSize <groupId>
  get the cell size of the boundary layer for a given group

- preserveThinStructure <groupId> <0|1>
  activate/deactivate the thin structure for a given group

- setFastMeshingState <0: disabled | 1: enabled>
  activate/deactivate the fast meshing

- "theBoundaryController" followed by the commands below to set boundary conditions

  - getAvailableBoundaryIds
    get the available boundary IDs for the selected software;

  - getCurrentBoundaryId
    get the selected boundary ID

- getBoundaryMethod
  get the current boundary method

- setBoundaryMethod <0: Plan | 1: Material | 2: Equation>
  set the boundary method

- createBoundaryPlan <bcName> <bcType> <0: XMIN, 1: XMAX, 2: YMIN, 3: YMAX, 4: ZMIN, 5: ZMAX> <materialName> <double:[0-100]>
  create a boundary contidtion with plan method. The arguments are: the boundary name, the boundary condition type, the plane of bounding box, the material to apply the boundary condition, and the tolerance.

- createBoundaryContact <bcName> <bcType> <materialName> <materialName>
  create a boundary condition with the material contact method. The arguments are: the boundary name, the boundary type, the first material and the second material

- createBoundaryEquation <bcName> <bcType> <materialName> <equation>
  create a boundary condition with equation method. The arguments are: the boundary name, the boundary type, the material and the equation.

- "theDisplayController" followed by the commands below to configure the display:

  - setDrawStyle <drawyStyle>
    draw style selection

  - enableClippingPlane <enable>
    activate/deactivate the clipping plane

  - setClippingPlaneOrigin <posX> <posY> <posZ>
    set the origin of the plane

  - setClippingPlaneNormal <vecX> <vecY> <vecZ>
    set the normal of the plane

- "theMeshExporter" followed by the commands below to configure the export of the mesh:

  - setMesh <mesh>
    select the mesh

  - setSoftware <software id>
    select export format

  - getMesh
    get the name of the selected mesh

- `getAvailableSoftwares`
  get the list of softwares

11.1.14 Progress Bar Indications

The progress bar outputs various messages during the generation of the mesh. Some of those steps may require a long time. You may encounter long delays with the following:

- Initialization triangulation: meshing engine initialization followed by protecting balls settings. This step might take a long time if you have a topologically complex mesh. Complex topology will be preserved by setting a high number of protecting balls.
- Meshing: Actual meshing/refinement.
- Optimization: optimization of the mesh points
- Creating output: converting internal mesh structure to HxTetraGrid

When fast meshing is disabled:

- The first messages comes from the internal surface generation based on the Generate Surface module (Smoothing labels, Computing triangulation). If those messages take a long time, the internal surface used for constraining the meshing will be very large. You can alternatively generate your own simplified surface (making sure that it is watertight, and that there is no autointersection between the different materials) and pass it on to the meshing room via TCL command ("theMeshingTabController" enableSurfaceSelection 1 or "theMeshingTabController" setSurfaceData yourSurface).
- Computing: Converting Avizo surface to internal structure for meshing
- Generating Meshing Domain: Building constraints based on input surface

11.2 Meshing Workroom Tutorial

The following tutorial, like the Meshing Workroom it demonstrates, requires an Avizo XWind license and is only available on Microsoft Windows.

In this step-by-step tutorial, you will learn how to use *Meshing Workroom* high-end pre-processing capabilities to create simulation inputs. The *Meshing Workroom* allows you to compute a tetrahedral mesh, assign boundary conditions to it, and export it to various CFD or FEA solver's file formats such as Fluent, COMSOL Multiphysics (c), etc. The meshing engine used in the workroom is based on Delaunay refinement (while Avizo legacy meshing engine is based on advancing front technique).

The tutorial illustrates this typical workflow with the following sections:

- *First mesh generation and inspection*

- *Global mesh refinement*
- *Mesh refinement of groups of materials*
- *Advanced mesh refinement*
- *Boundary layer refinement*
- *Optimization of mesh quality*
- *Color mapping of material properties*
- *Boundary conditions and export to CFD/FEA solvers*

To follow this tutorial, you should be familiar with the basic concepts of Avizo such as data import, interaction with the 3D viewer, etc. All these topics are discussed in Avizo*Getting Started* chapter. You should also be familiar with the workflow of generating label images, which are detailed in the *Visualizing and Processing 2D and 3D Images* chapter, and more specifically:

- Prepare data for segmentation: *About Image Filtering*
- Segment data: *Segmentation of 3D Images*
- Edit segmented data: *Volume Edit*, *Segmentation Workroom*

All illustrations using the corn kernel dataset are with the kind courtesy of, Pawan S Takhar, University of Illinois at Urbana-Champaign.

Note: In the following tutorial, units are not activated. All parameter sizes (i.e., measurement, cell size, ...) should be given within the current unit if units are activated.

11.2.1 First Mesh Generation and Inspection

In the Project Workroom, click **Open Data** and choose to load *data/tutorials/meshing/KernelCorn.labels.am*. Activate the *Meshing Workroom* by clicking the Meshing tab (Figure 11.30) in the workroom toolbar.

Set **Label Data** to KernelCorn.labels.am.

Click **CREATE** at the lower left of the room to launch mesh generation with the default parameters.

A first mesh is generated and automatically set in the *Mesh* input field.

Note: Once you have created a first mesh for a given label field, you can either create a new mesh for this label field by clicking **CREATE**, or replace (overwrite) the current mesh by clicking **REPLACE**.

The generated mesh has been automatically displayed in the main viewer.

You can go back to the Project Workroom at any time to visualize and customize the display of the mesh. The generated mesh **KernelCorn.labels.tetra** has been created in the *Project View* and is tight to the label input. Use *Tetra Grid View* module to display the mesh.

Figure 11.30: Meshing Tab

You can also find information about the size of the mesh in terms of nodes, triangles, and tetrahedrons in the *Properties Area* of the mesh object.

Go back to the *Meshing Workroom*.

By default the *Fast Meshing* mode is activated in the Meshing tab. This mode allows for fast prototyping the meshing of a given label. It is fast, because the meshing is not constrained by geometry. The resulting mesh is fast to build, but is missing precision at boundaries between materials.

Uncheck the *Fast Meshing* option and click *CREATE*.

When the *Fast Meshing* option is not used, the meshing engine is constrained to respect interfaces at boundaries between labels. Those interfaces are defined by surfaces that are automatically generated using an underlying *Generate Surface* module with the constrained mode and the fit to edge options turned on.

The generated mesh KernelCorn.labels2.tetra has been set in the *Mesh* input field (Figure 11.31).

You can compare the two generated meshes by switching from one to the other in the *Mesh* combo list.

You will notice how the contours of the second mesh (KernelCorn.labels2.tetra) are clearer than the contours of the first one (KernelCorn.labels.tetra).

Figure 11.31: Mesh Selection
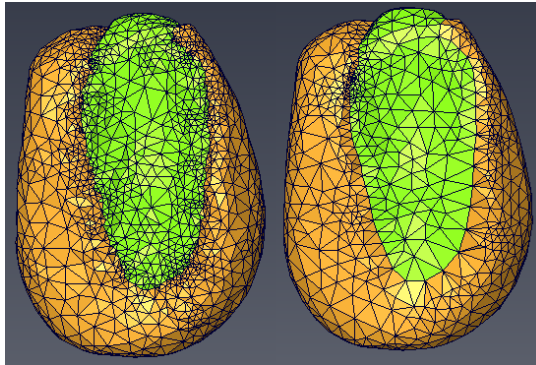
Figure 11.32: Fast Meshing Option turned On (left) and turned Off (right)

To push further the visual inspection of the mesh, the visibility of its materials can be turned on and off to inspect the different parts, and more specifically, the interior elements of the mesh. The visibility of each individual material can be accessed from the Meshing tab in the Materials section.

In the Materials section, hide the Germ material by unchecking it (Figure 11.33).
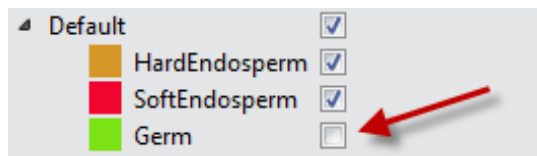
Figure 11.33: Hide Germ Material

In the viewer, compare the contours of the Germ (Figure 11.34).

11.2.2   Global Mesh Refinement

The *Mesh Quality* slider provides five quality levels to mesh the input label field from low to high. Each quality level automatically sets a number of meshing options. You can check the effect of the different quality levels on the meshing engine settings by looking how the values are set in the greyed out area below the slider (Figure 11.35).
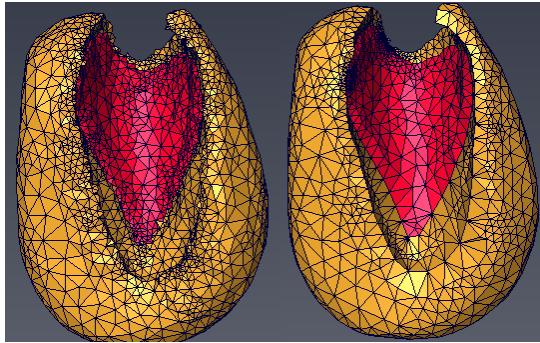
Figure 11.34: Germ Material Hidden with Fast Meshing Option turned On (left) and turned Off (right)
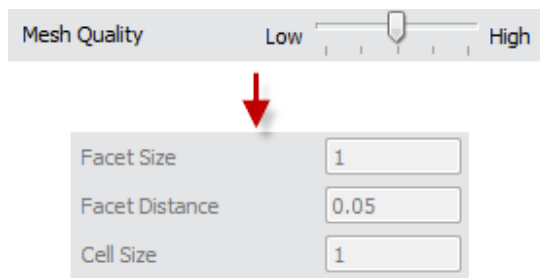
Figure 11.35: Mesh Quality Slider

We will come back to these parameters later in the tutorial. You can also refer to the *Meshing Workroom* documentation for details about the parameters.

Select **KernelCorn.labels.tetra** in the *Mesh* field.

Move the *Mesh Quality* slider to **Low** and click **REPLACE**.

KernelCorn.labels.tetra is replaced with a coarse mesh.

Select **KernelCorn.labels2.tetra** in the *Mesh* field.

Move the *Mesh Quality* slider to **High** and click **REPLACE**.

KernelCorn.labels2.tetra is replaced with a refined mesh.

Use the *Mesh* field for a first comparison of the two generated meshes (Figure 11.36).

To inspect interior elements, a clipping plane dedicated to the visualization of tetrahedral mesh is available from the main toolbar. The clipping plane automatically clips any tetrahedron that does not intersect the plane and is on the clipped side.

To activate clipping, turn on **Clip** (Figure 11.37).
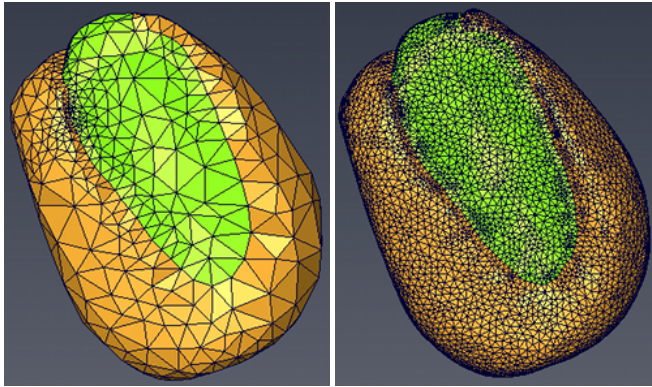
Figure 11.36: Meshes with Low (left) and High (right) Quality
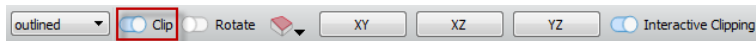
Figure 11.37: Clip Option

Again, use the *Mesh* field to switch from one mesh to the other and compare the refinement inside the meshes (Figure 11.38).
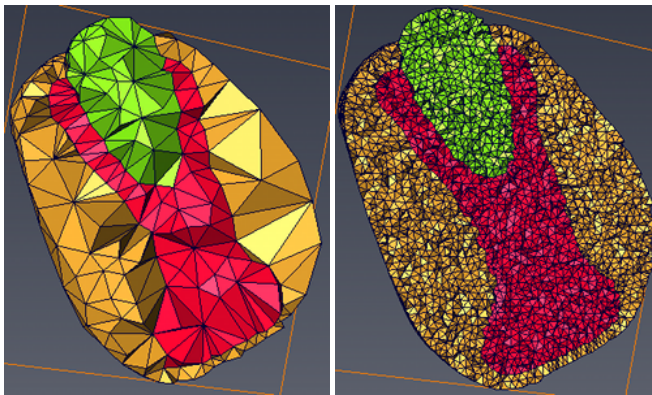
Figure 11.38: Clipped Meshes with Low (left) and High (right) Quality

In Interaction mode (press **Esc**), pick the plane with the left button of the mouse and move the mouse while keeping the left button pressed. This will translate the plane along its normal. Various options are available to manipulate the clipping planes. See more details *manipulating the clipping plane* in the *Meshing Workroom* help section.

11.2.3 Mesh Refinement of Groups of Materials

By default the quality settings are applied to all the materials, which are gathered in the *Default* group of the *Materials* section.

New groups can be added to assign different quality settings to the different materials. When selecting a group or any material in a group, the mesh quality settings are then applied to this specific group.

Select **KernelCorn.labels2.tetra** in the *Mesh* field and set the *Mesh Quality* to the **Medium** value.

Click **Add Group**.

Drag and drop the **SoftEndosperm** material in the newly created group (Figure 11.39).
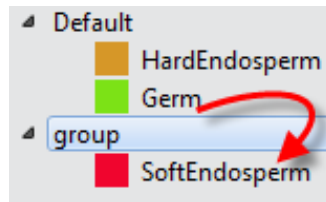
Figure 11.39: Add a Material to a Group

Set the *Mesh Quality* to **High**.

Click **Add Group** again and drag and drop the **Germ** material in the newly created group *group2*.

Remove *group2* by hovering the mouse on its line and clicking the garbage icon. You will notice that the material it contained goes back to the default group. Materials, as well as the default group, cannot be removed.

Click **REPLACE**.

Click **YZ** button of the clipping plane toolbar and move the camera to get a different view inside the mesh. Notice the different mesh qualities upon the materials used as shown in Figure 11.40.

11.2.4 Advanced Mesh Refinement

Switch to the **Advanced** mode of the Meshing tab.

Switching to **Advanced** mode makes advanced parameters available for edition.

The combo boxes in the *Scalar Fields* section of the Meshing tab are now enabled, as well as the advanced parameters located in the *Materials* section. The *Mesh Quality* slider is disabled.

Each of the three following parameters can be set to a constant value or to a varying scalar field:

- The *Facet Size* parameter controls the size of surface facets (i.e., the size of the triangles of surface facets). It provides an upper boundary for the radii of a ball circumscribing the surface
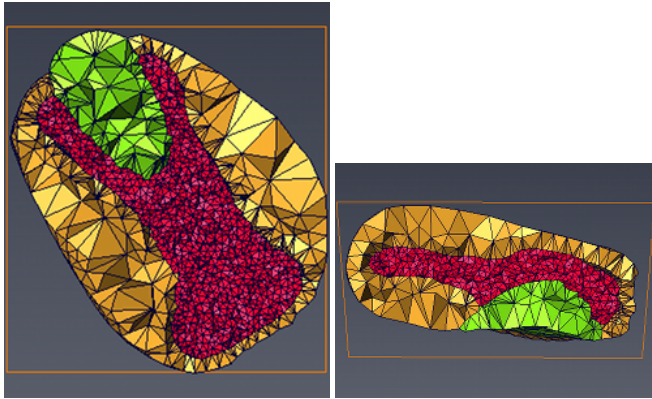
Figure 11.40: XY and YZ Clipped Mesh (One Material in High Quality, other in Low)

facet and centered on the surface patch.

- The *Facet Distance* parameter controls the approximation error of boundary and subdivision surfaces. The meshing engine will create small elements close to curved surfaces and large elements away from the surfaces.

- The *Cell Size* parameter controls the size of mesh tetrahedra. It provides an upper bound on the circumradii of the tetrahedra.

Select the **SoftEndosperm** material or its group.

Change the *Cell Size* to 0.1.

Click **CREATE**. A new mesh KernelCorn.labels3.tetra is generated.

With the *Mesh* list, switch from **KernelCorn.labels2.tetra** to **KernelCorn.labels3.tetra** and observe how the cell size of the SoftEndosperm material has been divided by two (Figure 11.41).
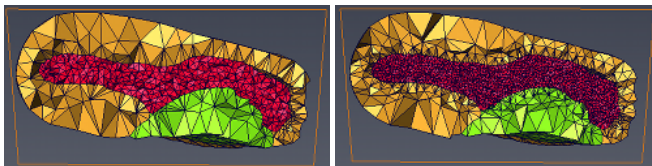
Figure 11.41: SoftEndosperm (red) with Cell Size 0.2 (left) and 0.1 (right)

Go to the Project Workroom and remove all three tetra meshes attached to the label field. Go back to the *Meshing Workroom*.

Disable the *Advanced* mode. Set the *Mesh quality* of all groups to **High** and click **CREATE**.

Switch back to *Advanced* mode and set the *Facet distance* for the SoftEndosperm group to 0.005. Click **CREATE**.

With the *Mesh* list, switch from KernelCorn.labels.tetra to KernelCorn.labels2.tetra back and forth and observe how the mesh is more refined where the SoftEndosperm surface is curved. (Figure 11.42)
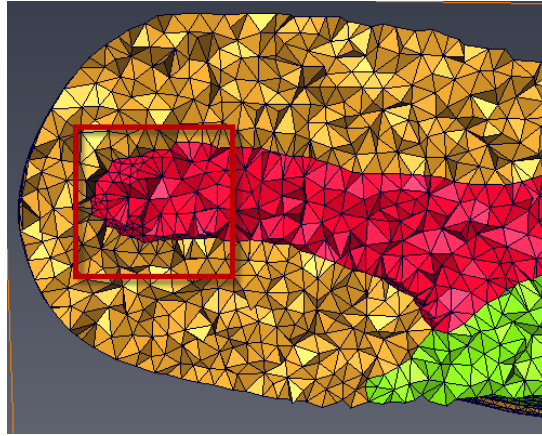
Figure 11.42: Refined mesh of the SoftEndosperm (red) material

Scalar fields can also be used to customize the refinement of a specific area by controlling the value of an advanced parameter. The scalar field will control the parameter globally (i.e., for the whole mesh) and not for a given group. You can, however, mix scalar field controlling on the whole mesh for a given parameter, and set constant values per group for another parameter. Be aware that the scalar field must have the same dimension as the label field.

Here is an example of how you can create such a scalar field. You can skip this part and retrieve the scalar field in the tutorial folder: *data/tutorials/meshing/KernelCornField.am*.

Go back to the Project Workroom. Connect a *Distance Map* module to the label field Kernel-Corn.labels.am. The Distance Map module assigns to each voxel a value depending on the distance to the nearest object boundary. The boundary voxels of the object are assigned a value of zero and the value increases in the object as the distance increases.

Click **Apply**.

Attach an *Interactive Thresholding* module to the *Distance Map* result. Set the *Intensity Range* to 1-15. Click **Apply**. A layer of voxels is isolated on the boundary of the whole label field (i.e., regardless of the materials).

Attach a second *Interactive Thresholding* module to the *Distance Map* result. Set the *Intensity Range* to 15-70. Click **Apply**. This isolates the rest of the voxels belonging to the label field.

Attach an *Arithmetic* module to the result of the first *Interactive Thresholding* module. Connect the

Input B of the *Arithmetic* module to the result of the second *Interactive Thresholding* module. Select **1 value (scalar)** in the *Result Channels* field. Enter the expression 0.05*(1+A+39*B) in the *Expression* field. Click **Apply** and see result in Figure 11.43.
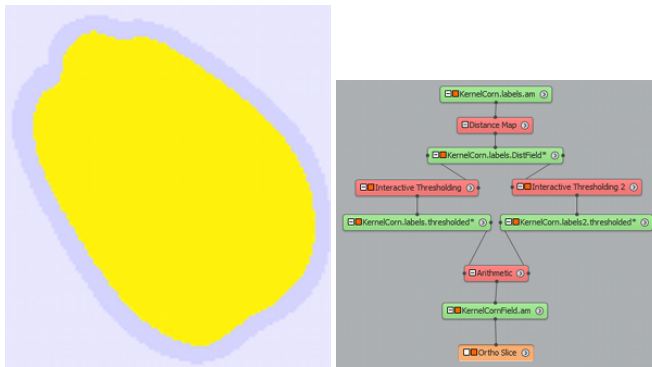
Figure 11.43: Cell Size Scalar Field Generation

The resulting scalar field takes value 0.05 outside of the object (0.0 would aso be correct), 0.1 in a layer of voxels close to the boundary inside the object, and 2 in the rest of the voxels inside the object.

In case of any problems or uncertainties you can find the network generating the scalar field in the tutorial folder: data/tutorials/meshing/CreateKernelCornField.hx.

Go back to the *Meshing Workroom*.

Set the *Facet Distance* back to 0.01. Set the cell size scalar field to **Result** in the *Scalar Fields* section. Notice that in the *Materials* section, the cell size port has been greyed out and replaced with *Managed by scalar field*.

Click **REPLACE** and see result in Figure 11.44.
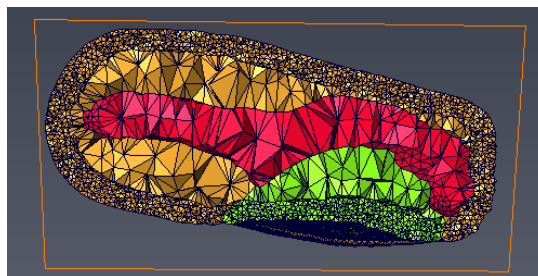
Figure 11.44: Mesh with computed Cell Size Scalar Field

You can observe how the mesh is refined in a boundary layer for the whole object regardless of the

material.

11.2.5   Boundary Layer Refinement

The option allows defining, by group, a boundary layer where the mesh can be refined.

Go to the Project Workroom and remove all tetra meshes attached to the label field. Go back to the Meshing Workroom.

Uncheck **Advanced** to switch back to the non-advanced mode.

Create a mesh with *Mesh Quality* set to *Medium* for all materials.

Visualize the clipped mesh in XY plan.

Create a group containing only the HardEndosperm material. Set the *Mesh Quality* to *Medium* for this group and, in the *Boundary layer* field, set the *Layer Thickness* to 0.4 and the *Layer Cell Size* to 0.2.

Note: to measure the layer thickness which is suitable for your data, you may go back to the project room, attach an orthoslice to your label field and measure a given thickness using a *line measure*.

Press *REPLACE*. KernelCorn.labels.tetra is updated.

Set the same boundary layer parameters for all groups of materials.

Press *CREATE*. KernelCorn.labels2.tetra is created.

Observe the mesh refinement in the two meshes that have just been generated.
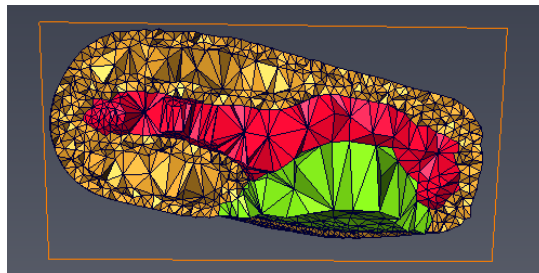
Figure 11.45: Boundary layer refinement for HardEndosperm material only

11.2.6   Optimization of Mesh Quality

You can optimize the tetrahedron nodes location to reduce the amount of sliver in the mesh. A sliver tetrahedron is formed by placing its four vertices near the equator of its circumsphere (e.g., flat tetrahedron). Slivers may slow down the convergence of numerical simulation. This means that the less sliver in a mesh, the better it is.
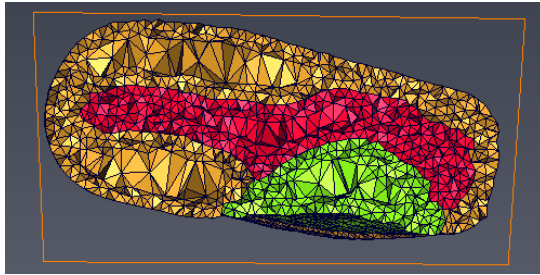
Figure 11.46: Boundary layer refinement for all materials

The optimize mesh option aims at reducing the amount of sliver by perturbing them through random vertex relocation. The mesh generation may be a bit longer if the option is activated. The option is available whether the *Fast Meshing* option is on or not.

Keep the previous settings and in the *Global Parameters* Section, select the *Optimize Mesh* checkbox.

Select **KernelCorn.labels.tetra** in the *Mesh* field and click **REPLACE**.

Now, KernelCorn.labels.tetra and KernelCorn.labels2.tetra are the optimized and non-optimized versions of a mesh generated with the same parameters. You can now compare their quality in terms of number of slivers.

1) Go to the Project Workroom.
2) Attach a *Tetra Grid View* module to each mesh.
3) For both, click **Show/Hide** then **Remove** in the *Buffer* port to select all tetrahedra and then hide them all in the viewer.
4) Select **KernelCorn.labels2.tetra** (non-optimized mesh) and click the **Grid Editor** button in the properties area. In the **Selector port**, **Tetra Quality** is selected. Click **Select**. This selects tetrahedra according to the quality measure defined in the next port: *R <0.02* where R is the ratio of the radii of the inscribed and circumscribed spheres. R reaches its optimal (i.e., maximal) value 1/3 for an equilateral tetrahedron. Slivers are detected by a small value of R.

The viewer now displays the slivers in the non-optimized mesh (Figure 11.47). You can also retrieve their number from the Console (click the application's lower-right corner button).

Toggle off the visibility of the *Tetra Grid View* attached to KernelCorn.labels2.tetra.

Repeat the previous actions on the *Grid Editor* of the optimized mesh KernelCorn.labels.tetra.

You can see in the viewer that there are less slivers in the optimized mesh. The Console displays the number: you can observe that there are much less selected tetra.

It is possible to further improve the quality of the mesh using the *Grid Editor*. In the Modifier port, select *Repair Bad Tetras*. Click **Modify**. Press *Select* again in the *Selector* port. In the Console, you will notice that the number of slivers has further decreased.
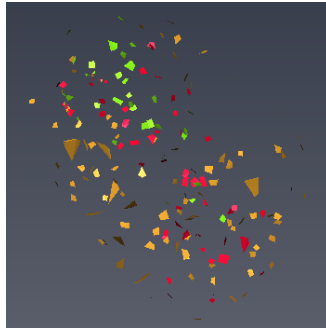
Figure 11.47: Slivers for the Non-optimized Mesh

For the purpose of next section, remove both Tetra Grid View modules.

11.2.7 Color Mapping of Material Properties

Scalar fields can be color mapped on the mesh to visualize material properties. For example, you will now visualize curvature information.

In the Project Workroom, attach an *Image Curvature* module to the label field. In the *Interpretation* port, select **3D** and click **Apply**.

Attach a Tetra Grid View module to KernelCorn.labels.tetra. Set the maximal local curvature Kernel-Corn.labels.curvature as color field of the Tetra Grid View. Set the *Colormap* to physics.icol and the range to [-0.8,1].

To discretize a scalar field per cell or per node of a tetra mesh, you can use the *Arithmetic* module connected to both the mesh and the scalar field.

Attach an Arithmetic module to the mesh KernelCorn.labels.tetra. Connect the *Input B* of the *Arithmetic* module to the curvature field. Set the *Expression* to B.

Result Location field can be set to on *Nodes* or on *Cell Center*. Try both.

Attach the Tetra Grid View color field to the results of the Arithmetic modules in order to map curvature field to the mesh (Figure 11.48)

11.2.8 Boundary Conditions and Export to CFD/FEA Solvers

Go back to the *Meshing Workroom*.

The formats supported for export to CFD/FEA solvers are listed at the top of the workroom in the *Type* list (Figure 11.49).

Select *FLUENT* format.
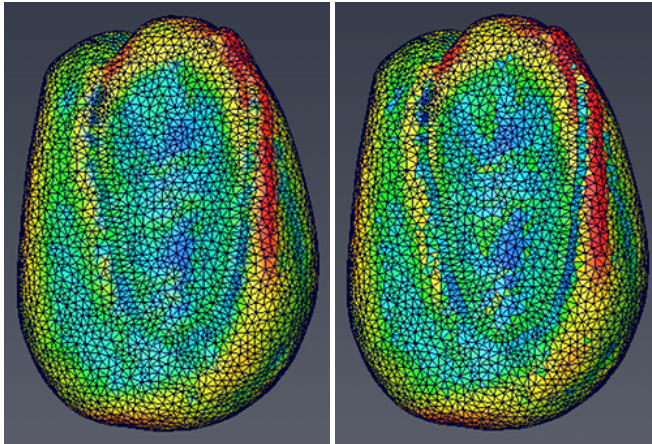
Figure 11.48: Curvature Field Interpolated on Nodes (left) and on Cells (right)

Figure 11.49: Export buttons

Open the *BOUNDARY* tab to set boundary conditions.

Three modes are available to set boundary conditions:

- Using plans, for example to define a flow rate input on a whole face of the bounding box of a sample,
- Using the contact surface between two materials,
- Defining a surface by its equation.

Select *Contact* mode.

In the left material list, select HardEndosperm and in the right material list, select Germ (Figure 11.50).
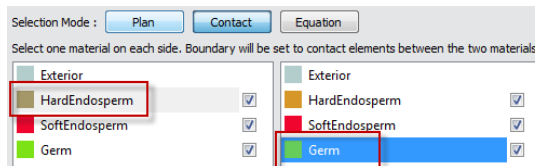
Figure 11.50: Material Contact Selection

You can visualize the contact surface by toggling off the visibility of the SoftEndosperm and Germ materials (Figure 11.51).
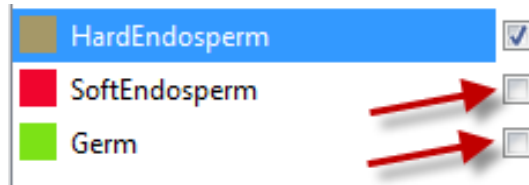
Figure 11.51: Material Contact Visualisation

The contact triangles are highlighted in red. You can also visualize it by enabling a clipping plane (Figure 11.52).
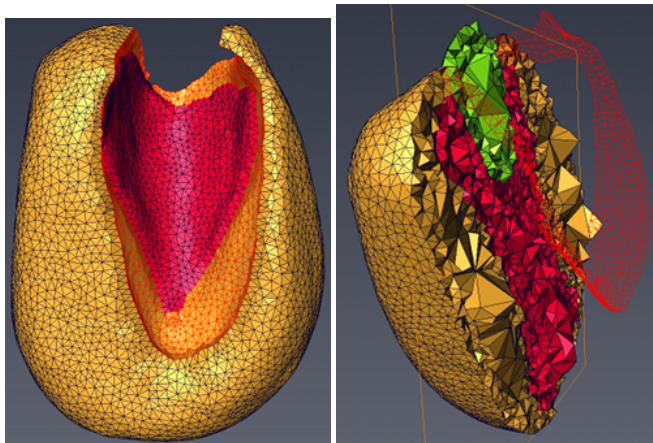
Figure 11.52: Contact Surface without (left) and with (right) Clipping.

Set the *Type* to *Interface* and name it HardEndo-Germ-contact.

Click **Add**.

The contact boundary is added to the *results list*. The boundary type, as well as the number of triangle elements, is displayed. It is possible to highlight the boundaries from the result list at any time: simply select it from the list. It will be highlighted in green in the viewer. It is also possible to add elements to an existing boundary: select its name from the name combo box and press add, once new elements are selected using one of the three methods above.

Once you have finished defining the boundary conditions you want to assign to the mesh, press the Export button. The *Export* dialog will pop up. Set the file name and location and press *Save*.

11.3   Avizo XWind Extension Measurements

This section will give you an overview of the measurement features provided in the Avizo XWind Extension.

- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Pressure` scalar field.
- Connect a *Boundary View* to the model.
- Unselect everything except `Wall` in the **Boundary types** port.
- Set the coloring to *Data Mapping* and use `Pressure` as the colorfield.
- Select *node values* in the *Value Mapping* port.
- Hide the *Bounding Box*.

11.3.1   3D measurements

You can access measuring tools via the *View / Measuring* menu or via the measuring tool button ![2D button] (and its pulldown menu - click on the little arrow) at the top of the viewer.

- Select *Line* in the pulldown menu of the measuring tool button.
- Select *Measuring* in the *View* menu.

You now have a *Measurement* object in the Display folder of the Project View. This module provides access to two-dimensional and three-dimensional measuring tools.

Line measurement

We will measure the leading edge of the wing. A line measurement (Line) is already selected.

- In the 3D viewer, click on one end of the leading edge of the wing.
  Notice that cursor changes to indicate when a valid object can be selected.
- Click on the other end of the wing edge.
- To adjust the position of a measurement line,
  select it in the Properties area, then click on one of its red handles and drag it to a new location
  or use the text ports **Point 0** and **Point 1** to change the position.
- Do the same on the side edge of the wing.
  Tip: You may need to reposition the camera to select measurement points. As usual you can press the [ESC] key to toggle between interactive mode and trackball mode or hold down the [Alt] key to temporarily switch to trackball mode.

You can measure that the wing has a leading edge of approximately 4.44 meters and a side edge of approximately 1.55 meters.

Figure 11.53: Measuring the wing size of the YF-17 aircraft.

3D angle measurement

- In the Properties area of the *Measurement* module, click on the "eye" icons of the two lines to hide them in the 3D viewer.
- In the **Add** port, click the *Angle* button.
- In the 3D viewer, click on the intersection of the attack edge of the wing and the fuselage.
- Click on the other end of the attack edge (intersection with the side edge).
- Click on the other end of the side edge.

You can measure that the angle is approximately 116 degrees.

11.3.2   Histograms

The *Histogram* module computes the histogram of a scalar field in 3D cells. We will use it on the `Pressure` scalar field.

- Right-click on the `Pressure` and select *Histogram* in the *Measure And Analyze* menu.
- Click Apply in the *Histogram* Properties area.

A window pops up, that contains a histogram in logarithmic scaling. The mean value (approx. 1849 Pa) and the standard deviation (approx. 23187 Pa) of the `Pressure` field are displayed in the Properties
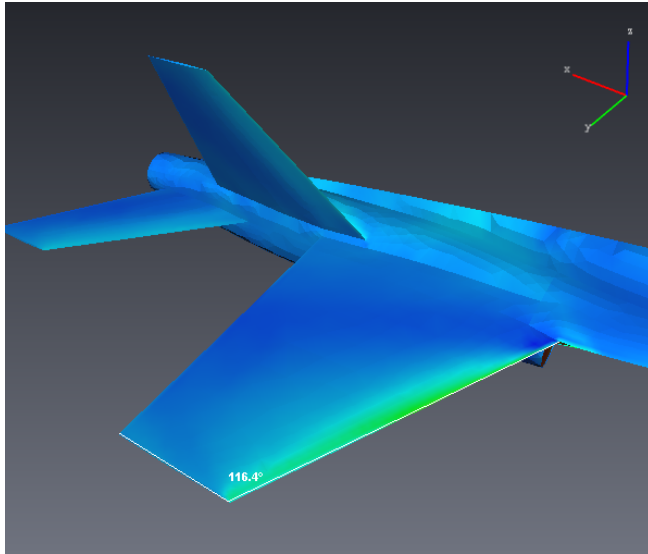
Figure 11.54: Angle measurement.

area.

- Set the **Range** minimum value to 0.
- Activate the **Threshold** and set it to 100000.
- Activate the **Tindex** and set it to 50.
- Click Apply.
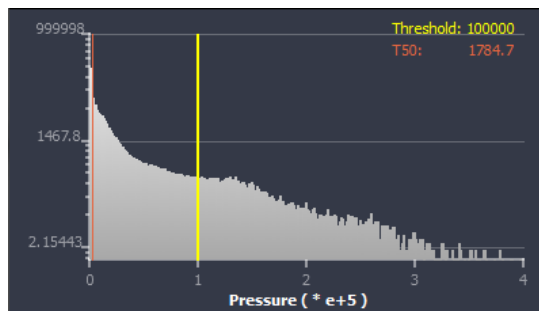
Figure 11.55: Histogram of pressure distribution.

What we can learn is that:

- for all cells where the pressure is in the new range, the mean value is approximately 10647 Pa and the standard deviation is approximately 26442 Pa,

- in this same range, 2.44 percent of the cells have a pressure greater than 100000 Pa,

- in this same range, 50 percent of the cells have a pressure lower than 1784 Pa (and 50 percent greater).

The histogram has been updated.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_histo.hx.

11.3.3   Data probing

The three data probing modules *Point Probe*, *Line Probe*, and *Spline Probe* are used to inspect scalar or vector data fields. The probes are taken at a point (*Point Probe*) or along a line (*Line Probe* and *Spline Probe*) which may be arbitrarily placed.

Probing along a spline

We will use the *Spline Probe* to plot the pressure around the wing of the aircraft. First we have to position properly the four control points of the spline.

- Right-click on `Pressure`.
- In the *Measure And Analyze* menu, select *Spline Probe*.

To position the control points within the bounding box of the given geometry you can either type in the coordinates in the **Points** port (see below) or you can move the points dragger interactively with the mouse. (You may have to zoom out to see the points dragger.)

- In the **Points** port, the coordinates of the first control point are displayed. Change them to 4, 2, 0.
- In the **Points** port, use the spin box to select the second point and set its coordinates to 0, 2, 0.
- Select the third point and set its coordinates to 0, 2, 0.3.
- Select the fourth point and set its coordinates to 4, 2, 0.3.
- You might want to hide the points and the dragger using the *options* submenu of the **Points** port.
- Click the *Show* button in the **Plot** port.

A plot window appears where the sampled pressure values are plotted against the length of the probe line. In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_splineprobe.hx.

Probing along a surface path

For probing purposes, it is often useful to have tools to define specific lines on a surface. The *Surface*
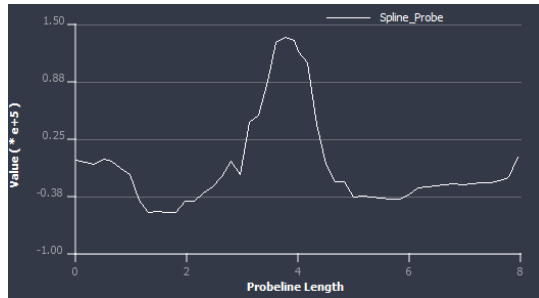
Figure 11.56: Pressure values against the spline probe line length.

Path Editor and the *Surface Intersector* module are designed to this end.

The *Surface Path Editor* allows creating paths on surfaces. Paths can be useful to cut surfaces, define regions or features of a surface, probe, etc. The editor can be accessed from the Properties area of a *Surface Path Set* by clicking on the editor button [icon]. Two types of editor are then provided:

- the Generic Path Editor allows defining paths arbitrarily across the surface mesh,
- the Vertex Path Editor allows defining paths only along the surface mesh edges.

Note that the Vertex Path Editor can be accessed directly from the *Mesure And Analyze* submenu of a surface (entry named *Create Surface Vertex Path*).

The *Surface Intersector* module intersects two surfaces, computes a path along the intersection and attaches it to each of the surfaces.

- Remove all objects from the Project View (use [Ctrl+N] or right click in the Project View and select *Remove All Objects*).
- Open `fan-0070.cas` from the `tutorials/cfd-fea-advanced/fan` folder.
- Load the `Pressure` scalar field.
- Hide the *Bounding Box*.
- Connect a *Boundary View* to the model.
- Unselect everything except `wall-1` in the **Boundaries** port and create the surface from the **Create surface** port.

We will plot the `Pressure` along a radial line section of the fan surface. We have to create a cylindrical surface first, in order to intersect it with the fan and then get the intersection line.

- Right click on the surface `fan-0070.surf` and create a *Surface Intersector* from the *Compute* submenu.

In the *Surface Intersector* Properties area, the second surface still has to be set. We will use the *Parametric Surface* module to create the intersecting surface we need, available from *Project / Create Object...* (*Surfaces And Grids* submenu).

- Create a *Parametric Surface*. A default plane is created.
- For **U**, set *min* to -0.02, *step* to 0.0005, *max* to 0.01.
- For **V**, set *min* to 1, *step* to 0.0005, *max* to 2.
- Set **X** to u, **Y** to `0.12*sin(v)` and **Z** to `0.12*cos(v)`.
- Click on the *more options* button in the **Draw style** port and select *Create surface*. `Parametric-Surface.surf` is added to the Project View.
- Set `Parametric-Surface.surf` as the second surface of the *Surface Intersector* and press Apply.

Two paths along the intersection are created, one attached to each of the surfaces.

- Hide the *Parametric Surface* and connect a *Line Set View* display module to `IntersectionPath2`. The path is displayed on the fan surface.
- In the *Measure And Analyze* submenu of `Pressure`, select *Line Set Probe*.
- Attach the *Line Set Probe* to `IntersectionPath2` in the *Line set* port and press the *Show* button.

A window displaying the `Pressure` along the line probe appears. We will improve the display.

- For *X-Axis*, choose the z coordinate.
- In the *Edit* menu, select *Edit Objects*.
- In the *axis* section, deselect the *Auto* option that adjusts the range to the X range and set it to [-0.025, 0.04].
- Change the Y label to "Pressure".
- In the *LineSetProbe_001* section, change the *Draw style* to *Marker*.
- Change the markers shape (e.g. to dots) and color.
- Change the label to "Radial section: 0.12m".
- Press OK.

11.4   Getting Started with reading and visualizing CAE/CFD data

By following this step-by-step tutorial, you will learn the basics of reading and visualizing CAE/CFD data with Avizo XWind Extension.

Note: This tutorial shows the results on a voluntary small and lightweight sub-sampled case study. The artifacts may be visible on some rendering methods due to this low-quality model and not due to
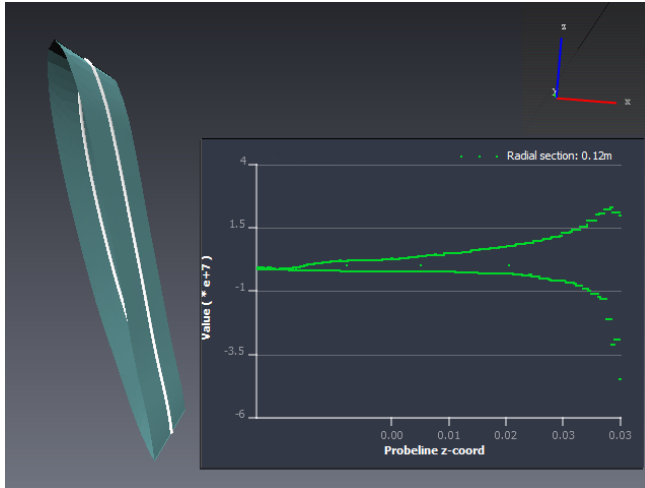
Figure 11.57: Pressure values along the radial 0.12m line section of the fan surface.

Avizo XWind Extension capabilities.

11.4.1   User interface short overview

Avizo user interface is divided into three major regions: The Project View (1) contains folders where data and modules will appear. The Properties area (2) displays interface elements (ports) associated with Avizo objects. The 3D viewer window (3) displays visualization results (see Figure 11.59).

Note: If your *Project View* is displayed as a graph, you can switch to the *Project Tree View* by clicking on the *Project Tree View* button. You can easily go back to the *Project Graph View* whenever you want (see Figure 11.58). These buttons are also available in the *Project Menu*.

Figure 11.58: 1: Switch to the *Project Tree View* - 2: Go back to the *Project Graph View*

If you click on Help, you can browse the *User's Guide*. When an object (data, module...) is selected in the *Project Tree View*, information about it is displayed in the *Properties* area. A click on the question tag "?" in the upper right side of the area will take you to the contextual help page for the active object.

For more information about the user interface and especially the *Viewer* window, please refer to the

Program Description and to its *Viewer Window* subsection.
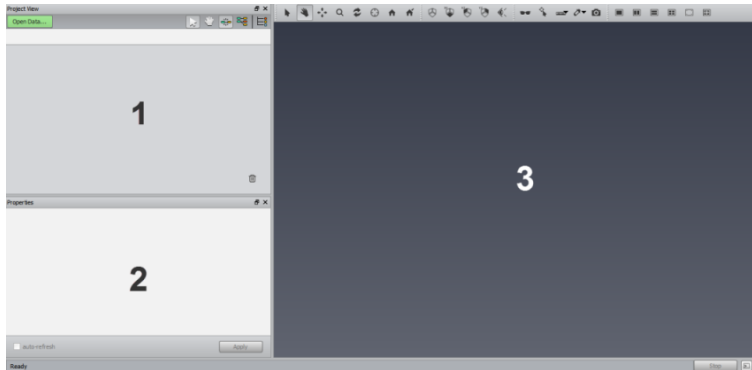
Figure 11.59: Avizo user interface

The Project Tree View

In the following tutorials, we will use the *Project Tree View* to manipulate objects.

The results of your CFD/CAE simulations and computations are stored in the *Data* folder. These results are composed of one or more models and data sets (see Figure 11.60).
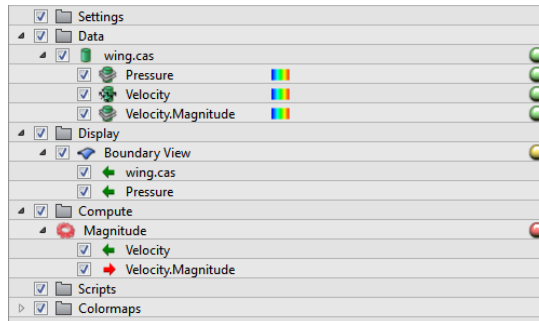
Figure 11.60: Example of a data set and its connected modules in the Project Tree View

A model contains:

- the mesh of the domain under study, with 2D or 3D cells depending on the dimension of the model,
- the different regions the domain might be composed of (e.g. the rotor and the stator of a pump),
- the boundaries which are the physical limits of the domain,
- the material the domain is made of.

A data set attached to a model contains one or more scalar, vector or tensor fields defined on the mesh and/or the boundaries of the model. These are the physical quantities that have been computed during the simulation and need to be visualized and analyzed.

Display modules will be listed in the *Display* folder and compute modules in the *Compute* folder. Shortcuts to the input and output data of a module appear below the module, with green and red arrows indicating inputs and outputs respectively.

In the *Colormaps* folder you will find the default colormaps and also any colormaps you have loaded.

To get the same project tree view organization than in the tutorial screenshots, go to *Edit / Preferences / Layout* and select *Group by display/compute/data in tree view*). You can alternatively use the Avizo Lite Edition version of the *Project Tree View* (without the "Group by display/compute/data in tree view" option) or switch to the *Project Graph View* from the *Project Menu*.

In the remaining of the tutorial, the *Project Tree View* will be named only *Project View* for easing the reading.

11.4.2   Reading data

Avizo XWind Extension reads a wide range of CFD/CAE formats, including:

- *Abaqus*
- *ANSYS*
- *CGNS*
- *Ensight*
- *Fluent/UNS*
- *NASA/Plot3D*
- *Nastran Bulk Data*
- *Nastran Output2*
- *SDRC/IDEAS Universal*
- *STAR-CCM*
- *Tecplot*

Please refer to the file formats index of the *User's Guide* for details.

We will start this tutorial by loading a Fluent data set:

- Click on *Open Data* in the *Project View*.
- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder (do not select the .dat file, Avizo XWind Extension will retrieve the .dat in the folder or will ask you in which folder to find it).

The Datasets selector pops up. Many scalar and vector fields can be attached to a given model and it might be very memory-consuming to load them all. It can also be space-consuming in the project tree view and make it difficult to read. The Datasets selector allows you to load only a part of the solution and, if necessary, to unload some data and load additional data later (see Figure 11.61).
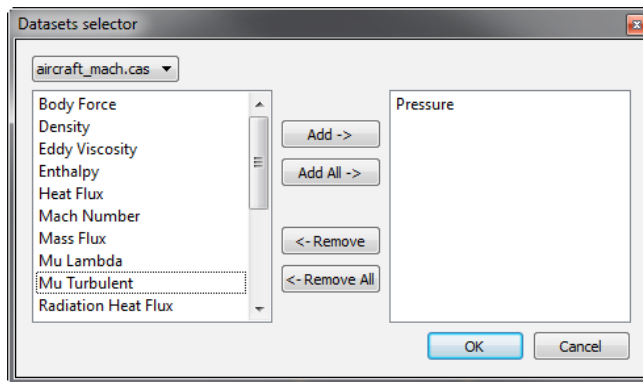
Figure 11.61: Datasets selector.

Select `Pressure` in the data column and click Add-> then OK. The `Pressure` now appears in the Project View under the model it is attached to and its colormap is displayed.

In the 3D viewer you can see the *Bounding Box* of the model. This display module is connected by default to the model when you load it.

- Remove the bounding box by right-clicking on the module in the Project View and selecting *Bounding Box / Remove Object* (or press on [Del]).

We do this because the bounding box encloses the entire data set and we will initially focus on a specific region of the model. Removing (or simply hiding) the bounding box makes it more convenient to "zoom in" on this smaller region.

If you want to load other data from the same model, you can access the Datasets selector at any time by clicking on its button in the Properties area of the model (`aircraft_mach.cas`).

11.4.3 Getting started

We will now make our first visualization.

- In the *Project View*, select the model *data/tutorials/cfd-fea-advanced/aircraft_mach.cas*.
- Right-click on the model and select *Boundary View* in the *Display* submenu.
  Tip: You could also have accessed the *Boundary View* module by clicking on its macro button in the upper part of the Project View, after selecting the model. Macro buttons provide easy access to the modules that are most commonly used and/or that have been recently used with the currently selected object.

The *Boundary View* module now appears in the *Display* folder of the Project View and the green arrow shows its input is the aircraft model. Its properties are displayed in the Properties area. You now see the boundaries of the model in the 3D viewer. Remember that boundaries are the limits of the domain under study, here they delimit the air flow. In Avizo XWind Extension, boundaries are classified according to their *type*, which is the physical condition imposed on a boundary for the simulation (boundary condition).

The types of boundaries are listed in the Properties area of the module in the multi-selection port **Boundary types**.

- In the **Boundary types** port, deselect the items: `Symmetry` and `Pressure Far Field`.

Only the boundaries of type `Wall` remain and they delimit half of a YF-17 Cobra aircraft.
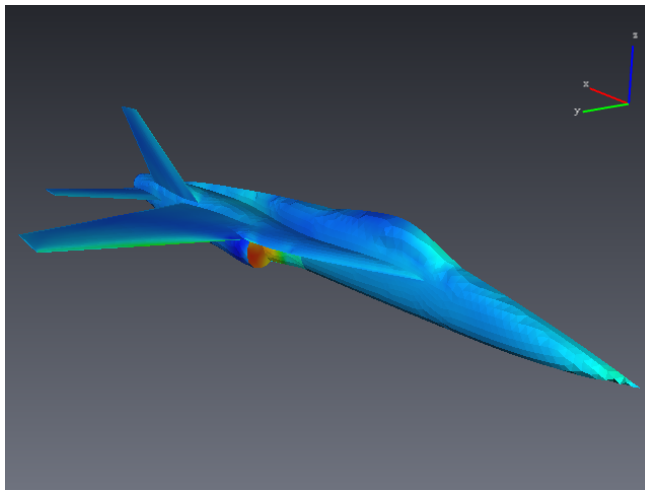
Figure 11.62: Pressure field on the boundaries of a YF-17 Cobra aircraft.

- Click in the 3D viewer window and press the [SPACE] key to enlarge the aircraft view (this does a View All operation).
You can also zoom in and out more accurately by using your mouse wheel, moving your mouse while simultaneously pressing the left and middle buttons, or by clicking the zoom mode button

  🔍 and moving your mouse up and down. If necessary, go back to viewing mode by clicking

  on the trackball button 👆 .
- Rotate the aircraft to see it the right way up.
To do this, press the left mouse button in the 3D viewer and move the mouse until you reach the desired orientation.
- Select Pressure in the *Colorfield* port of *Boundary View*.
- Select *Data Mapping* in the *Coloring* port.
- Select *node values* in the *Value Mapping* port.

You can make a snapshot by selecting the camera in the toolbar of the 3D viewer. Please refer to *Snapshot Dialog* description for more details.

If you want to display a compass on your viewer, like in Figure 11.62:

- Go to Avizo Preferences (*Edit / Preferences...*);
- In *Layout* tab, click on the *Compass* tab of *Viewer gadgets* group;

- Check *Show the compass* and select "axis.iv" :
- You can choose the position of the compass using *Compass position* option (*Upper right* in the figures).

11.4.4   Units and legends

To complete this visualization, we can add a legend that displays the color scale as well as the name and units of the data set displayed.

- Select Pressure in the *Project View*.
You can select Pressure where it appears under the model or where it appears under the *Boundary View* module, whichever is more convenient.
- Notice that when a data set is selected, the Properties area gives you some information about it, for example the units and range.
- You can change or adjust the colormap if needed.
- Click the *create legend* button in the **Options** port in the Properties area.

The legend is displayed in the 3D viewer and a *Data Legend* module is created under the *Pressure* data in the Project View. Use the Properties area of the module to set the legend properties (position, size...)
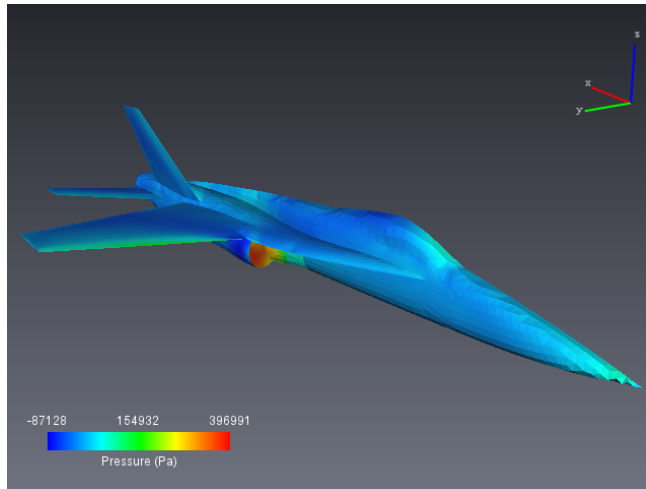
Figure 11.63: Pressure field on the boundaries of a YF-17 Cobra plane, with legend.

as desired.

11.4.5 Saving your project

- Select *Save Project As...* in the *File* menu.
- Enter the file name `getstarted.hx` for example.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_firstvisu.hx.

11.4.6 Tip: Template projects

Template projects can be used to ease repetitive tasks on a set of similar data. A template project is a backup of an original project that can be replicated on another data set of the same type. Please refer to *Section "Template Projects Description"* for a complete overview of template projects.

We will save the present project as a template project.

- Select *Save Project As Template...* in the *File* menu.
- The input selection dialog appears with a list of data sets that may be used as input to the template (data that will be replaced at run-time). Choose the model and the `Pressure` as template inputs.
- Change the input labels to `modelFluent.cas` and `scalarfield`.

- Click OK.
- Choose a name and a destination folder for the template project.
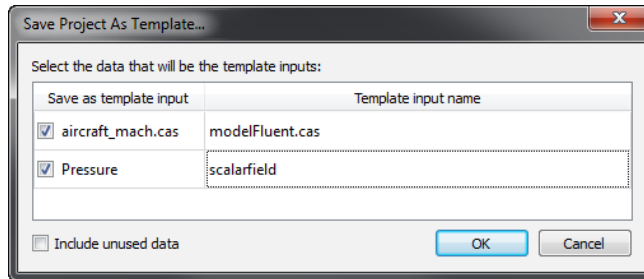
Figure 11.64: Input selection dialog box for template projects.

In case of any problems or uncertainties you can find the same template project predefined in your tutorial folder under the file name cfd-fea-advanced/wind_templatenetwork.hxtemplate (load it by selecting *Open Data...* in the Project View).

Now that the template is saved, you can easily make the same kind of visualization on any Fluent model connected to a scalar field.

- Hide the *Boundary View* and *Data Legend* modules.
  To do this, uncheck the boxes next to them in the Project View.
- Load `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder again and add its `Pressure` data set again.
  They appear in the tree view under the names `aircraft_mach2.cas` and `Pressure2`.
  Tip: You can quickly reload a recently used file using the Project View's popup menu. Right-click in the Project View on any line that is empty or contains a folder icon, then select the file from the *Recent Files* submenu.
- Hide the *Bounding Box* connected to the new model by unchecking the box next to the *Bounding Box* module in the *Display* folder in the Project View.
- Go to the *Project >Create Object...* menu and select the previously saved template in the *Templates* category.

The run dialog appears on template execution.

- Select the new model `aircraft_mach2.cas` and the scalar field `Pressure2`.
- Click OK.

The *Boundary View* visualization of the second model has now appeared in the 3D viewer and should be perfectly identical to the previous one (see Figure 11.63, appart from the zooming and rotation).
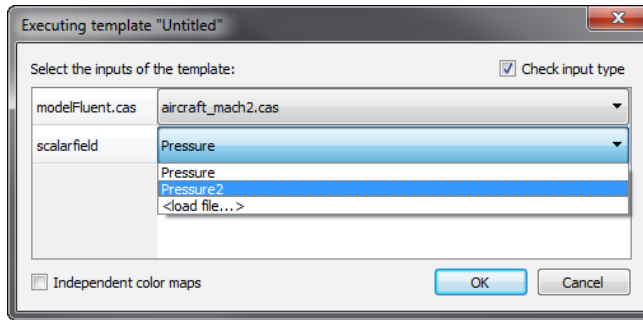
Figure 11.65: The template project run dialog.

This project was rather simple and was reloaded on the same model, to keep the example simple, but keep in mind that template projects become very useful and will save you time if you have several modules to connect in the same way to different models and data sets.

11.4.7 Time animation

Time animation is essential for transient data analysis. We will now see how to read and visualize such data.

- Remove all objects from the Project View.
  To do this you can use [Ctrl+N] to start a new project (you can discard the current project) or you can right click in the Project View window and select *Remove All Objects*.

We will now load the time dependent data.

- In the *File* menu, select *Open Time Series Data....*
- Navigate to the tutorials/cfd-fea-advanced/fan folder.
- Select and open all 11 model files fan-0070.cas through fan-0080.cas (don't open the .dat files).
- Choose Pressure in the Datasets selector.
- Hide the *Bounding Box*.

In the *Compute* directory of the Project View, a time *Sequence Controller* module has appeared.

- Create a *Boundary View* module as before (right-click on the model in the Project View).
- In the **Boundaries** port, click on the *Deselect all* button and then select only wall-7.
- Set the *Colorfield* port to Pressure.
- In the *Coloring* port, choose *Data Mapping*.

- Select *Node values* in the *Value Mapping* port.
- Rotate the display.
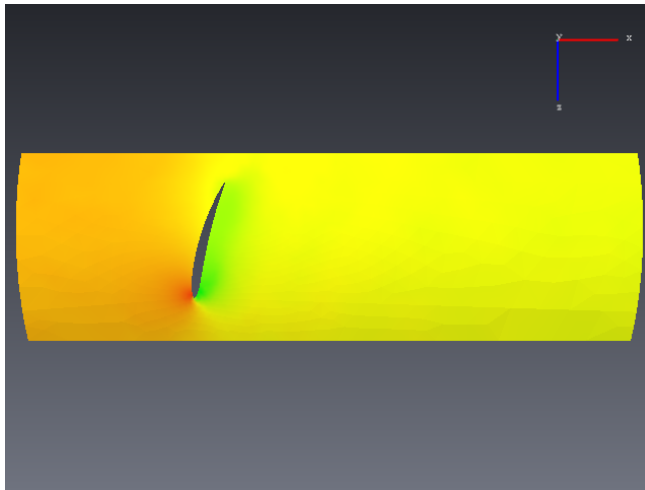
Figure 11.66: Pressure field on a tip section, around a blade.

In case of any problems or uncertainties you can find the same project predefined in your tutorial `fan` folder under the file name data/tutorials/cfd-fea-advanced/fan/wind_timeseries.hx.

Use the time sequence controller module to animate the display of pressure over time. Select the controller in the Project View.

- Keep the **Time mode** port on *time step*. The *physical time* stands for the physical time associated with each time step.
- Move the slider in the **Time step** port rightwards. With the step button ▶ next to the time slider you can go through the data step-by-step. By clicking the play button ▶ you can run the whole animation. Both buttons are also available for the reverse direction.
- Clicking on the configuration button ⋯▾ of the **Time step** port opens a context menu that allows you to play the animation once, play it over and over (*Loop* mode), or play it forward and backward alternately (*Swing* mode).

You might have noticed that the range of the **Colormap** port of the `Pressure` data set does not change with each time step. This is because the colormap range is set by default to the global range of the first time step data set. You can change it by setting the minimum and maximum to the values you want and the colormap range will remain set to this range during the animation.

11.5 Avizo XWind Extension Models Information and Display

In this section we will learn how to load a model and its associated data, how to retrieve information about the model and how to display it.

- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Pressure` field.

11.5.1 Properties and parameters

Model

The Properties area contains basic information about the selected model or data field.

- Select the model in the *Project View*. Details about it appear in the Properties area (see Figure 11.67).
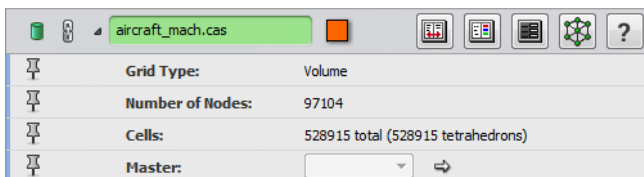
Figure 11.67: Properties area of the model

In the Properties area of the model you can find the grid type (volume or surface), the number of nodes of the mesh, the number of cells and their type.

- Click on the *Model Colors Editor* button (see Figure 11.68).

A model might be composed of regions and boundaries. Regions are parts of the domain, generally corresponding to a physical property (e.g., the rotating part of a pump is distinguished from the static part) or to the different partial meshes a global mesh is made of. Boundaries are the limits of the domain where boundary conditions are defined.

A different color is associated with each region and boundary of the model. These colors are used, for example, by the *Grid View*, *Boundary View* and *Isosurface* display modules. We will talk more about this in a later section. Close this dialog.

- Click on the *Data Parameter Editor* button .
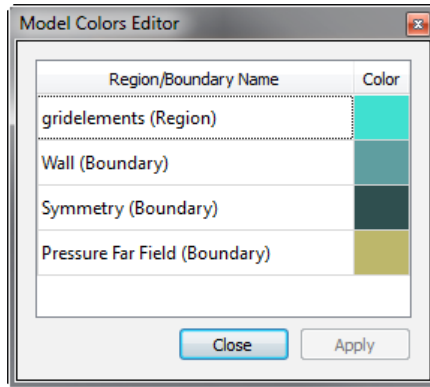
Figure 11.68: Models color editor

The *Parameter Dialog* window pops up (see Figure 11.69). In this window you will find additional information about the model, including the boundary and region names, id numbers and types, physical details about the material(s) under study and solver information about the model. Close this dialog.
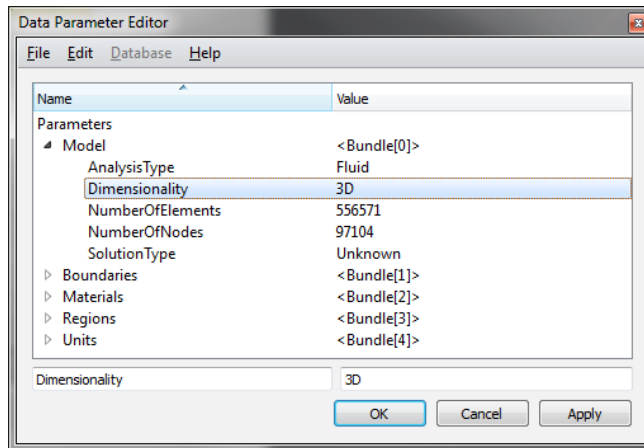
Figure 11.69: Parameter dialog window

Data field

- Now select the `Pressure` scalar field in the *Project View*. Details about it appear in the Properties area.

In the Properties area of the scalar field you can find (see Figure 11.70):

- The data type (**Type**),
- The physical quantity under study and its unit (**Content** and **Unit**),
- The global range of the data (including the mesh and the boundaries: **Range**),
- The range of the data inside the mesh (**Dataset range**),
- The range of the data on the boundaries (**Boundaries range**),
- The data binding (per node or per cell) (**Binding**).
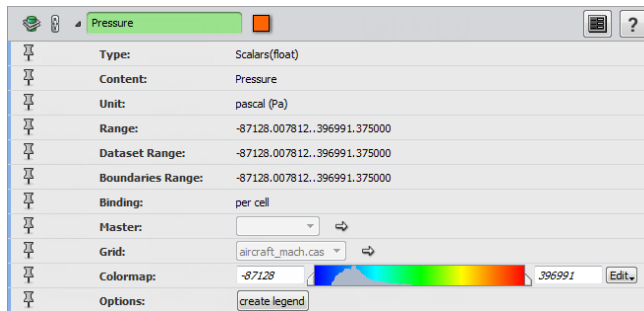
Figure 11.70: Properties area of the `Pressure` scalar field.

11.5.2 Colormaps

In the Properties area of the `Pressure` field, you can see that a colormap is connected. For convenience a default colormap is defined for each type of data field. The colormap connected to the data field will (initially) be used by all the display modules connected to the field. And therefore modifying the data field colormap affects all the connected display modules. It is also possible to use a different colormap for any of the display modules.

Figure 11.71: Colormap port of the `Pressure` field.

We will now see in detail what you can do with the options of the *Edit* menu of the **Colormap** port.

- Display the *Edit* menu of the **Colormap** port.
  Do this by clicking the *Edit* button or by right-clicking in the color bar.

You can change the colormap:

- Select one of the default colormaps listed in the menu, or

- Go to *Options-->Load Colormap...* to load a colormap from the directory of your choice or
- Go to *Options-->Edit Colormap...* and use the *Colormap Editor* to edit your own colormap.

In case the range has been changed and you want to adjust it back to the data field range:

- Select *Adjust range to* in the *Edit* menu of the **Colormap** port and select the data.

Tip: The range of the colormap is set to *Local* and adjusted to the field global range (except for the time series data, as seen in the Time Animation section of the *Getting Started* chapter). You can switch between the global and local range modes by selecting and deselecting *Options-->Local range*. In the global range mode, the coordinates used to map data values to colors are taken from the colormap itself. If the same colormap is used by two different fields and if the range is modified, both fields colormap ranges are updated. In contrast, in the local range mode the coordinates are defined by the port itself. Thus, although the same colormap is used by two different fields, the ranges can still be different. As many fields may share the same colormap in the Avizo XWind Extension, we advise you to be very careful when using the global range mode.

11.5.3 Viewing the grid

- Hide the *Bounding Box*.
- Right-click on the model.
- Select *Grid View* in the main menu or in the *Display* submenu.
- In the **Options** port, select *cell filtering*.

Two new ports appear that allow you to select and deselect regions and/or materials. In the present case there is only one of each so this option is not useful but keep in mind that it exists for more complex geometries (e.g., a pump with a rotor and a stator).

- In the **Rendering** port, select *Solid Outline* as draw style.

In this view you can observe the mesh on the boundaries of the model (see Figure 11.72).

Model color editor

In the *Rendering* port, the *Coloring* option is set to *Per Region*. This means that each region of the model (here there is only one) is rendered using the color associated with it in the *Model Colors Editor*. We will now change this color.

- Select the model and open the *Model Colors Editor* (click on the button ⊞).
- Click on the color of the `gridelements` part.
- The *Color Dialog* window pops up. Define a new color and press OK.
- Back in the *Model Colors Editor*, click Apply. The color is updated in the 3D viewer. Now click Close.
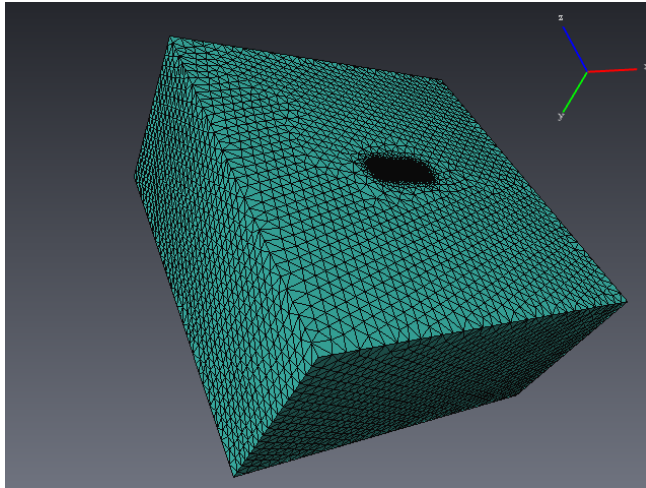
Figure 11.72: Grid view: the mesh on the boundaries of the model.

In case there are several regions in the model and you want them all to have the same color, you can choose a uniform coloring.

Select the *Grid View* module then:

- Select *Uniform* in the *Coloring* section of the **Rendering** port. A **Uniform color** port appears.
- Click on the color sample. The *Color Dialog* window pops up.
- Define a new color and click OK. The color is updated.

Cell information

- Click in the 3D viewer window (to change focus) and press the [ESC] key to switch the viewer into interaction mode. The cursor should now be an arrow.

    Alternatively, you can click on the arrow ➤ button in the 3D viewer menu bar or right-click in the 3D viewer window and unselect Viewing.

- Left click on the mesh in order to select a cell.

Information about the cell will appear in the upper left corner of the 3D viewer and a *Spreadsheet In Viewer* module appears in the Project View. The behavior is controlled by the **On left mouse click** port of the *Grid View* module. This way you can retrieve, for the selected cell:

- the cell topology,
- the physical type of the cell material,
- the coordinates of the picked point,

- the volume of the cell,
- the data value (if the *Grid View* is attached to a data field instead of a model).

Go back to viewing mode, for example by clicking in the 3D viewer window and pressing the [ESC] key.
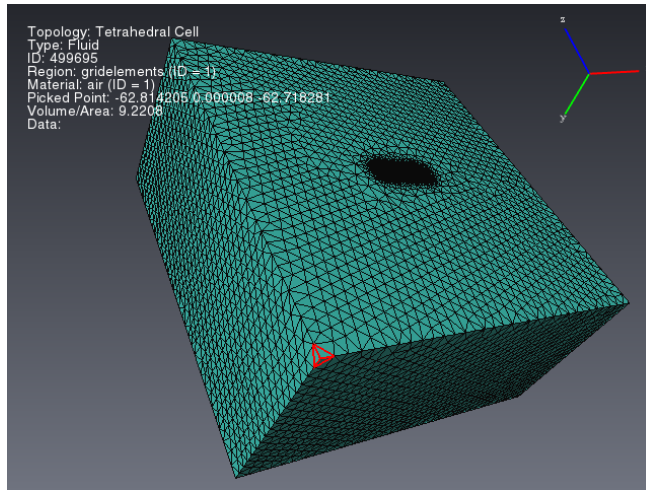
Figure 11.73: Grid view and cell information.

11.5.4 Viewing the boundaries

- Hide the *Grid View* and the *Spreadsheet In Viewer* modules.
- Right-click on the model in the Project View.
- Select *Boundary View* in the main menu or in the *Display* submenu.

Model color editor

In the **Coloring** port of the *Boundary View*, the *Per Boundary Type* coloring is selected. This means that each boundary of the model is colored with the color associated with its type in the *Model Colors Editor*. To change these colors, proceed the same way you did to change the region colors. It is also possible to choose a uniform coloring, the same way as previously.

Data mapping

If your solution contains data fields stored on the boundaries, you might use data mapping to color them.

- Select *Data Mapping* in the *Coloring* port.

- Select *Pressure* in the *Colorfield* port.
- Select *Node values* in the *Value Mapping* port.

The pressure contour is now displayed on the boundaries of the model. The colormap that is used is the `Pressure` field's colormap. If you want to modify the colormap or its range, do that in the data Properties area of the `Pressure` field.

Tip1: If the *Boundary View* is currently selected, you can quickly select the `Pressure` field by clicking the right-arrow button in the **Colorfield** port.

Tip2: For convenience you can keep the `Pressure` field's **Colormap** port visible in the Properties area even when the *Boundary View* (or other display module) is selected. Select the `Pressure` field, then click on the *pin* to the left of the **Colormap** port. Select the *Boundary View* again. Results are shown in Figure 11.74.
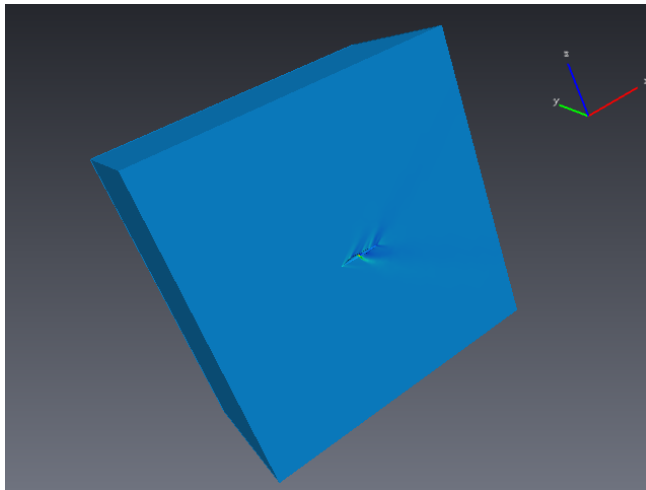
Figure 11.74: Data mapping of the pressure on the boundaries of the model.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_boundariesview01.hx.

Boundaries filtering

The previous view is not very interesting. We would rather see the pressure on the aircraft boundaries than far from it.

The boundaries of the aircraft are of type `Wall`. The rest of the boundaries of the model are of type `Symmetry` or `Pressure Far Field`. Select the *Boundary View*.

- In the **Boundary types** port, deselect `Symmetry` and `Pressure Far Field` types.
- Enlarge the view and rotate it in order to get a better image of the aircraft (see Figure 11.75).

You can see that in the **Boundaries** port, the boundaries that are of type `Symmetry` and `Pressure Far Field` are now deselected. So you could have achieved the same effect by deselecting non-wall boundaries one by one in this port.
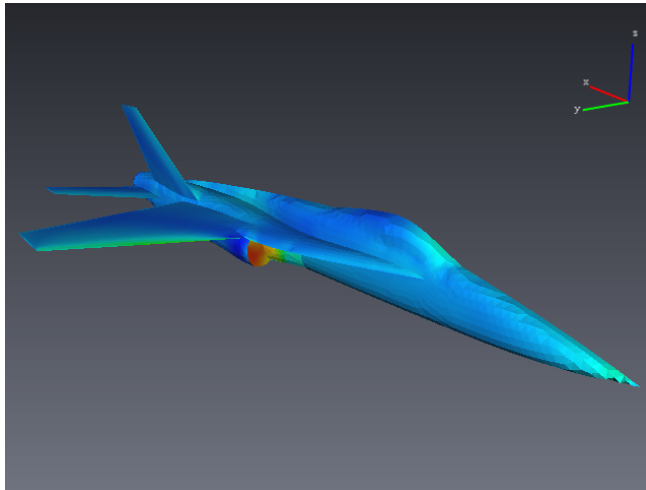
Figure 11.75: Pressure field on the boundaries of a YF-17 Cobra aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_boundariesview02.hx.

Extracting surfaces from boundaries

We will now create a surface from the boundaries of the aircraft.

- Click the *create* button in the **Create surface** port of the *Boundary View*.

The surface `aircraft_mach.surf` has been created. This surface can then be used as a seed region for illuminated streamlines distribution (see Chapter 11.7).

11.6 Avizo XWind Extension Scalar Fields Display

In this section we will introduce the different features you can use to display scalar fields.

- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Pressure` field.
- Connect a *Boundary View* to the model and set the *Coloring* to *Per Boundary Type*, then deselect *Symmetry* and *Pressure Far Field* in the **Boundary types** port.

- Hide the *Bounding Box*.

11.6.1 Scalar field profile on a cross section

- Right-click on the `Pressure` scalar field in the Project View.
- Select *Cross Section* in the *Display* menu.

In the Properties area of the module you can see some ports that we have studied in Chapter 11.5.

- *Value mapping* port: The *node values* and *cell values* options specify if a value of the field under consideration will be affected to each node of the mesh and interpolated along the cells or if a constant value will be associated with each cell.
  Select *node values* in the *Value Mapping* port.
- The *cell filtering* option allows restricting the *Cross Section* to selected regions of the model.

- *On left mouse click* port: *Display cell info* allows you to left click on cells and get cell information. This option is visible in the *Display Option* port.
- *Draw Style* allows you to set different draw styles. *Solid Outline* and *Wireframe* display the intersection of the mesh with the section plane. Keep (or come back to) the *Solid* setting.

Several coloring modes are available in the **Rendering** port. The default setting is *Data Mapping* which means the representation of the scalar field using its colormap with local range. You have already seen the *Uniform* mode and the *Per Region* mode that colors the parts of the cross section according to the region of the model they belong to.

- Now select the *Iso Contouring* mode in the *Coloring* section of the **Rendering** port.
- The **Uniform distribution** port has appeared. Set the *count* value to 30.

This option virtually transforms the colormap into a colormap with 30 steps (the actual attached colormap is not modified).

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_crosssection01.hx.

- Set the *Coloring* back to *Data Mapping* in the **Rendering** port.
- Click on the *xz* button in the **Orientation** port.
- Set the slider in the **Translate** port to 0.
- Rotate the display and zoom in to get a better view.
  Tip: Try using the viewer's "Seek" function. Activate Seek mode by pressing the "S" key or clicking its button ⊕ in the viewer menu bar. Now click on an interesting part of the model. The viewer automatically moves closer and sets the selected point as the center of rotation (see Figure 11.6.1).
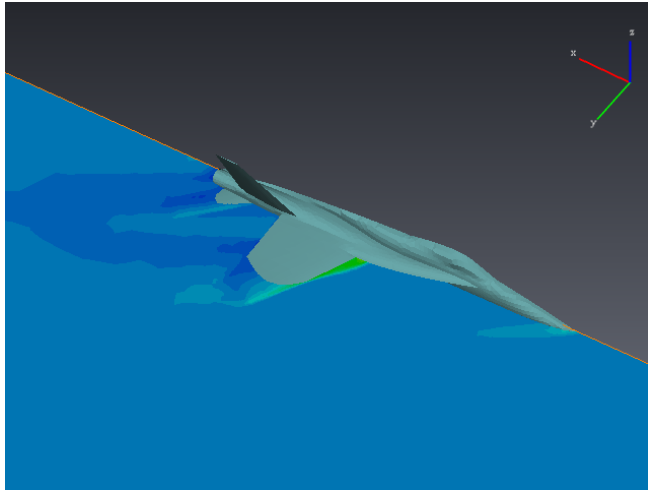
Figure 11.76: Cross section of the pressure field around a YF-17 Cobra aircraft using iso-contouring.
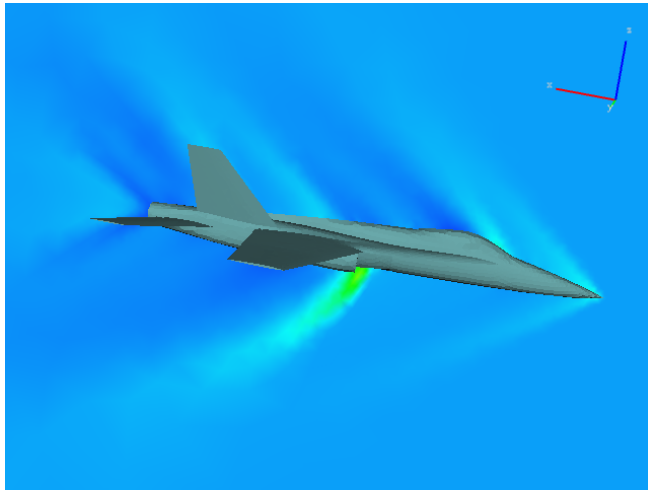
Figure 11.77: Cross section of the pressure field in the middle plane of a YF-17 Cobra aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_crosssection02.hx.

11.6.2 Scalar field isolines

We will now add isolines to the previous display.

- Right-click on the `Pressure` scalar field.
- Select *Isocontour Slice* in the *Display* submenu.

Two coupled objects were added to the Project View: a *Clipping Plane* that defines the plane in which the isolines are plotted, and the *Isocontour Slice* module itself. We want the *Clipping Plane* to coincide with our existing *Cross Section* plane.

In the *Clipping Plane* module:

- Select *xz* as **Orientation**.
- Set the slider in the **Translate** port to 0.

In the *Isocontour Slice* module:

- Set *num*, the number of isolines, to 50 in the **Values** port.
- Improve the quality of the plot by setting the *resolution* to 512 in the **Parameters** port.
- Rotate the display in order to match the cross section lighting with a good isolines view (the isolines may be difficult to see from some angles).
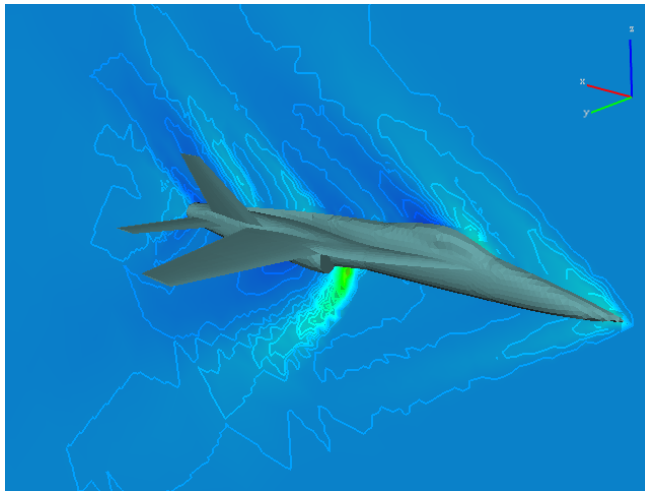
Figure 11.78: Cross section and isolines of the pressure field in the middle plane of a YF-17 Cobra aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_isolines01.hx.

11.6.3   Legend and captions

Our plot lacks some information about what is displayed.

Legend

- Right-click on the `Pressure` scalar field.
- Select *Data Legend* in the *Annotate* menu.
  (Alternatively, you could click *create legend* in the Properties area of the scalar field.)

The colormap, the range, the data name and the data unit are now displayed in the 3D viewer. You can set the size, position... of the legend in the Properties area of the module.

Caption

We will now give a title to our display.

- Go to the *Project >Create Object...* menu in the main menu bar.
- Select *Annotations / Caption*.
- Change the position of the text: Set the left and the top offset to 10 in the **Offsets** port.
- Change the text to "Pressure profile in the middle section of a YF-17 aircraft".
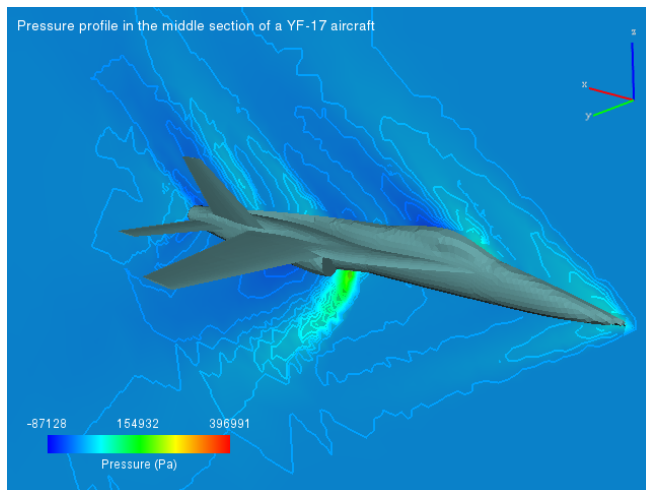
Figure 11.79: Pressure profile in the middle section of a YF-17 aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_isolines02.hx.

11.6.4 Isosurfaces of pressure

We will now use isosurfaces to display the Mach cones close to the aircraft.

- Hide the *Isocontour Slice* and the *Cross Section*.
  (To hide the *Isocontour Slice* you must actually hide the *Clipping Plane*.)
- Right-click on the `Pressure` scalar field.
- Select *Isosurface* in the *Display* menu.
- Set the **Isovalue** port to 1000.
- Select *Data Mapping* as coloring mode.
- Choose the `Pressure` as **Colorfield**.
- Change the title to "Pressure isosurfaces: P = 1000 Pa" in the **Text** port of the *Caption*.

Region of interest (ROI)

We would like to restrict the isosurface to a region close to the aircraft. To this end, we can use a region of interest which is a box that restricts the output of many visualization modules.

- Choose *ROI Box* in the *Display* menu of `Pressure`.
- You can expand the view to see entire the ROI by clicking on the viewer and pressing [SPACE]. The initial ROI is the extent of the entire data set, same as the bounding box.
- Switch to interactive mode (click on the viewer and press [ESC] or click on the arrow ➤).
- In the viewer window, drag the green squares on the ROI and modify the size of the box until it includes only a small region around the aircraft.
- In the **ROI** port of the *Isosurface* module, choose *ROI Box*.

We would like to see the Mach cones from different points of view. In order to see the aircraft too, we will use the opacity factor of the *Isosurface*.

- Hide the *ROI Box*.
- Set the **Opacity** of the *Isosurface* to 0.15.
- Rotate and zoom in and out to get a different view.
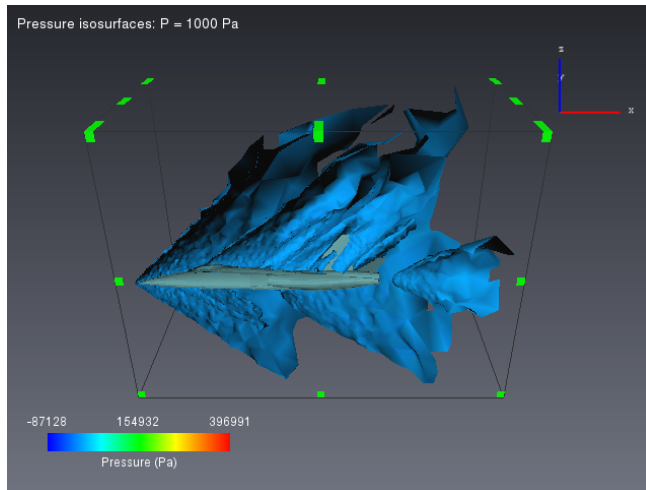
Figure 11.80: Mach cones as isosurfaces of pressure in a region of interest.
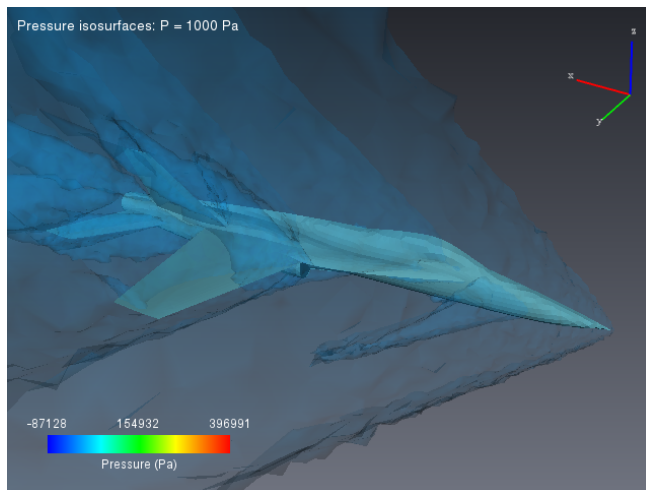
Figure 11.81: Mach cones as isosurfaces of pressure for a YF-17 aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_isosurfaces.hx.

11.7 Avizo XWind Extension Vector Fields Display

In this section we will introduce the different features you can use to display vector fields.

- Open `wing.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Velocity` vector field.
- Connect a *Boundary View* to the model and unselect everything except `Wall` in the **Boundary Types** port.
- Hide the *Bounding Box*.

You can see a wing in the 3D viewer. We will study the air flow around this wing.

11.7.1 Particles animation

We do not know anything about the flow around this wing. To get a quick overview of the flow and see which regions of the model we should focus on during our study, we will seed particles in the vector field and observe their behavior.

Region of interest (ROI)

The domain is very large compared to the wing size. We would like to focus on the flow next to the wing. To this end, we can use a region of interest (ROI) which is a box that restricts the output of many visualization modules. In this case, we will use an ROI to define the starting position of our particles.

- Right-click on the model `wing.cas`.
- In the *Display* menu, select *ROI Box*. A box appears in the 3D viewer, this is the ROI. Initially the ROI is identical to the bounding box of the data set.
- Change the shape of the ROI by dragging the green squares. Change the position of ROI by dragging its faces. Resize and position the ROI close to the leading edge of the wing.

Remember that you must be in interaction mode, indicated by the arrow cursor, to affect the ROI. Switch modes by pressing the [ESC] key or clicking the buttons in the viewer menu bar.
Tip: You often need to switch between interaction mode and trackball mode multiple times while performing tasks like resizing and positioning an ROI box. While in interaction mode, you can temporarily switch to trackball mode by holding down the [ALT] key. When the key is released, the viewer returns to interaction mode.

Animated Particles

- Right-click on `Velocity` and select *Animated Particles* in the *Display* submenu.
- In the **SeedROI** port, select the `ROI Box` ROI.
- Modify the **Frequency** to *every* 10 *timestep*.
- Set the step size to 0.0002 in the **Animate** port in order to slow down the animation.
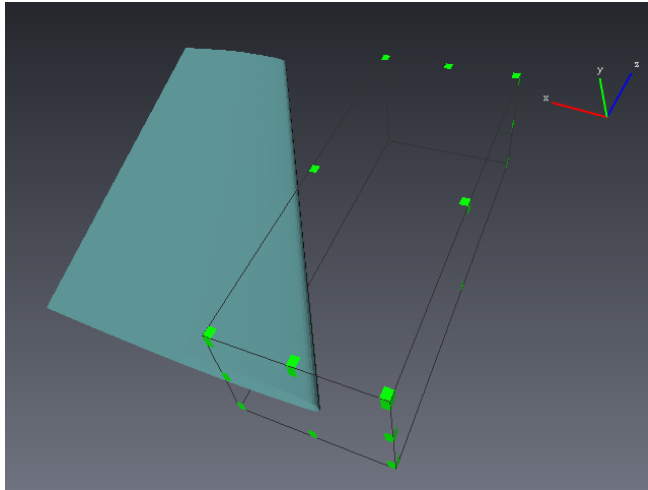
Figure 11.82: Region of interest.

- After a few seconds (time for some particles to have lived their life and gone "old"), select *Adjust range* in the *Edit* menu of the **Colormap** in order to adjust the range of the colormap to the range of the particles age.
- Select *spheres* in the **Shape** port and set their size to 0.1.

What we can learn by observing the Properties area of the *Animated Particles* module is that 10 particles are seeded randomly accross the ROI every 10 time steps. They are colored by age, which means the longer a particle remains in the model, the more red it becomes.
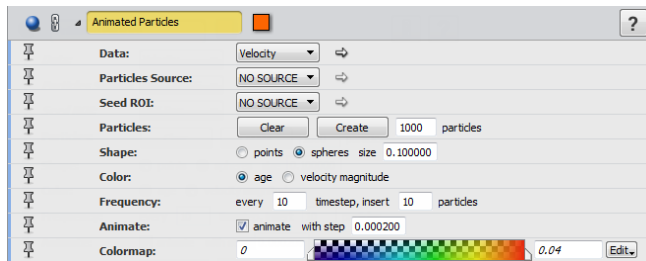
Figure 11.83: Properties area of the *Animated Particles* display module.

The behavior of the particles indicates the presence of a region of vorticity close to the wing: indeed you can see some particles swirling and becoming red (that is to say old) close to the upper side of the wing (see Figure 11.84).
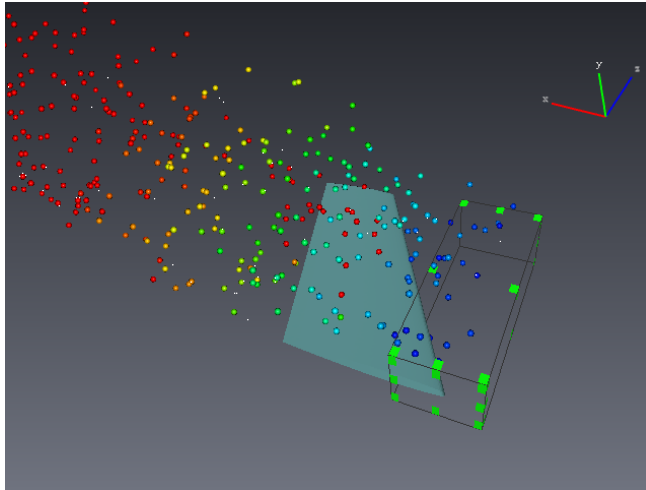
Figure 11.84: Animated particles seeded close to the wing.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_animatedpart.hx.

Stop the animation:

- In the **Animate** port, uncheck the *animate* button.
- Clear all the particles from the 3D viewer using the *Clear* button in the **Particles** port.

11.7.2 Illuminated streamlines (ISL)

The ISL technique is used to compute a large number of field lines by integrating the vector field starting from random seed points. We will restrict the region where the points are seeded to the previous ROI.

- Move the ROI closer to the wing. To do so, switch to interaction mode and drag the ROI by clicking on one of the faces of the box and holding while you move the mouse.
- Right-click on `Velocity` and select *Illuminated Streamlines* in the *Display* menu.
- In the **Seed ROI** port, select the `ROI Box` ROI.
- Set the number of lines in **Num Lines** to 400.
- Set the lines **Length** to 100.
- Set the **Step size** to 0.001.
- Click Apply.
- Hide the ROI.

Using two viewers

We will use two viewers to see the recirculation of the air in two different ways.

- Click the two viewers side-by-side button ▦ in the 3D viewer menu.

- In the Properties area of *ROI Box*, select/unselect the display on the right viewer by clicking on the right red half of the viewer toggle ▯▮. This will make the ROI appear/disappear from the right viewer. You can do the same for the *Bounding Box* if desired.

- Rotate and zoom in and out to get two different views, something like Figure 11.85.
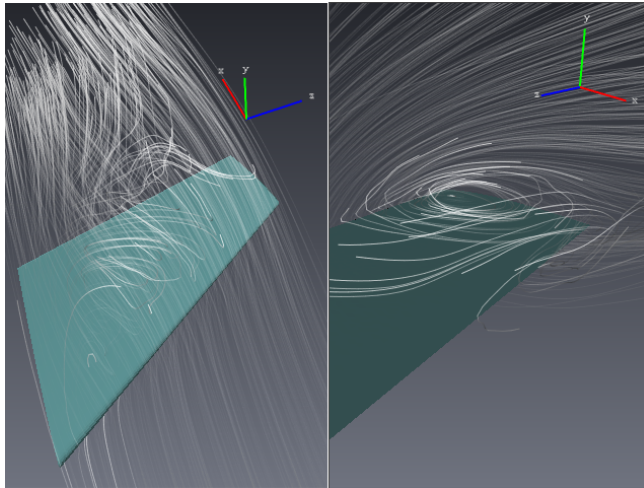
Figure 11.85: Illuminated Streamlines of the velocity vector field.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_displayisl.hx.

Tip: You might want to seed the ISLs from a given boundary (a velocity inlet for example). To do so, first extract the chosen boundary (use the **Create surface** option of the *Boundary View* module as explained at the end of Chapter 11.5). Then connect the created surface to the **Distribution** port of the *Illuminated Streamlines* module. Be aware though that a line will be seeded from each node of the surface mesh and that, therefore, the display might take a while to appear in the viewer.

Two other modules use illuminated streamlines: *Illuminated Streamlines Slice*, visualizes a surface vector field using ISLs, and *Illuminated Streamlines Surface*, intersects an arbitrary 3D vector field and visualizes its directional structure in the cutting plane using ISLs. A demonstration of these modules is given at the end of Chapter 11.9.

11.7.3 Line integral convolution (LIC)

This method consists of intersecting the vector field and visualizing its directional structure in the cutting plane.

- Go back to only one viewer (click on the one viewer button ▣ in the 3D viewer menu).
- Create a new ROI that contains the wing and a part of the domain behind it.
  Tip: You already know how to create a new ROI by right-clicking the model and using the *Display* sub-menu. However, this results (as before) in a ROI the same size as the bounding box, meaning that you have to zoom out to manipulate it down to the desired size. It may be more convenient to duplicate the existing ROI that is already close to the desired size. Right-click on the *ROI Box* in the Project View and select *Object/Duplicate Object* in the popup menu.
- Right-click on `Velocity` and select *Stream LIC Slice* in the *Display* menu.
- Select the new ROI `ROI Box 2` in the **ROI** port.
- If necessary, move the plane with the **Translate** port to position it at about two thirds of the length of the wing.
- Set the resolution, in the **Lic** port to 700.
- Press the Apply button.
- Hide the ROI.
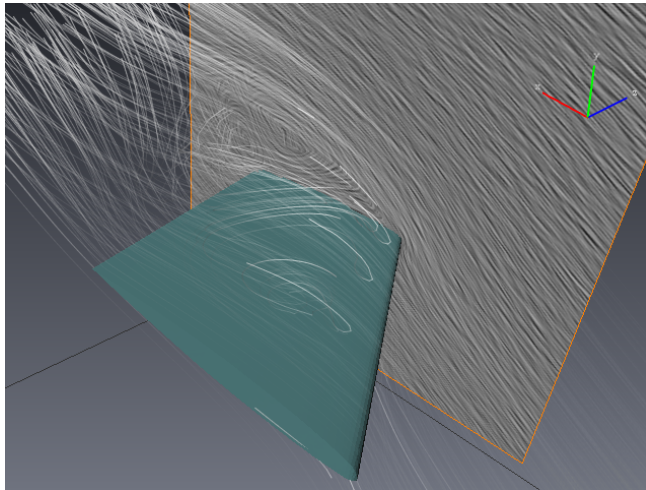
Figure 11.86: LIC representation of the velocity close to the wing.

Tip: You can move the plane using the **Translate** port in the Properties area or by dragging the plane in the Viewer window (switch to interaction mode if necessary). After moving the plane, press the Apply button in the Properties area to recompute the LIC.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_planarlic.hx.

11.7.4 Vectors in a plane

- Hide all display modules, except the *Boundary View*.
- Right-click on `Velocity` and select *Vector Plane* in the *Display* menu.
- Choose `ROI Box 2` in the **ROI** port.
- Set the **Scale** to 0.01 and the **Sampling distance** to 0.08.
- Set the **Translate** in order to see the recirculation correctly.

Now we would like the vectors to be colored in accordance with the vector's magnitude.

- Right-click on `Velocity` in the Project View and select *Magnitude* from the *Compute* sub-menu. The *Magnitude* module appears in the Project View just under the *Velocity* Data , showing its input (green arrow) is `Velocity` and its output (red arrow) is `Velocity.Magnitude`. The new velocity magnitude data set appears in the Project View underneath the model.
- Display a legend for `Velocity.Magnitude`.
- In *Vector Plane*, set the **Rendering** coloring mode to *Data Mapping* and the **Colorfield** to `Velocity.Magnitude`.
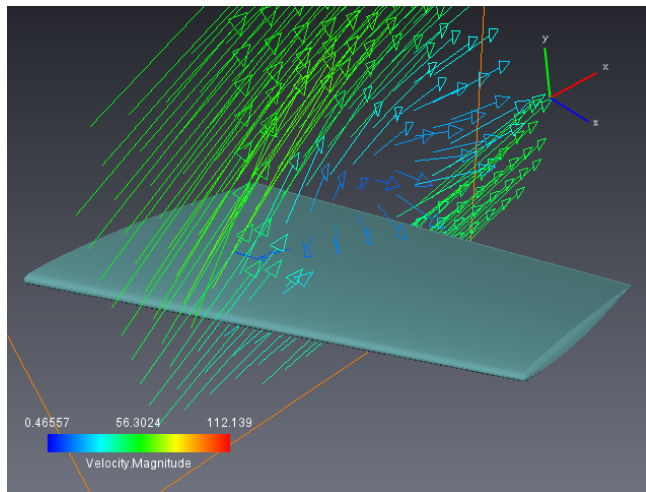
Figure 11.87: Velocity vector in a plane cutting the wing.

Similar to the *Stream LIC Slice* you can move the plane using either the **Translate** port or direct dragging. Unlike *Stream LIC Slice* the vectors are dynamically recomputed as the plane moves.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_vectorplane.hx.

11.7.5 Stream ribbons

- Hide or delete all the display modules, except the *Boundary View*.
- Right-click on `Velocity` and select *Stream Ribbons* in the *Display* menu.

Streamlines seeded from a line appear in the 3D viewer.

- Click *Show* in the **Dragger** port of the *Stream Ribbons*.

A dragger appears for the line from which the streamlines are seeded.

- Click in the Viewer window and switch to interaction mode.
- Move the dragger and use the green spheres to rotate the dragger until the line is parallel to the upper side of the wing.
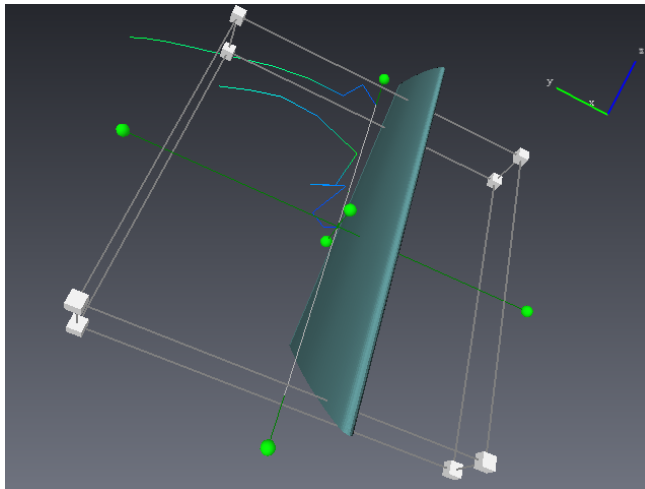
Figure 11.88: Stream ribbons dragger.

- Select *ribbons* in the **Mode** port.
- Set the **Resolution** to 1.
- Set the **Density** to 0.8.
- Set the **Width** to 0.35.

- Set the **Length** to 3.
- Click *Hide* in the **Dragger** port.
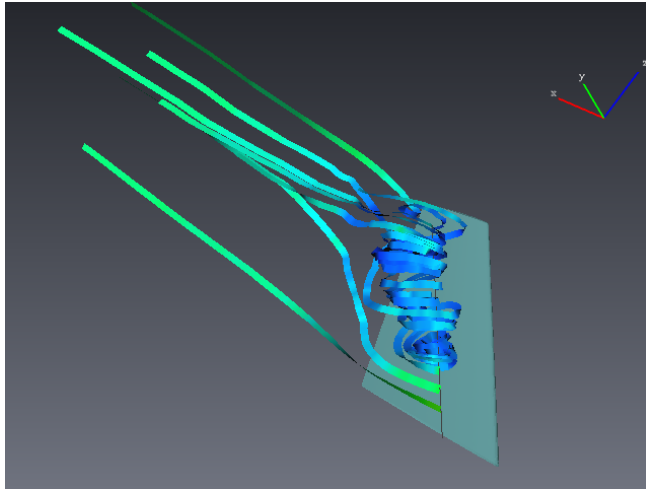
Figure 11.89: Stream ribbons close to the wing.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_streamribbons.hx.

11.7.6   Find the 3D critical points

3D critical points are points around which different flow patterns can be identified. E.g. the flow behavior around a source is uniformly an inflow, while around a sink it is an outflow. The flow around a saddle point is a mixture of both.

From a mathematical point of view, a first order critical point for a 3D vector field is a point where the velocity is null and the determinant of the velocity Jacobian matrix is not. First order critical points can be classified by an eigenvalue/eigenvector analysis of the Jacobian matrix. The *Critical Points* display module finds and represents critical points by icons of different shapes, depending on the critical point classification. Please refer to the documentation of *Critical Points* for more details.

- Hide or remove the previous display modules, keep only the *Boundary View*.
- Right-click on `Velocity` and select *Critical Points* in the *Display* menu.
- Reduce the **Icons size** to 0.05.
- Click Apply.
- Select the *show* option to display the illuminated streamlines seeded from the critical points.
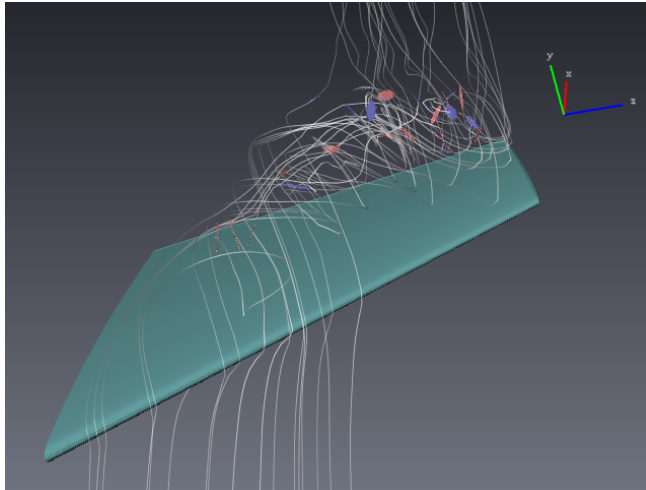
Figure 11.90: Critical points of the velocity vector field and illuminated streamlines seeded from the points.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_cp3d.hx.

11.8 Avizo XWind Extension Statistical and Arithmetic Computations

- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Pressure` and `Density` scalar fields and the `Velocity` vector field.
- Hide the *Bounding Box*.

11.8.1 Surface and volume integrals

In the Avizo XWind Extension, statistical modules allow you to compute statistics on the boundaries and in the volume of an unstructured model. The output of these modules are *spreadsheets* and these objects are created in the Project View. You can export them as .CSV files and then import them in Microsoft Excel (c) for further work.

We will not illustrate all the possible computations with examples here, but will give short examples of some integral computations. The workflow is basically the same every time, whatever the module and the computation chosen.

The statistic modules that can be connected to a model are listed in the *Measure And Analyze* right-click submenu. Some of these modules can also be connected to data fields.

Area computation

- Right-click on the model.
- In the *Measure And Analyze* menu, select *Surface Integrals.*
- Click Apply.
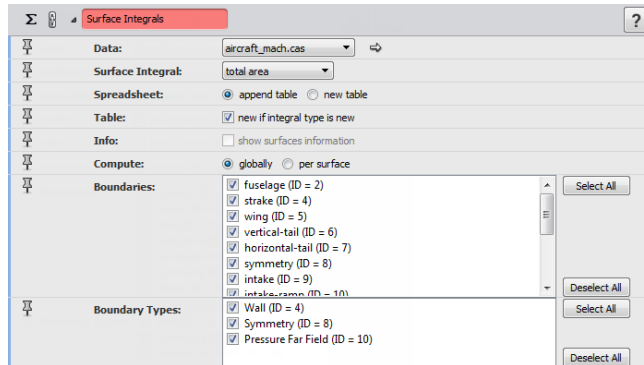
Figure 11.91: Surface integrals Properties area.

The results of the computation are printed to a spreadsheet that pops up. It contains the total area of all the boundaries of the model under study.

In the **Boundaries filter** port, all the boundaries are selected, which means that the computation is done on all the boundaries. The *globally* option is selected in the **Compute** port, which means that the area that is computed is the sum of the areas of all the boundaries.

- Select *per surface* in the **Compute** port.
- Click Apply.

You can now see that the area of each boundary has been printed to the spreadsheet.

- Select *globally* in the **Compute** port.
- Unselect the boundaries of type `Symmetry` and `Pressure Far Field` in the **Boundary types** port. The only boundaries now selected are the aircraft boundaries.
- Click Apply.

You can see that the area of the half aircraft (composed of boundaries 2, 4, 5, 6, 7, 9, 10 and 11) is equal to approximately 83.61 square meters.

Volume computation

- Right-click on the model.

Figure 11.92: Surface integrals spreadsheet.

- In the *Measure And Analyze* menu, select *Volume Integrals*.
- Click Apply.

A new spreadsheet pops up and you can see that the volume of the flow domain is equal to approximately 1024215 cubic meters. In case the model is composed of several regions, you can use the **Regions filter** port to restrict your computation to some regions the same way we did for boundaries.
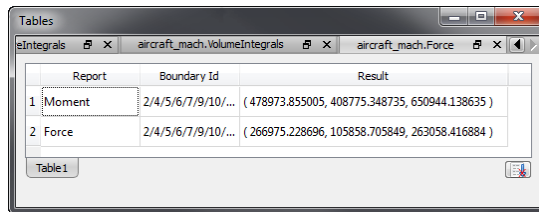
Figure 11.93: Volume integrals spreadsheet.

Pressure force vector computation

- Right-click on the model.
- In the *Measure And Analyze* menu, select *Force*.
- Press Apply.

Note that for convenience only boundaries of type `Wall` are preselected in this module.

The *Force* module computes the pressure force vector generated on the aircraft surface and the pressure moment about the moment center which is here set to the origin. Notice that the `Pressure` scalar field was identified and automatically selected in the **Pressure** port.

Figure 11.94: Pressure force vector spreadsheet.

Scalar Field mean value computation

Volume and surface integrals can also be computed on data fields.

- Select the *Volume Integrals* module.
- In the Properties area, change the **Data** to `Pressure`.
- Select *mean* in the **Field integral** port. We will compute the mean value of the pressure in the whole volume.
- Click Apply.

As *new if integral type is new* is checked by default in the **Table** port, a new table opens, which title is *mean* in accordance with the field integral type.

Many other types of volumetric and surfacic integrals and statistics can be computed from the *Volume Integrals* and *Surface Integrals* modules. Computations on data fields can always be restricted to regions or boundaries using the appropriate filter.

Pressure surface integral computation on a sequence of cross sections

Surface integrals can be computed not only on 3d unstructured grid boundaries, but also on 2d unstructured grids, on 2d unstructured surfaces and on triangulated *Surfaces*. We will study here a convenient way to use the *Surface Integrals* module to compute integrals on several parallel cuts of a 3d model.

- Connect a *Boundary View* to the model.

Figure 11.95: The mean value of pressure in the model is approx. 1849 Pa.

- Unselect everything except `Wall` in the **Boundary types** port.
- Select *Cross Section* in the *Display* menu of the `Pressure`.
- Set the **Orientation** of the *Cross Section* to *yz*.
- Right-click on the *Cross Section* and select *Animate Ports*.

An *Animate Ports* module is created in the Project View. We will use this module to animate the value of the **Translate** port of the *Cross Section*.

- Select *Translate* in the **Port** port of the *Animate Ports* module.
- Set the **Time** to 0.
- Click on the configuration button of the **Time** port.
- Select *Configure* and set the *Increment* to 10.
- Change the **Tanslate** equation to *0.4*t+30*.
- Go back to the `Pressure` and connect to it a new *Surface Integrals* module.

Note there is a new port **Surfaces** in the Properties area of the module. This port is displayed because a surface (the *Cross Section*) has been detected in the Project View.

- Select *surface integral* in the **Field integral** port.
- Deselect all boundaries. Only the *Cross Section* is checked.
- Set the computation mode of the module to *auto-refresh*.
- Launch the animation in the *Animate Ports* module by pressing on the play button .

The spreadsheet is updated at each step of the animation with the value of the integral computed on each new plane. This highlights the important pressure raise in the environment of the aircraft.

11.8.2   Arithmetic computation

Secondary Variables computation

Figure 11.96: `Pressure` surface integral on a sequence of plane cuts orthogonal to the aircraft.

With Avizo XWind Extension, you can also implement your own computations, taking as inputs the variables on the unstructured model. As an example, we will now compute the momentum vector field (product of the density and velocity).

- Load the `Velocity` data from the Datasets selector by clicking on its button in the Properties area of the model (`aircraft_mach.cas`).
- Right-click on the `Velocity` vector field.
- Select *Arithmetic* in the *Compute* submenu.
- Select the `Density` as second input **Input B**.
- The momentum is a vector so you can keep *same as input* or select *vector* in the **Output data type** port.
- Enter the components of the momentum vector field: `B*Ax` in **Expr X**, `B*Ay` in **Expr Y**, `B*Az` in **Expr Z**.
- Click Apply.
  A new data module named `Result` appears under the model.
- Rename this module `Momentum`.
  To do this, right-click the module, select *Object/Rename Object* and type a new name in the dialog box. Alternatively, you can select the module, press [F2] and type a new name directly in the Project View.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_arithmetic.hx.

Regular data field generation

You can also use the arithmetic module to generate a data field on a regular grid and then use some other Avizo Lite Edition display modules that take only regular inputs, such as the *Volume Rendering* module.

- Attach an *Arithmetic* module to the `Pressure` data set.
- Choose *regular* in the **Output grid type** port.
- Enter the expression `A` in **Expr**.
- Change the **Resolution** to 100 by 50 by 100.
- Click Apply.
- Rename the resulting data set `Pressure.Regular`.

`Pressure.Regular` is the pressure field generated on a regular grid of size 100 per 50 per 100.

Volume rendering

- Use a *Boundary View* on the model to display the aircraft boundaries (display only walls).
- In the *Display* right-click submenu of `Pressure.Regular`, select *Volume Rendering.*
- In the Properties area of the *Volume Rendering* module, load and select *physics_VolRend.am* in the **Colormap** port:
    - Click on the **Colormap** port *"Edit"* button, then *"Option / Load colormap..."*
    - Open `data/colormaps/physics_VolRend.am`
- Set the colormap range to -5000, -1000.
- Open the *Colormap Editor* (through the *Window* menu or the Standard Toolbar shortcut).
- Click on the edit button for values superior to max range . The *Color Dialog* opens.
- Set alpha (A) to 0.
- Click OK.

You now have a new view of the Mach cone.
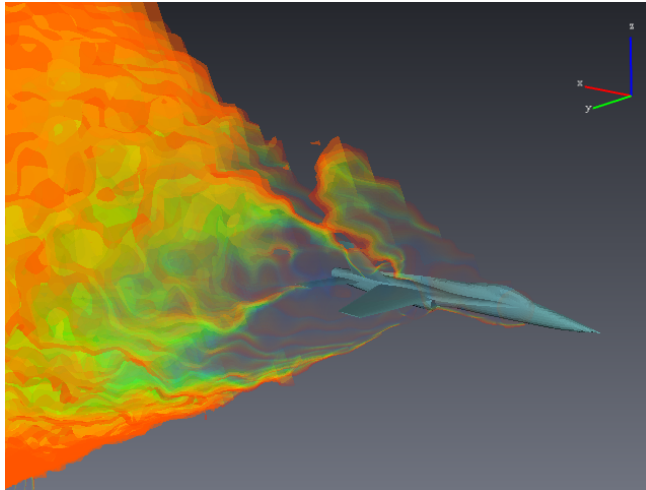
Figure 11.97: Volume rendering of pressure around the YF-17 aircraft.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_volrend.hx.

11.9 Avizo XWind Extension Vorticity Identification

This section will give you an overall view of the vorticity detection, computation and analysis features provided by the Avizo XWind Extension.

Although there are a number of studies devoted to vortex identification, there is no agreement on a formal definition. In the absence of a formal characterization of vortical structures, swirling motion around some central region is used as a working definition. Depending on the chosen approach, this leads to features that are either lines (see subsection about vortex core lines), surfaces or volumes (see subsection about vorticity-related variables).

- Open `wing.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Velocity` vector field.
- Connect a *Boundary View* to the model and keep selected only the walls in *Per Boundary Type* coloring mode.
- Hide the *Bounding Box*.

11.9.1 Vorticity-related variables computation

- Right-click on the model.

- Select *Secondary Variables* in the *Compute* submenu.

In the Properties area of the compute module you can see the **Category** port that lists the main categories of the secondary variables Avizo can compute.

- Select the *vorticity* category.

In the **Variable** port, several vorticity related quantities that can be computed from the velocity vector field are listed. The velocity vector field has been retrieved by Avizo and set by default in the **Velocity** port. The vorticity related variables might be used to find vorticity regions, by plotting cross sections or by delimiting regions with isosurfaces for example.

Some examples of the most common criteria that can be computed and used to identify vortices:

- high vorticity regions,
- high enstrophy regions,
- non-zero helicity regions...

- Now select the *turbulence* category.

As previously, several turbulence related quantities are listed. They might also be used to find voriticity regions.

Some examples of the most common criteria that can be computed and used to identify vortices:

- negative lambda2 regions,
- positive Q criterion regions...

Example 1: vorticity magnitude

- Select the *vorticity* category.
- Select the *vorticity magnitude* variable.
- Click Apply to compute.

A `VorticityMagnitude` scalar field is created. Visualize it with a *Cross Section*:

- Select *Cross Section* in the *Display* right-click submenu of `VorticityMagnitude`.
- Select *node values* in the *Value Mapping* port.
- Set the orientation to yz.
- Translate the plane to 52.
- Go back to `VorticityMagnitude` and set the upper value of the colormap range in its Properties area to 1000 in order to highlight the regions with high vorticity.
- Select *physics.am* colormap. Load it if it doesn't appear in the list of colormaps (`data/colormaps/physics.am`).

- Display a legend.
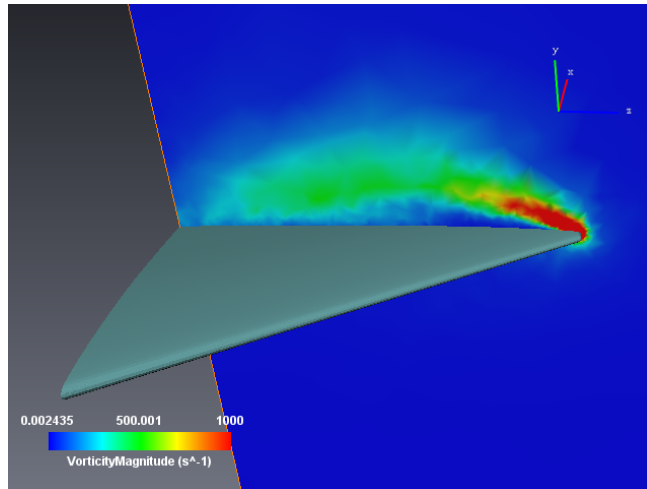
Figure 11.98: Vorticity magnitude close to the aircraft.

Example 2: lambda2

- Hide the *Cross Section*.
- Attach a new *Secondary Variables* to the model.
- Select the *turbulence* category; *lambda 2* is selected by default.
- Click Apply.
- Connect an *Isosurface* to the new `LambdaTwo` object.
- Set the **Isovalue** of the *Isosurface* module to -500.
- Inside *Display Options* port, choose *Data Mapping* in the *Coloring* port.
- Inside *Optional Connections* port, set the *Colorfield* port to `VorticityMagnitude`.
- Select *node values* in the *Value Mapping* port.

The *Isosurface* of `LambdaTwo` isolates a region with negative lambda2 where there are likely to be vortices. Sub-regions with high vorticity are located close to the wing (see Figure 11.9.1).

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_secondaryvariables.hx.

11.9.2   Vortex core lines identification

The *Vortex Corelines* module retrieves the lines around which the flow swirls.
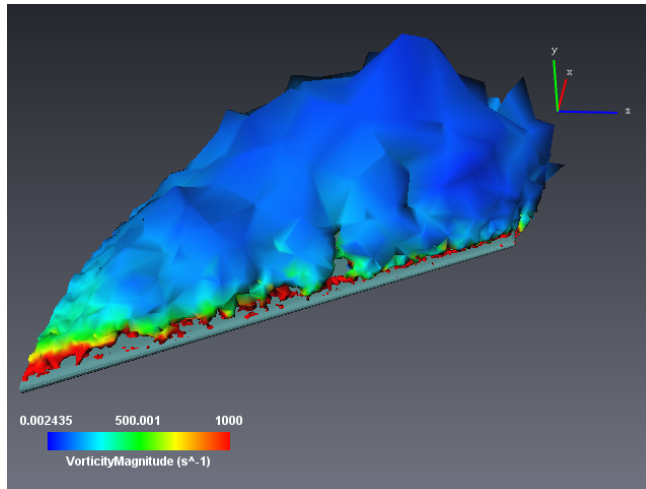
Figure 11.99: Lambda2 = -500 isosurface colored by vorticity magnitude.

- Hide the *Isosurface*, the *Cross Section* and the *Data Legend*.
- Plot *Illuminated Streamlines* using a *ROI Box* positioned close to the leading edge of the wing as explained in Chapter 11.7.
- Right-click on the `Velocity` vector field.
- Select *Vortex Corelines* in the *Compute* submenu.
- Click Apply.

A *Line Set* has been created in the *Models* directory, under the name `VortexCorelines`. We will now display the result.

- Select the `VortexCorelines` object and connect a *Line Set View* in the *Display* submenu.

You can see that there are some very small lines and some noisy lines. If we want to focus only on the main core line, we should filter those lines. Filtering tools are provided for this purpose in the *Vortex Corelines* module.

- First we will remove lines that are too small : set the **minimum line size** to 35.
- Click Apply.

The *Line Set View* is updated. All lines with less than 35 core points are deleted. There remains three lines among which one is obviously outside of the previously plotted lambda2 isosurface.

- Select the *lambda 2* criterion in the **Post-filtering** port.

- Set the **Lambda 2 threshold** to -500.
- Click Apply.

Again, the *Line Set View* is updated. All core points where lambda2 is bigger than -500 are removed, that is to say all points outside of the volume delimited by the *Isosurface* previously studied. The filtering has been effective and there remain only the core line of the illuminated streamlines swirls. If you select the *VortexCorelines* line set, you will see in its Properties area that there are actually two lines composed of a total of 115 core points.

- Click several times on the *smooth* button of the **Line Set** port if you want the line to appear smoother.
- Select the *Line Set View* module.
- Choose *Circle* in the **Shape** port.
- Set the **Scale Factor** to 0.02.
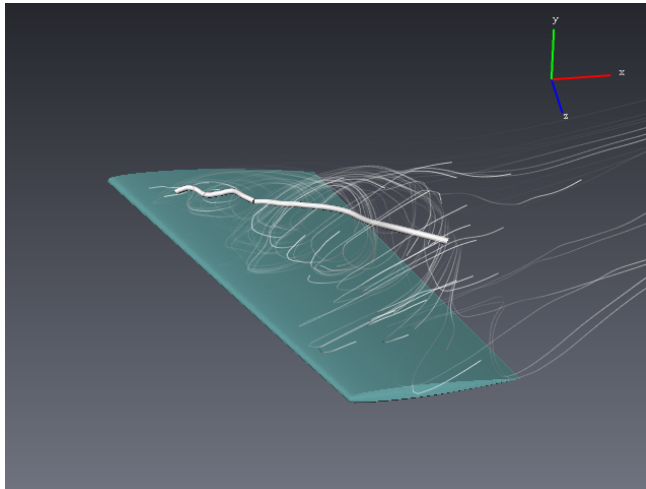
Figure 11.100: Filtered core line and the swirling flow close to the wing.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_vcl.hx.

You can complete this visualization with the display of the 3D critical points (see Chapter 11.7). The illuminated streamlines seeded from them (with the *show* option) swirl around the core line.

11.9.3   Vortical flow visualization

Here are some more visualization modules that can be useful for highlighting some flow behaviors such as the vortical ones under study here.

Surface Illuminated Streamlines

This module sparsely seeds streamlines on a surface in order to display the surface vector field using illuminated streamlines (ISLs). If we want to display the streamlines on the wing, we first need to extract this surface.

- Hide or delete *Illuminated Streamlines* or *Critical Points* that might remain in the Project View.
- Select the *Boundary View*. Only the *wing* should be selected in the **Boundaries** port.
- Press the *create* button in the **Create surface** port. A `wing.surf` surface is created and added to the Project View.
- Right-click on the new surface and select *Illuminated Streamlines Surface* in the *Display* sub-menu.
- Select the *Illuminated Streamlines Surface* module and set the **Vector field** to `Velocity`.
- Uncheck the *early termination* option in **Options** port.
- Set the **Seed** port to 1.
- Press Apply.
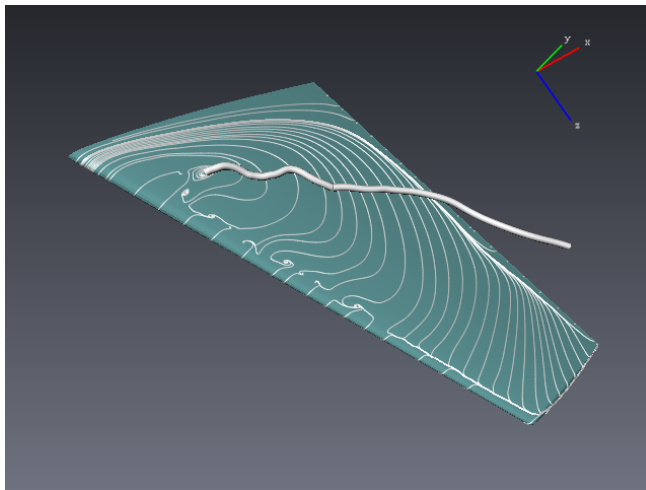
Figure 11.101: Surface illuminated streamlines on the wing and main core line.

Streamlines on the wing are displayed and vortical phenomena appear pretty well. A main swirl corresponds to the starting point of the main core line and smaller swirls correspond to small core lines

we noticed in the first core lines display and then filtered.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_surfaceISL.hx.

Illuminated Streamlines Slices

This module visualizes the directional structure of a vector field in a cutting plane using illuminated streamlines (ISLs). What would be interesting here is to visualize the ISLs in a plane orthogonal to the core line. To do so, we will use the *Trajectory* module.

- Hide or delete the *Illuminated Streamlines Surface*.
- Create a small ROI Box on the upper side of the wing, around the core line.
- Select *Illuminated Streamlines Slice* in the *Display* submenu of the `Velocity` field.
- Set the ROI Box you just created as **ROI** in the *Clipping Plane* Properties area.
- Hide the ROI.
- Right-click on the *Clipping Plane* and attach a *Trajectory* module to it.
- Select `VortexCorelines` as **Data** in the Properties area of the *Trajectory* module.

The *Clipping Plane* is now orthogonal to the core line. If you use the **Position** slider of the *Trajectory* module, the plane will slide along the core line, remaining orthogonal. As the core line is actually composed of two lines, you have to use the **Line** slider to slide the plane along the other part of the core line.

Tip: You could do the same with a *Stream LIC Slice* module for example. The LIC technique is also a good tool to visualize the swirl of the flow around the core lines.

- Select the *Illuminated Streamlines Slice* module.
- Uncheck the *early termination* option in **Options** port.
- Set the **Resolution** to 200 and the **Separation distance** to 15.
- Press Apply.
- Hide the plane frame from the **Frame** port of the *Clipping Plane*.

You can also animate the streamlines to see them swirl around the core line by selecting *animate* in the **View options** port.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_planarISL.hx.

11.10   Avizo XWind Extension Measurements

This section will give you an overview of the measurement features provided in the Avizo XWind Extension.
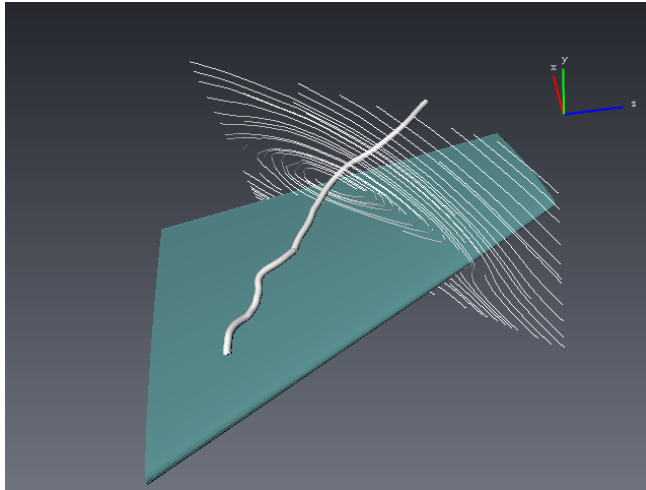
Figure 11.102: Illuminated streamlines on a plane othogonal to the main core line.

- Open `aircraft_mach.cas` from the `tutorials/cfd-fea-advanced` folder.
- Load the `Pressure` scalar field.
- Connect a *Boundary View* to the model.
- Unselect everything except `Wall` in the **Boundary types** port.
- Set the coloring to *Data Mapping* and use `Pressure` as the colorfield.
- Select *node values* in the *Value Mapping* port.
- Hide the *Bounding Box*.

11.10.1   3D measurements

You can access measuring tools via the *View / Measuring* menu or via the measuring tool button
(and its pulldown menu - click on the little arrow) at the top of the viewer.

- Select *Line* in the pulldown menu of the measuring tool button.
- Select *Measuring* in the *View* menu.

You now have a *Measurement* object in the Display folder of the Project View. This module provides
access to two-dimensional and three-dimensional measuring tools.

Line measurement

We will measure the leading edge of the wing. A line measurement (Line) is already selected.

- In the 3D viewer, click on one end of the leading edge of the wing.
  Notice that cursor changes to indicate when a valid object can be selected.
- Click on the other end of the wing edge.
- To adjust the position of a measurement line,
  select it in the Properties area, then click on one of its red handles and drag it to a new location
  or use the text ports **Point 0** and **Point 1** to change the position.
- Do the same on the side edge of the wing.
  Tip: You may need to reposition the camera to select measurement points. As usual you can
  press the [ESC] key to toggle between interactive mode and trackball mode or hold down the
  [Alt] key to temporarily switch to trackball mode.

You can measure that the wing has a leading edge of approximately 4.44 meters and a side edge of
approximately 1.55 meters.

Figure 11.103: Measuring the wing size of the YF-17 aircraft.

3D angle measurement

- In the Properties area of the *Measurement* module, click on the "eye" icons of the two lines to
  hide them in the 3D viewer.
- In the **Add** port, click the *Angle* button.
- In the 3D viewer, click on the intersection of the attack edge of the wing and the fuselage.
- Click on the other end of the attack edge (intersection with the side edge).

- Click on the other end of the side edge.

You can measure that the angle is approximately 116 degrees.
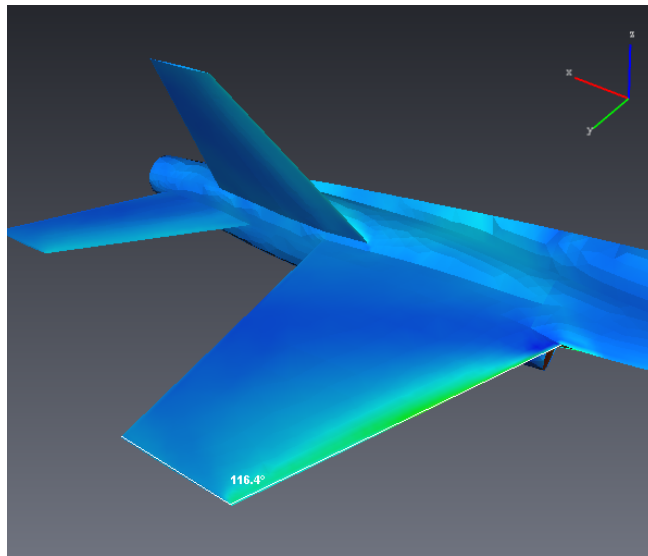
Figure 11.104: Angle measurement.

11.10.2 Histograms

The *Histogram* module computes the histogram of a scalar field in 3D cells. We will use it on the `Pressure` scalar field.

- Right-click on the `Pressure` and select *Histogram* in the *Measure And Analyze* menu.
- Click Apply in the *Histogram* Properties area.

A window pops up, that contains a histogram in logarithmic scaling. The mean value (approx. 1849 Pa) and the standard deviation (approx. 23187 Pa) of the `Pressure` field are displayed in the Properties area.

- Set the **Range** minimum value to 0.
- Activate the **Threshold** and set it to 100000.
- Activate the **Tindex** and set it to 50.
- Click Apply.
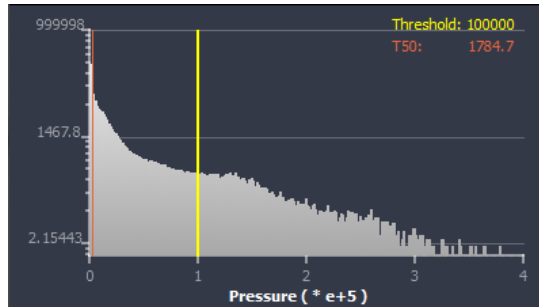
Figure 11.105: Histogram of pressure distribution.

What we can learn is that:

- for all cells where the pressure is in the new range, the mean value is approximately 10647 Pa and the standard deviation is approximately 26442 Pa,
- in this same range, 2.44 percent of the cells have a pressure greater than 100000 Pa,
- in this same range, 50 percent of the cells have a pressure lower than 1784 Pa (and 50 percent greater).

The histogram has been updated.

In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_histo.hx.

11.10.3 Data probing

The three data probing modules *Point Probe*, *Line Probe*, and *Spline Probe* are used to inspect scalar or vector data fields. The probes are taken at a point (*Point Probe*) or along a line (*Line Probe* and *Spline Probe*) which may be arbitrarily placed.

Probing along a spline

We will use the *Spline Probe* to plot the pressure around the wing of the aircraft. First we have to position properly the four control points of the spline.

- Right-click on `Pressure`.
- In the *Measure And Analyze* menu, select *Spline Probe*.

To position the control points within the bounding box of the given geometry you can either type in the coordinates in the **Points** port (see below) or you can move the points dragger interactively with the mouse. (You may have to zoom out to see the points dragger.)

- In the **Points** port, the coordinates of the first control point are displayed. Change them to 4, 2, 0.
- In the **Points** port, use the spin box to select the second point and set its coordinates to 0, 2, 0.
- Select the third point and set its coordinates to 0, 2, 0.3.
- Select the fourth point and set its coordinates to 4, 2, 0.3.
- You might want to hide the points and the dragger using the *options* submenu of the **Points** port.
- Click the *Show* button in the **Plot** port.

Figure 11.106: Pressure values against the spline probe line length.

A plot window appears where the sampled pressure values are plotted against the length of the probe line. In case of any problems or uncertainties you can find the same project predefined in your tutorial folder under the file name data/tutorials/cfd-fea-advanced/wind_splineprobe.hx.

Probing along a surface path

For probing purposes, it is often useful to have tools to define specific lines on a surface. The *Surface Path Editor* and the *Surface Intersector* module are designed to this end.

The *Surface Path Editor* allows creating paths on surfaces. Paths can be useful to cut surfaces, define regions or features of a surface, probe, etc. The editor can be accessed from the Properties area of a

Surface Path Set by clicking on the editor button . Two types of editor are then provided:

- the Generic Path Editor allows defining paths arbitrarily across the surface mesh,
- the Vertex Path Editor allows defining paths only along the surface mesh edges.

Note that the Vertex Path Editor can be accessed directly from the *Mesure And Analyze* submenu of a surface (entry named *Create Surface Vertex Path*).

The *Surface Intersector* module intersects two surfaces, computes a path along the intersection and attaches it to each of the surfaces.

- Remove all objects from the Project View (use [Ctrl+N] or right click in the Project View

and select *Remove All Objects*).

- Open `fan-0070.cas` from the `tutorials/cfd-fea-advanced/fan` folder.
- Load the `Pressure` scalar field.
- Hide the *Bounding Box*.
- Connect a *Boundary View* to the model.
- Unselect everything except `wall-1` in the **Boundaries** port and create the surface from the **Create surface** port.

We will plot the `Pressure` along a radial line section of the fan surface. We have to create a cylindrical surface first, in order to intersect it with the fan and then get the intersection line.

- Right click on the surface `fan-0070.surf` and create a *Surface Intersector* from the *Compute* submenu.

In the *Surface Intersector* Properties area, the second surface still has to be set. We will use the *Parametric Surface* module to create the intersecting surface we need, available from *Project / Create Object...* (*Surfaces And Grids* submenu).

- Create a *Parametric Surface*. A default plane is created.
- For **U**, set *min* to -0.02, *step* to 0.0005, *max* to 0.01.
- For **V**, set *min* to 1, *step* to 0.0005, *max* to 2.
- Set **X** to `u`, **Y** to `0.12*sin(v)` and **Z** to `0.12*cos(v)`.
- Click on the *more options* button in the **Draw style** port and select *Create surface*. `Parametric-Surface.surf` is added to the Project View.
- Set `Parametric-Surface.surf` as the second surface of the *Surface Intersector* and press Apply.

Two paths along the intersection are created, one attached to each of the surfaces.

- Hide the *Parametric Surface* and connect a *Line Set View* display module to `IntersectionPath2`. The path is displayed on the fan surface.
- In the *Measure And Analyze* submenu of `Pressure`, select *Line Set Probe*.
- Attach the *Line Set Probe* to `IntersectionPath2` in the *Line set* port and press the *Show* button.

A window displaying the `Pressure` along the line probe appears. We will improve the display.

- For *X-Axis*, choose the z coordinate.
- In the *Edit* menu, select *Edit Objects*.
- In the *axis* section, deselect the *Auto* option that adjusts the range to the X range and set it to [-0.025, 0.04].

- Change the Y label to "Pressure".
- In the *LineSetProbe_001* section, change the *Draw style* to *Marker*.
- Change the markers shape (e.g. to dots) and color.
- Change the label to "Radial section: 0.12m".
- Press OK.

Figure 11.107: Pressure values along the radial 0.12m line section of the fan surface.

Chapter 12

Avizo XLabSuite Extension User's Guide

Avizo XLabSuite Extension provides numerical simulation capabilities to calculate physical properties of materials from the 3D image of a sample (for instance, scanned with CT, FIB/SEM, MRI, etc.). Material properties are directly computed from the segmented 3D image.

The calculated materials properties are:

- absolute permeability,
- molecular diffusivity,
- formation factor and electrical conductivity,
- thermal conductivity.

Note: see section System Requirements about system requirements and hardware platform availability.

For each of these properties, two different modules are available, corresponding to two different simulation approaches. The first approach consists of estimating a given property as the result of an experiment performed in a laboratory. To do that, the external conditions of an experiment are simulated by imposing boundary conditions resembling those existing in a laboratory. This way it is possible to compare the module results to actual experimental results. The second approach is an effective property calculation. In that case, the material is considered as representative of an infinite medium from which it is extracted. By imposing spatially periodic boundary conditions, it is possible to obtain the effective property of the macroscopic medium.

Experiment simulation is designed to be close to realistic laboratory experiment. It considers very perturbing boundary conditions: the sample is hermetically closed on four faces and constant values are imposed on the remaining two. The transport phenomenon is highly constrained by these kinds of

conditions.

Tensor calculation is based on a mathematical approach which considers the sample to be representative of an infinite or macroscopic material. Periodicity is a lot softer boundary condition and the transport phenomenon is more free. Tensor calculation is usually the preferred method when the Representative Elementary Volume is reached.

Provided modules are:

- *Absolute Permeability Experiment Simulation*
  Simulation of an experiment, by hermetically closing a given sample on four faces while experimental setups are added on two opposite faces to guide the flow along one direction.

- *Absolute Permeability Tensor Calculation*
  Calculation of the intrinsic permeability tensor, by imposing periodic boundary conditions on a representative elementary volume.

- *Molecular Diffusivity Experiment Simulation*
  Fick's second equation is solved under laboratory conditions to simulate an experiment.

- *Molecular Diffusivity Tensor Calculation*
  The diffusivity tensor is computed from a volume averaging method applied to Fick's second equation. The studied sample is considered representative of a larger scale material, allowing the imposition of periodic boundary conditions.

- *Formation Factor Experiment Simulation*
  Ohm's equation is solved under laboratory conditions to simulate an experiment.

- *Effective Formation Factor Calculation*
  The conductivity tensor is computed from a volume averaging method applied to Ohm's equation. The studied sample is considered representative of a larger scale material, allowing the imposition of periodic boundary conditions. The formation factor is deduced.

- *Thermal Conductivity Experiment Simulation*
  Fourier's equation is solved under laboratory conditions to simulate an experiment.

- *Thermal Conductivity Tensor Calculation*
  Calculation of the thermal conductivity tensor, by imposing periodic boundary conditions on a representative elementary volume.

The following sections provide, for each property, an overview of the theory on which the computation modules are based, and tutorials for getting started with the modules.

- *Getting started with absolute permeability computation*
- *Getting started with molecular diffusivity computation*
- *Getting started with formation factor and electrical conductivity computation*
- *Getting started with thermal conductivity computation*

In the tutorials, some steps are **mandatory**: you must follow them for successful completion of the

tutorial. Other steps are **optional**: you should at least read them, because their content could be useful later. Whether a step is optional or mandatory is indicated at the beginning of each section of the tutorial.

Acknowledgements

12.1 Getting started with absolute permeability computation

The purpose of this step-by-step tutorial is to increase your familiarity with the use of the absolute permeability computation modules provided with the Avizo XLabSuite Extension for Avizo. The following subjects will be addressed in this chapter:

- *theory basics about absolute permeability*
- *data preparation for simulation (step 1 to step 6)*:
    - setting voxel size and units
    - segmenting void space
    - removing non-percolating space with Avizo
    - defining a region of interest
- *experiment simulation (step 7)*: define parameters, run simulation, interpret and visualize results
- *effective property calculation (step 8)*: define parameters, run simulation, interpret and visualize results
- *validate results with the Kozeny-Carman equation (step 9)*.

A demo script is also provided. This script will automatically perform all the steps detailed in the tutorial.

12.1.1 Theoretical elements

Darcy's law: definition of absolute permeability
Absolute permeability is defined as the measure of the ability of a porous material to transmit a single-phase fluid. Its SI unit is square meter (m^2), but square micrometer (μm^2) is more common since it is almost equivalent to one darcy (d): $1d = 0.9869233\mu m^2$. It is an intrinsic property of a material, independant of any external condition.

Absolute permeability appears in Darcy's law (see [1]) as a constant coefficient relating fluid, flow and

material parameters:

$$\frac{Q}{S} = -\frac{k}{\mu}\frac{\Delta P}{L}$$

where:

- Q is the global flow rate that goes through the porous medium (unit: $m^3.s^{-1}$);
- S is the cross section of the sample which the fluid goes through (unit: m^2);
- k is the absolute permeability (unit: m^2);
- μ is the dynamic viscosity of the flowing fluid (unit: $Pa.s$);
- ΔP is the pressure difference applied around the sample (unit: Pa);
- L is the length of the sample in the flow direction (unit: m).

$\frac{Q}{S}$ is often noted v and accounts for the superficial or mean fluid flow velocity through the porous medium or Darcy's velocity.

Only single-phase fluids are considered for absolute permeability. Multi-phase flows are concerned by relative permeability.

Stokes equations and flow conditions

To numerically estimate absolute permeability, the Stokes equations are solved:

$$\left\{\begin{array}{rcl} \overrightarrow{\nabla}.\overrightarrow{V} & = & 0 \\ \mu\nabla^2\overrightarrow{V} - \overrightarrow{\nabla}P & = & \overrightarrow{0} \end{array}\right.$$

where:

- $\overrightarrow{\nabla}.$ is the divergence operator;
- $\overrightarrow{\nabla}$ is the gradient operator;
- $\overrightarrow{V}$ is the velocity of the fluid in the fluid phase of the material;
- μ is still the dynamic viscosity of the flowing fluid;
- ∇^2 is the Laplacian operator;
- P is the pressure of the fluid in the fluid phase of the material.

This equation system is a simplification of the Navier-Stokes equations, considering:

- an incompressible fluid, which means that its density is a constant;
- a Newtonian fluid, which means that its dynamic viscosity is a constant;
- a steady-state flow, which means that velocity does not vary over time;
- a laminar flow, which means that the concerned velocities are small enough not to produce turbulence.

The last point is equivalent to considering flow at a low Reynolds number (see [2] for the first appearance of Reynolds numbers).

Once this equation system is solved, estimating the permeability coefficient consists of applying Darcy's law. All the values of this equation can be deduced from the solution of the equation system $(Q, \Delta P)$ or are external conditions (S, L, μ). It is computed by the *Absolute Permeability Experiment Simulation* module.

Volume averaged form of the Stokes equations

The *effective* permeability can be defined as the influence of the solid phase on the velocity of the fluid. A change of scale is necessary to get equations valid on the entire volume. The method of volume averaging is a technique that accomplishes a change of scale. Its main goal is to spatially smooth equations by averaging them on a volume. The interested reader will find relevant details and references in [3].

This very general theory leads to develop a closure problem which transforms the Stokes equations into a tensorial problem. It remains very similar to the Stokes equations, despite the fact it is a higher order problem:

$$\begin{cases} \vec{\nabla}.\overrightarrow{\overrightarrow{D}} &= \vec{0} \\ \nabla^2 \overrightarrow{\overrightarrow{D}} - \vec{\nabla}\vec{d} &= \overrightarrow{\overrightarrow{I}} \end{cases}$$

where:

- $\overrightarrow{\overrightarrow{D}}$ is a tensor that can be considered as the source of the spatial deviation of the velocity, which we'll refer to as velocity perturbation field;

- $\vec{d}$ is a vector that can be considered as the source of the spatial deviation of the pressure, which we'll refer to as pressure perturbation field;

- $\overrightarrow{\overrightarrow{I}}$ is the unit tensor.

The permeability tensor is extracted from the solution of this problem by calculating the mean value of $\overrightarrow{\overrightarrow{D}}$ over the volume V on which the system was solved:

$$\overrightarrow{\overrightarrow{k}} = \frac{1}{V} \int_V \overrightarrow{\overrightarrow{D}} dV$$

This permeability tensor gives additional information about the intensity of permeability along any direction of space. It can give rise to the anisotropy of a porous medium, i.e., the dependence of the permeability intensity on the direction of the flow. It is computed by the *Absolute Permeability Tensor Calculation* module.

Boundary conditions

Avizo XLabSuite Extension provides two approaches to estimate absolute permeability.

The first is an experiment simulation based on Stokes equations resolution; this is done in *Absolute Permeability Experiment Simulation* module. The boundary conditions are specified as:

- a no-slip condition at fluid-solid interfaces.
- one-voxel-wide plane of solid phase (with no-slip condition) is added on the faces of the image that are not perpendicular to the main flow direction. This allows isolation of the sample from the outside, allowing no flow out of the system.
- experimental setups are added on the faces of the image that are perpendicular to the main flow direction. They are designed in a manner that creates a stabilization zone where pressure is quasi static, and the fluid can freely spread on the input face of the sample.
- two among the following three conditions can be chosen by the user, the third being estimated from the chosen two: input pressure, output pressure, flow rate.

The second approach solves the closure problem derived from Stokes equation by volume averaging. This is done in the *Absolute Permeability Tensor Calculation* module. The tensorial problem that is solved in this case is closed by imposing periodic boundary conditions to $\overrightarrow{\overrightarrow{D}}$, $\overrightarrow{d}$ and the geometry. A no-slip condition is imposed at the fluid-solid interfaces. The sample represents a macroscopic, infinite material and must thus be representative of this porous medium.

Artificial compressibility

The equation systems cannot be solved using fully implicit methods (matrix inversion) because the matrices of this kind of system are singular. This is why an artificial compressibility coefficient and some time derivative terms are introduced in the system. The method of artificial compressibilty was first described in [5].

Introducing these terms in the equation system allows an iterative resolution of the problem. The unique solution is attained when the time derivatives tend to zero. The time that is introduced in the equations has no physical sense.

Discretization of the equation system

Avizo XLabSuite Extension uses a finite volume method to solve the equation systems. The equations are discretized on a staggered grid (proposed by [6]), allowing a better estimation of the no-slip boundary condition. Pressure unknowns are located at the center of the voxel while velocity unknowns are decomposed at the faces of the voxels.

The discretization scheme assumes that the voxel is isotropic (cubic).

Bibliography

1. Darcy, H., *Les fontaines publiques de la ville de Dijon* , V. Dalmont, Paris, 1856
2. Reynolds, O., *An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels* ,

Philosophical Transactions of the Royal Society 174 (0): 935-982, 1883

3. Whitaker, S., *The Method of Volume Averaging*, Kluver Academic Publishers, 1999

4. Gray, W. G., *A derivation of the equations for multiphase transport* , Chemical Engineering Science, 30, 229-233, 1975

5. Chorin, A. J., *A Numerical Method for Solving Incompressible Viscous Flow Problems* , Journal of Computational Physics, 2, 12-26, 1967

6. Harlow, F. H., and Welch, J. E., *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Physics of Fluids, v.5, p.317, 1965

12.1.2   Step 1 - Activate Avizo unit management

Purpose	Activate the unit management provided in Avizo.
Mandatory step	Yes.

The length unit is one of the most important parameters to scale correctly in order to get accurate permeability results. It is stored in the Avizo format, for example, but not all data formats save this information. Avizo provides a length unit management tool that must be activated for this tutorial.

- Start a new Project *File > New Project*
- Open the *Edit > Preferences...* dialog.
- Select the *Units* tab.
- Select *Spatial information only*. Keep all the boxes checked (see 12.1).
- Click on *OK*.
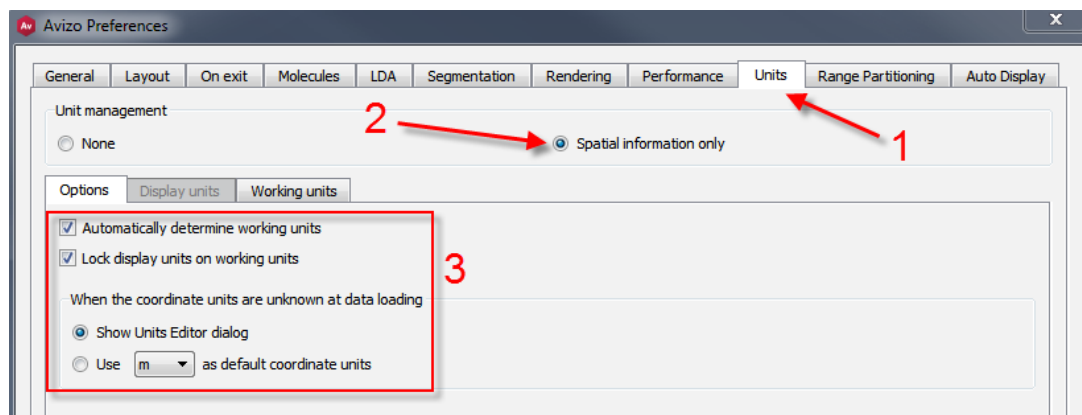
Figure 12.1: (1) In the *Edit > Preferences...* dialog, select the *Units* tab. (2) Select *Spatial information only* and (3) keep all the boxes checked.

Once activated, the unit management tool will ask for the length unit to use for voxel size whenever loading a data set. For more details about this feature, please refer to the *Units in Avizo* chapter in the reference guide.

12.1.3   Step 2 - Load the data set and select the length unit for voxel size

Purpose	Load the data set in Avizo and set the unit length.
Mandatory step	Yes.

The data set that is used in this tutorial is a random packing of glass spheres. It was scanned by Dominique Bernard, Research Director at ICMCB-CNRS. The spherical particles were sieved to have diameters in the 100-120 microns range. The sample was sintered for 10 minutes at 700° Celsius prior to scanning. Figure 12.2 shows the data set.

The complete 3D data set used in this tutorial is a $200 \times 200 \times 200$ cube.

- Open the *File > Open Data...* dialog.
- Open the `data/tutorials/xlab/10mc3_200.vol.am` file from `AVIZO_ROOT` directory (see Figure 12.3).
- Click on *Open*.

Note: The file `10mc3_200.vol.am` has embedded unit information, therefore length unit is automatically set for this data. However, the file `10mc3_400.vol.am` doesn't have such unit information. Since the Avizo unit management tool was activated at step 1, you will need to specify the length unit every time the file is opened. The dialog presented by Figure 12.4 will appear each time a data set without length unit information is loaded. For instance, for the file `10mc3_400.vol.am`, please select *micrometer [μm]* as the coordinate units for the data set of the tutorial.

Note: If a data set is saved after the length unit has been selected, this information is stored in Avizo format and will be reused at its next loading. Length unit then can be modified using the *Units Editor* (see Figure 12.5) and a dialog box appearing, which looks like Figure 12.4.
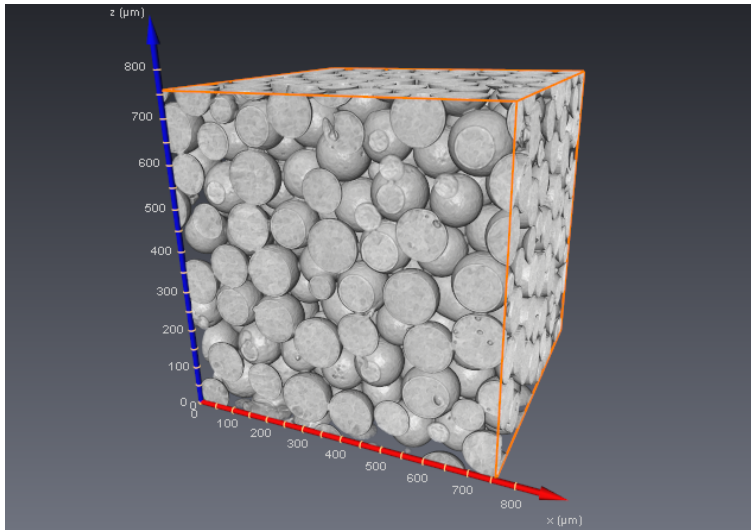
Figure 12.2: 3D visualization of the sphere pack used in this tutorial.
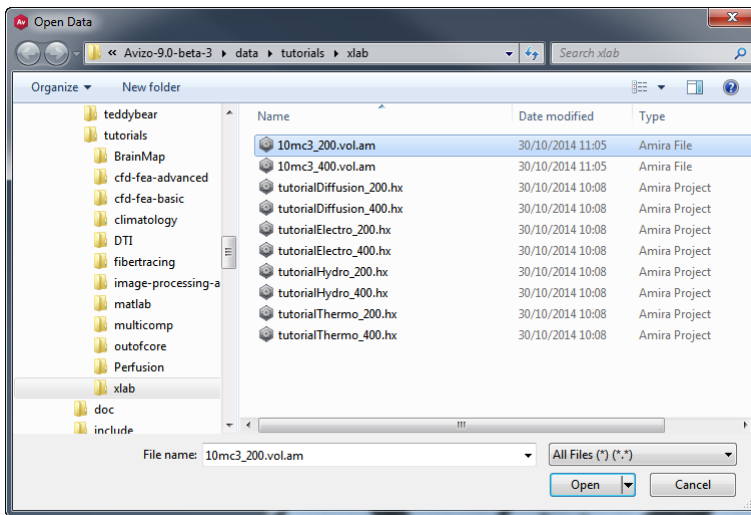
Figure 12.3: *Open Data...* dialog to load the data set of this tutorial. Choose the path to the data set, then select and open it.
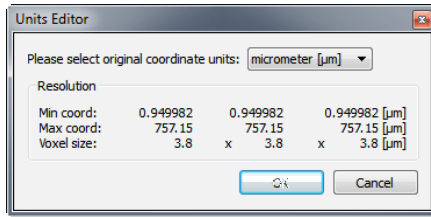
Figure 12.4: *Units Editor* dialog. Use the drop-down menu to select the length unit for voxel size. This dialog will appear each time a data set without length unit information is loaded.
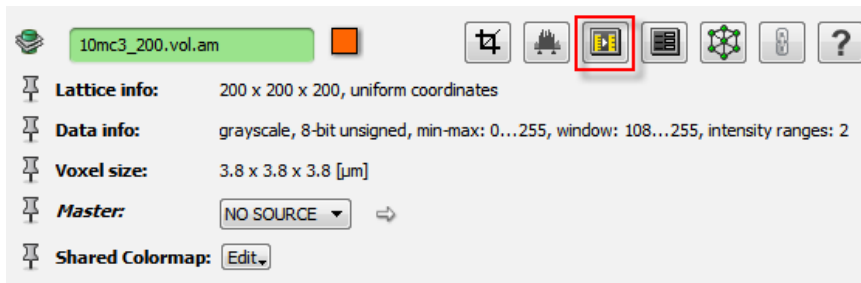
Figure 12.5: The *Units Editor* button is in the red square (visible only when a data set is selected in the Project View).

12.1.4 Step 3 - Set the voxel size

Purpose	Set the edge size of the voxels in the uniform data set.
Mandatory step	Yes.

Let's first display the loaded data set:

- Right-click on `10mc3_200.vol.am` and select *Bounding Box*.
- Use the *Ortho Slice* module to navigate through the volume.
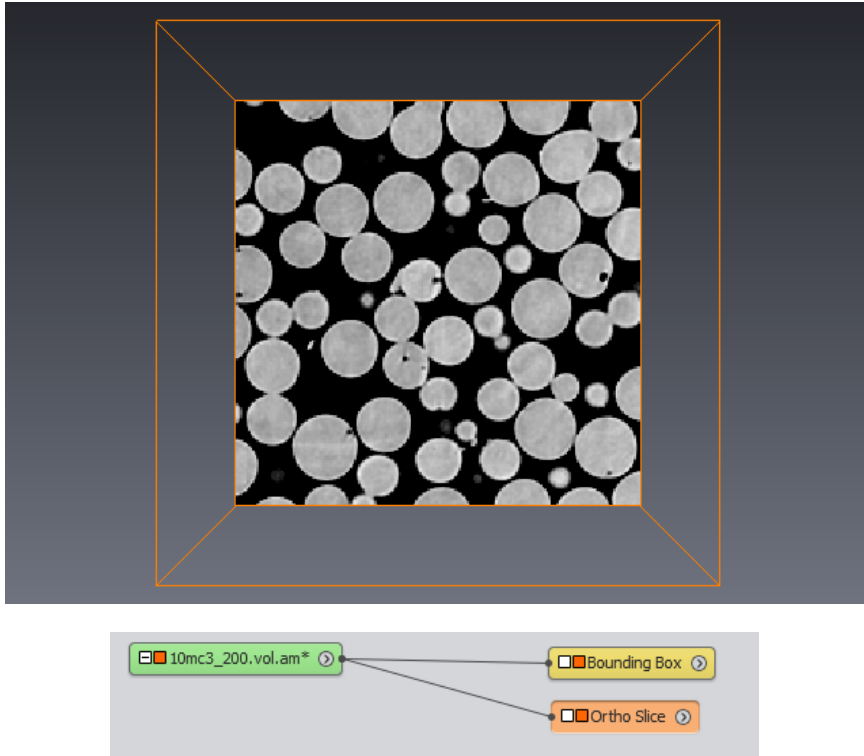
Figure 12.6: At beginning of step 4, after having connected visualization modules, the viewer and the Project View should look like in this picture.

The voxel size is stored in Avizo format, for example, but not all data format save this information. In case the stored values are wrong, Avizo provides an editor allowing modification of the voxel size of the loaded data set.

- Open the *Crop Editor* by clicking on the highlighted button in Figure 12.7.
- The voxel size of the data set can be modified in the dialog that appeared (see Figure 12.8).
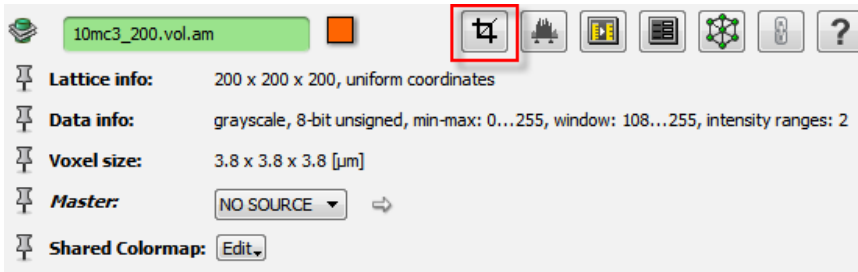
Figure 12.7: Finding the *Crop Editor* button when the data set is selected in the Project View.
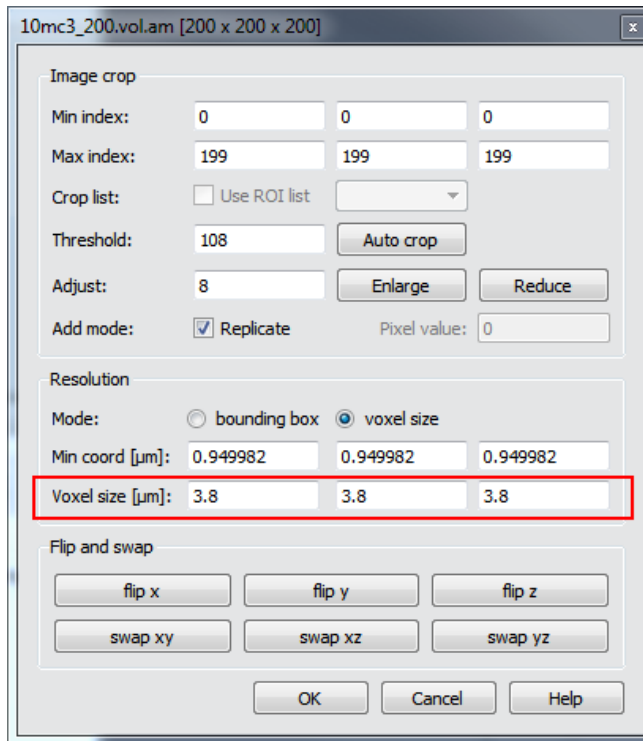
Figure 12.8: Fields to fill to modify the voxel size in the *Crop Editor*.

For the data set of this tutorial, the correct voxel size is 3.8, which is set by default.

If the Avizo unit management is not activated, the voxel size indicated in the data set will be considered to be in microns. Otherwise, the length unit was set at data set loading time.

Note: Isotropic, i.e., cubic, voxels are mandatory for Avizo XLabSuite Extension computations.

12.1.5 Step 4 - Create a label image from the data set

Purpose	Use basic segmentation tools to create a label image from the data set.
Mandatory step	Yes.

Segmentation in this tutorial is straightforward, since the data set that is used is almost binary from the beginning. As it is not the focus of this tutorial, please refer to the *Segmentation of 3D Images* chapter in the user's guide for more details about segmentation tools in Avizo.

Apply a fast smoothing filter to remove the noise in the image.

- Right-click on `10mc3_200.vol.am` in the Project View and select *Image Processing > Smoothing and Denoising > Median Filter* .
- Set the *Filter* interpretation to *3D* (see Figure 12.9).
- Click on *Apply*.
- Right-click on the result of the filtered image (`10mc3_200.vol.filtered`) in the Project View and select *Display > Ortho Slice* (see Figure 12.10).
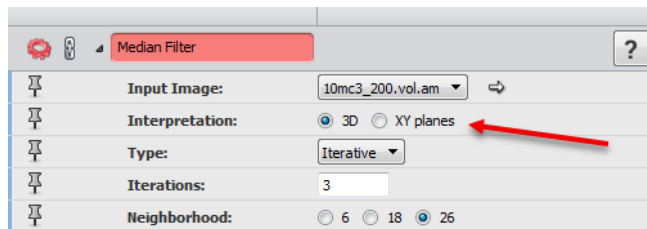
Figure 12.9: Modified parameter of the *Filter > Median Filter* module is highlighted by a red arrow.
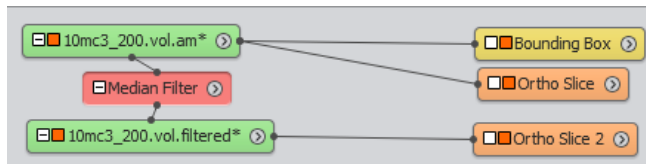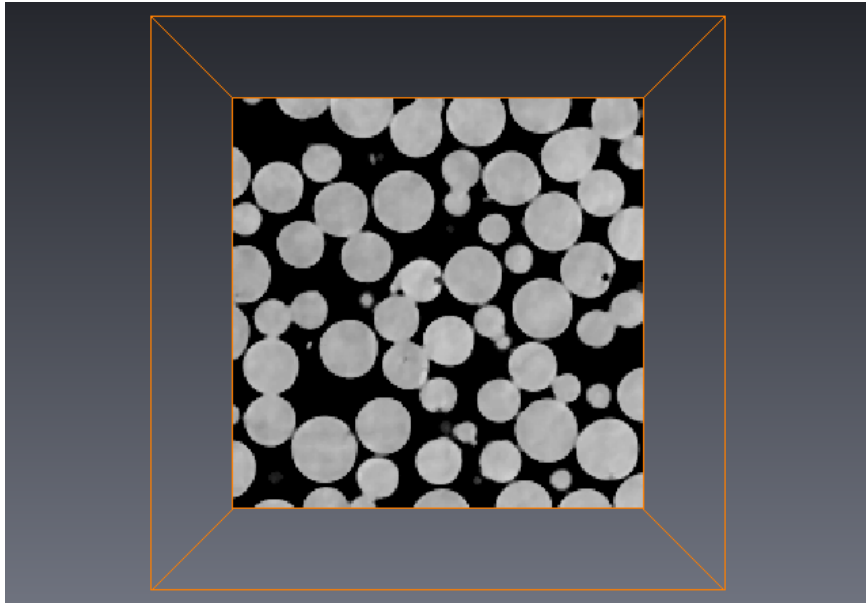
Figure 12.10: Visualization of the filtered data set. The viewer and Project View should look like in this picture.
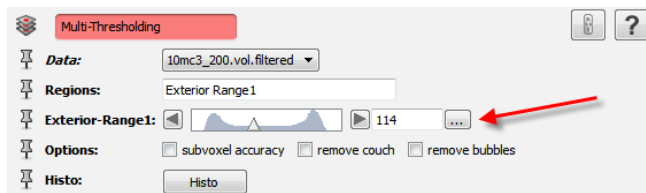
Figure 12.11: Modified parameter of the *Segment > Multi-Thresholding* module is highlighted by a red arrow.
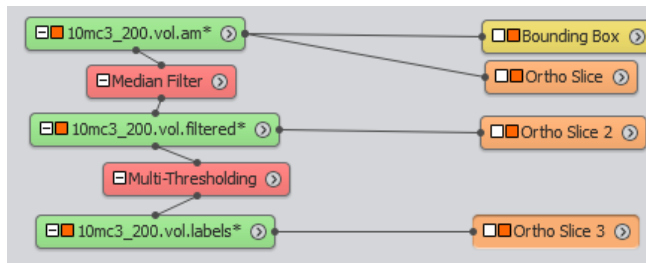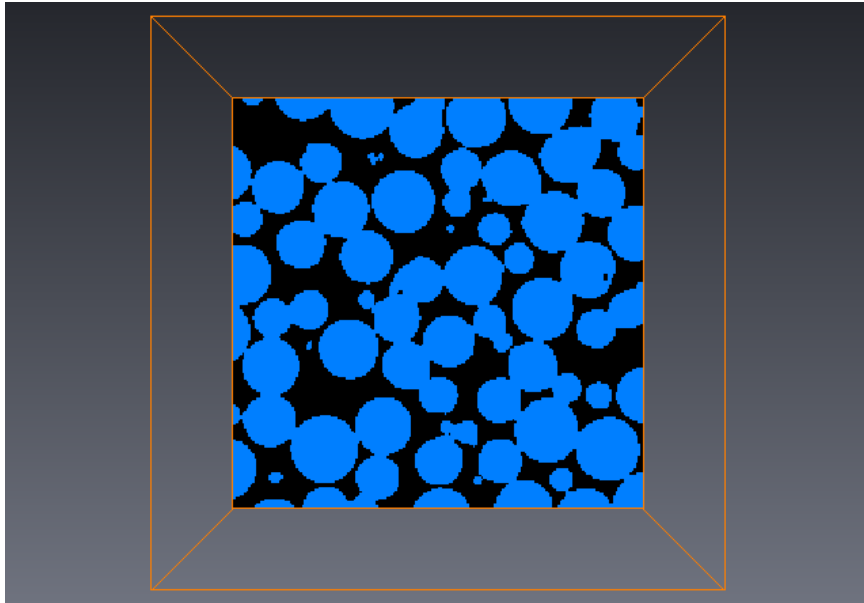
Figure 12.12: Visualization of the thresholded data set. The viewer and Project View should look like in this picture.

Create a label image by thresholding the smoothed image.

- Right-click on `10mc3_200.vol.filtered` in the Project View and select *Image Segmentation > Multi-Thresholding*.
- The default option should be the same as in Figure 12.11. The most important parameter to verify is the value of the *Exterior-Range1* slider, which must be set to 114.
- Click on *Apply*.
- Right-click on the result of the threshold operation (`10mc3_200.vol.labels`) in the Project View and select *Display > Ortho Slice* (see Figure 12.12).

12.1.6 Step 5 - Remove non-percolating void space

Purpose	Use advanced segmentation tools to remove non-percolating void space in the thresholded image.
Mandatory step	No. Only available with Avizo.

A preliminary test that can be performed prior to permeability computation is a percolation test. The aim is to be sure that the void space existing in the material allows a fluid going from one side to another. If this test is wrong, it means that a fluid injected through one of the faces of the sample cannot find any path to go out on the opposite face. Permeability of such a material is nil, but numerical convergence until 0 can be rather long. The necessary connectivity, from a discrete point of view, in 6-connectivity: a fluid can go from one voxel to its neightbor only through a voxel face.

First, the spheres are 1-valued, the void is 0-valued. For the percolation process to be done, these values must be negated.

- Right-click on `10mc3_200.vol.labels` in the Project View and select *Compute > Logical Operations > Invert*.
- Click on *Apply*.
- Right-click on the result (`10mc3_200.vol.invert`) in the Project View and select *Display > Ortho Slice* (see Figure 12.13).
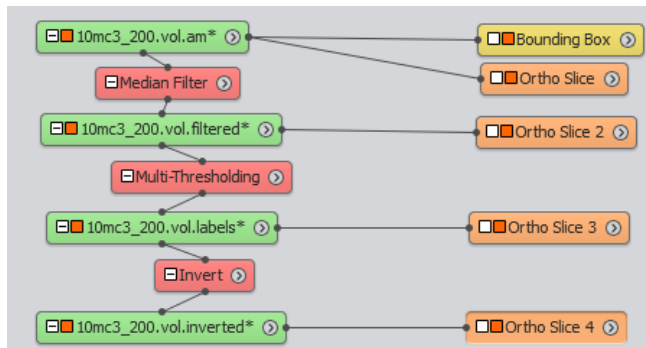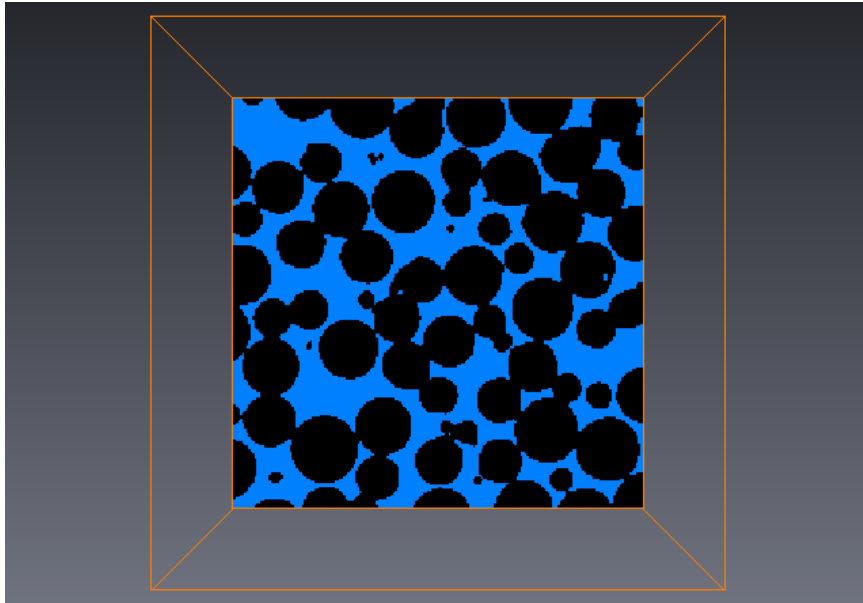
Figure 12.13: Visualization of the negative data set. The viewer and Project View should look like in this picture.

The percolation test can be applied to this negative image.

- Right-click on `10mc3_200.vol.invert` in the Project View and select *Image Processing > Propagation > Axis Connectivity*.
- Select *6* in the *Neighborhood* port.
- Click on *Apply*.
- Right-click on the result (`10mc3_200.vol.Axis-Connectivity`) in the Project View and select *Display > Ortho Slice* (see Figure 12.14).
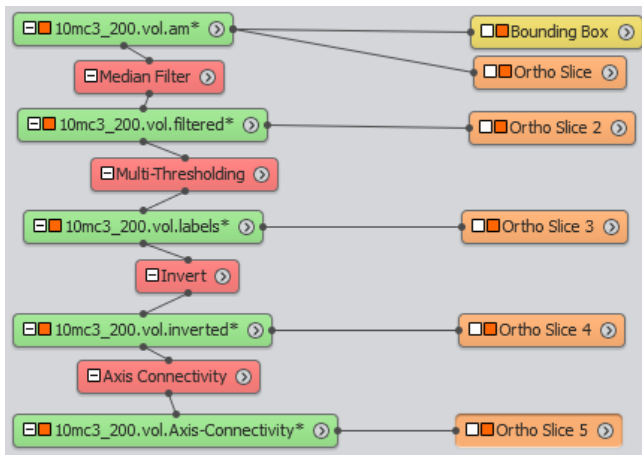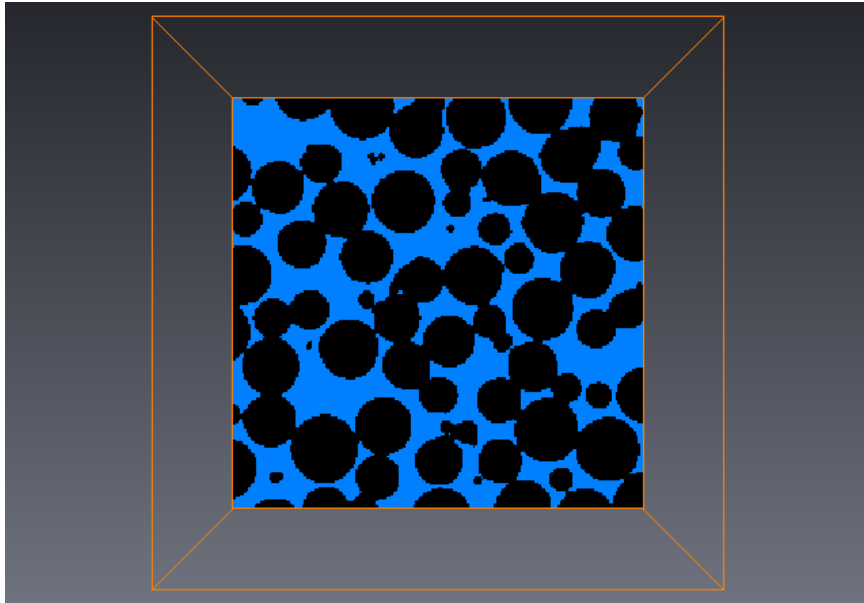
Figure 12.14: Visualization of the percolating porosity in data set. The viewer and Project View should look like in this picture.

This image contains all the 1-valued voxels of the void space that can be reached starting from the plane $z = 0$ and moving only in 1-valued voxels until plane $z = 200$. The isolated void spaces are removed, which makes the computation to follow a little faster. The usefulness of this test is increased when the resulting image is empty. It means that there is no connected porosity, and permeability is nil.

12.1.7 Step 6 - Selection of a sub-region

Purpose	Define a region of interest for the permeability computation.
Mandatory step	Yes.

There is an easy way to compute permeability on a sub-region of the loaded data set without cropping it. A region of interest (ROI) can be defined and connected to Avizo XLabSuite Extension modules, so that the computation only occurs in the ROI.

- Right-click on `10mc3_200.vol.Axis-Connectivity` in the Project View and select *Display > ROI Box*.
- Set the three editable fields of the *Minimum* port of this new module to 285 and the three editable fields of the *Maximum* port to 473.1. See Figure 12.15.
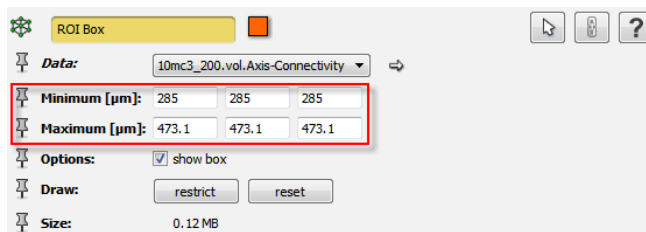
Figure 12.15: Modified parameters of the *Display > ROI Box* module are highlighted by the red rectangle.

This ROI defines a cube with 50 voxel edges centered in the complete volume (see Figure 12.16). It will be used in this tutorial to compute permeability on small volumes. It can be moved through the data set at will during the tutorial, although its initial position will be used for illustrating purposes.
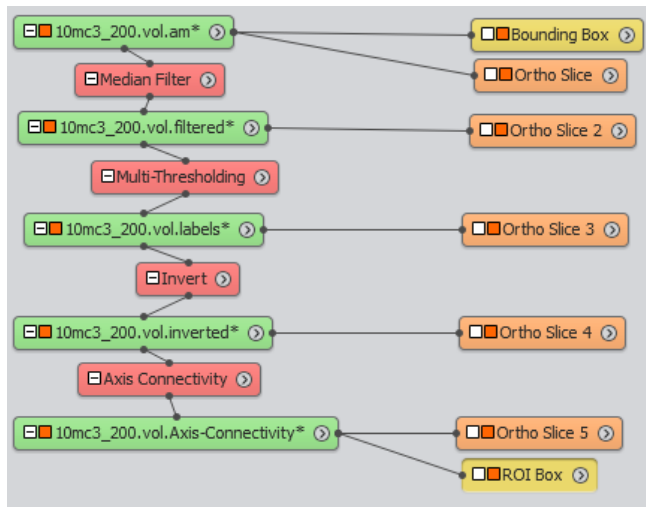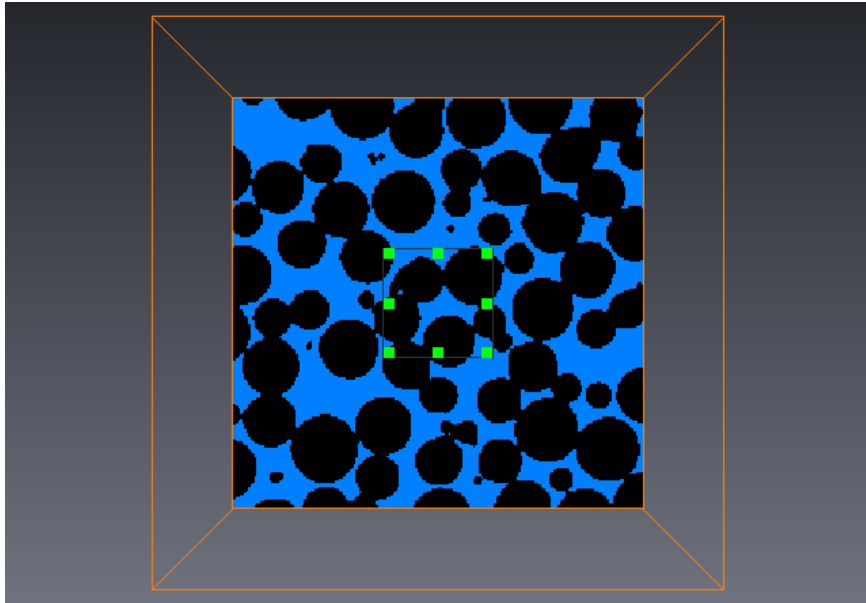
Figure 12.16: Visualization of the cube representing the region of interest. The viewer and Project View should look like in this picture.

12.1.8   Step 7 - Absolute permeability experiment simulation

Purpose	Simulate an absolute permeability experiment.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.labels` if step 5 was not completed) and select *XLab Simulations > Absolute Permeability Experiment Simulation*.
- Connect the *ROI Box* to the ROI input connection of the *Absolute Permeability Experiment Simulation* module.
- If step 5 was completed, check the box with *1* in the *Pore space* port, since the label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 5 was not completed, check the box with *0* in the *Pore space*, since the label of the void space in `10mc3_200.vol.labels` is 0.
- The module parameters should look like Figure 12.17 if step 5 was completed.
- Click on *Apply*.
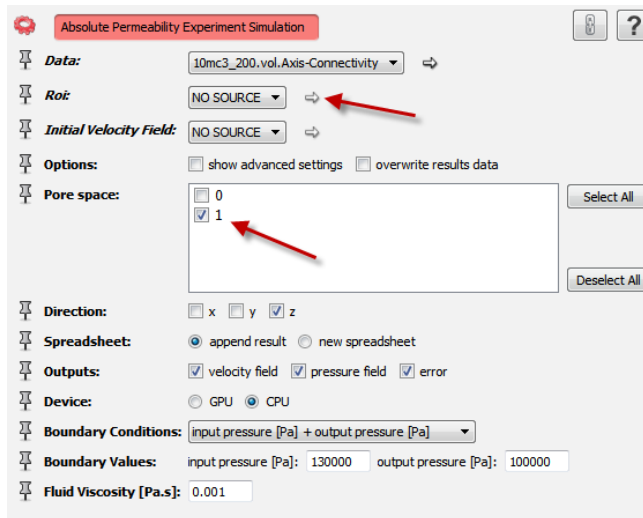
Figure 12.17: Modified parameters of the *XLab Simulations > Absolute Permeability Experiment Simulation* module are highlighted by red arrows. (1) Connect the *ROI Box* module to reduce the computation domain. (2) Select the label 1 as it represents the void space between the spheres.

The default parameters simulate an experiment along the Z axis with the input pressure at 1.3×10^5 Pa and the atmospheric pressure at output. The default viscosity is the viscosity of water. The options

can be modified to simulate several experiments. For example, the direction of the main flow can be adjusted to X, Y or Z direction (default is Z). If several directions are selected, the computations will be done successively. The boundary conditions of the experiment can also be modified, so that the velocity and pressure fields are scaled with these values. Two values among three can be imposed: input pressure, output pressure, flow rate. Modifying these values will not change permeability, which is intrinsic to the porous medium. It will only modify the output fields.

12.1.8.1 Retrieving and interpreting results

Four outputs appeared in the Project View:

- `10mc3_200.vol.KExp.Spreadsheet` : a spreadsheet containing the most relevant results of the computation:
    - name of the data set describing the geometry of the material;
    - region of interest, on which the computation occured;
    - permeability value in μm^2;
    - permeability value in d (darcy).
- `10mc3_200.vol.KExp.Error.Spreadsheet` : a spreadsheet containing the estimation of the convergence criterion at each iteration. The curve of convergence criterion w.r.t. iterations is displayed by `Plot 10mc3_200.vol.KExp.Error.Spreadsheet`.
- `10mc3_200.vol.VelocityZ` : a vector field representing the velocity field which is a part of the solution to the Stokes equation system solved in the geometry defined by the ROI. Its unit is $\mu m.s^{-1}$.
- `10mc3_200.vol.PressureZ` : a scalar field representing the pressure field which is the second part of the solution of the Stokes equation system solved in the geometry defined by the ROI. Its unit is Pa.

The spreadsheet containing all results and related information can be visualized by selecting it in the project view and clicking on the *Show* button (in the *Properties area*). See Figure 12.18.

	10mc3_200.vol.KExp.Spreadsheet		
	Geometry file	Region of interest	k [um^2]	k [d]
1	10mc3_200.vol....	285 285 285 / 47...	3.9243627	3.9763603

Figure 12.18: Spreadsheet containing the main results of the *Absolute Permeability Experiment Simulation* computation.

A spreadsheet can be exported into several formats (CSV, XML, txt for example).

12.1.8.2 Visualizing output fields

To visualize the velocity field:

- Hide *Bounding Box* and the five *Ortho Slice* by clicking on their viewer toggle.
- Right-click on `10mc3_200.vol.VelocityZ` and select *Compute > Magnitude*.
- Right-click on `10mc3_200.vol.VelocityZ` and select *Display > Illuminated Streamlines* and set its parameters as described in Figure 12.19.
- Click on *Apply*.

The resulting visualization and Project View are described by Figure 12.20.
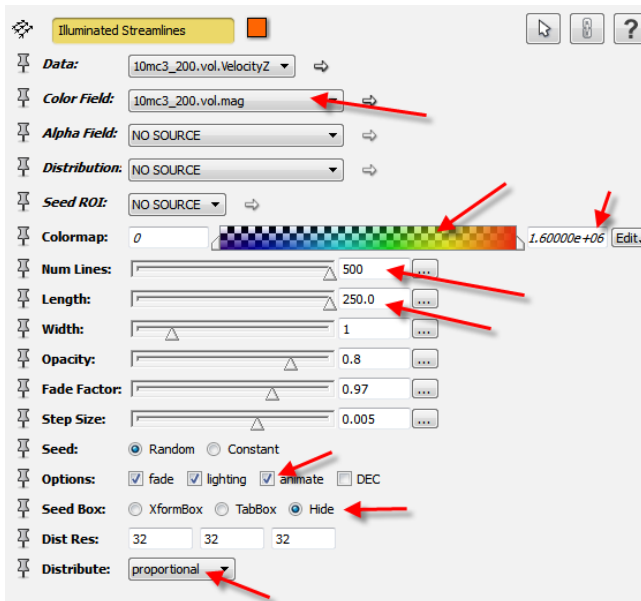
Figure 12.19: Modified parameters for *Illuminated Streamlines* representing the velocity field in the experiment simulation are highlighted by red arrows.
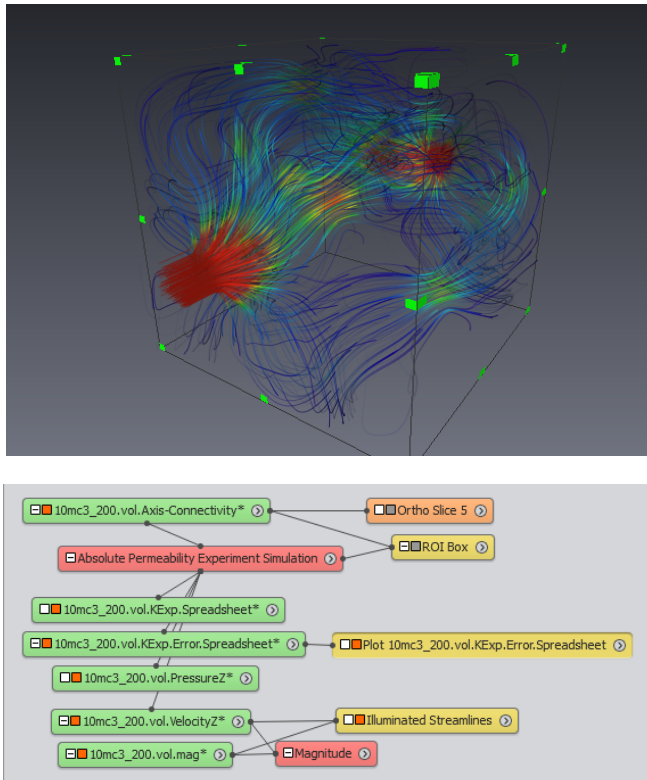
Figure 12.20: Visualization of the *Illuminated Streamlines* representing the velocity field in the experiment simulation. The viewer and Project View should look like in this picture.

To visualize the pressure field:

- Hide the *Illuminated Streamlines* module by clicking on its viewer toggle.
- Right-click on `10mc3_200.vol.PressureZ` and select *Display > Height Map Slice*.
- Configure the *Height Map Slice* as in Figure 12.21.
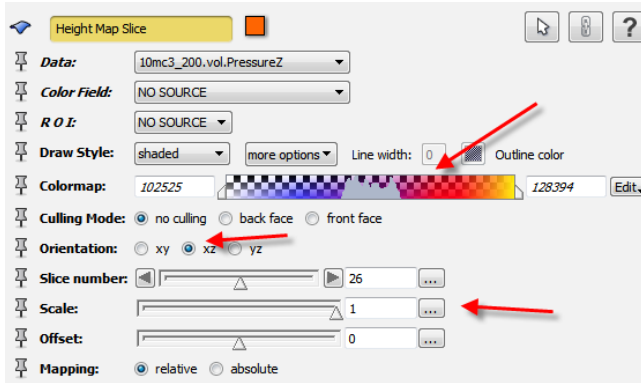
Figure 12.21: Modified parameters for *Height Map Slice* representing the pressure field in the experiment simulation are highlighted by red arrows.
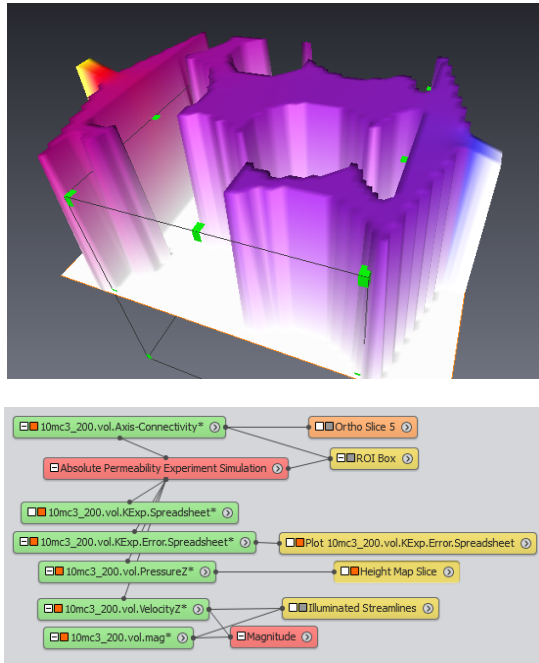
Figure 12.22: Visualization of the *Height Map Slice* representing the pressure field in the experiment simulation. The viewer and Project View should look like in this picture.

Note: The experimental setups are automatically designed after the calculation is initiated. Figure 12.23 describes their shape. On the faces perpendicular to the flow direction, a simple shape is added. It is composed of:

- a channel with a square section to stabilize (from a numerical point of view) the flow in the system, to ease convergence of the iterative algorithm;
- a diverging part to minimize the amount of void space added to the system and to spread the flow on the input face;
- a totally fluid zone to be sure that the fluid can enter through the complete surface of the sample.

On the other faces of the sample, a one-voxel-wide solid plane is added, to be sure that the fluid is contained in the system.
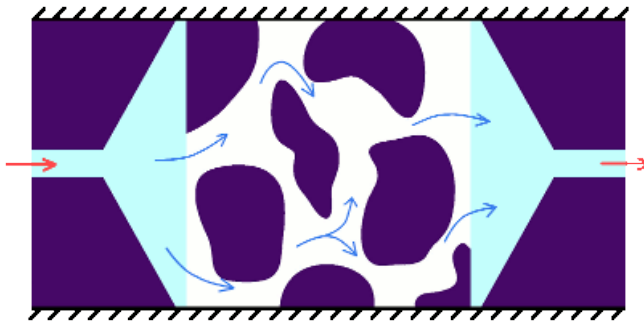
Figure 12.23: 2D example of experimental setups created The experimental setup (left and right of the system) enables the flow to spread through the sample.

12.1.9   Step 8 - Absolute permeability tensor calculation

Purpose	Calculate the intrinsic permeability tensor.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.Labels` if step 6 was not completed) and select *XLab Simulations > Absolute Permeability Tensor Calculation*.
- Connect the *ROI Box* to the ROI input connection of the *Absolute Permeability Tensor Calculation* module.
- If step 5 was completed, check the box with *1* in the *Pore space* port, since the label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 5 was not completed, check the box with *0* in the *Pore space*, since the label of the void space in `10mc3_200.vol.Labels` is 0.

- The module parameters should look like Figure 12.24 if step 5 was completed.
- Click on *Apply*.

With these parameters, the module will compute the full intrinsic permeability tensor. A full tensor computation requires three computations equivalent in time and memory consumption to the experiment simulation.
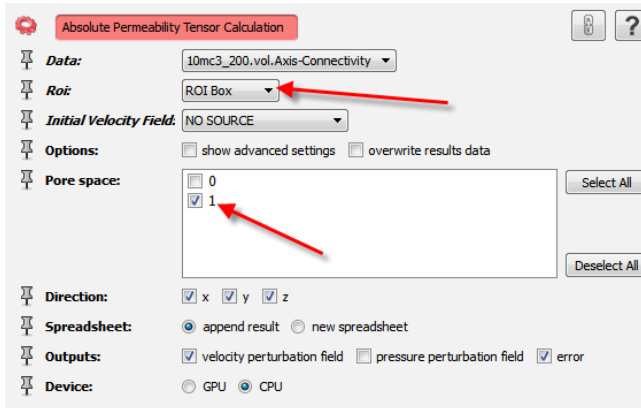
Figure 12.24: Modified parameters of the *XLab Simulations > Absolute Permeability Tensor Calculation* module are highlighted by red arrows. (1) Connect the *ROI Box* module to reduce the computation domain. (2) Select the label 1 as it represents the void space between the spheres.

12.1.9.1 Retrieving and interpreting results

Five outputs appear in the Project View:

- `10mc3_200.vol.KTensor.Spreadsheet` : a spreadsheet containing the most relevant results of the computation:
    - name of the data set describing the geometry of the material;
    - region of interest, on which the computation occured;
    - full permeability tensor in a 3 by 3 matrix form, in μm^2;
    - the eigen system of the tensor. The eigenvalues and their associated vector are described on a line.

- `10mc3_200.vol.KTensor.Error.Spreadsheet` : a spreadsheet containing the estimation of the convergence criterion at each iteration and for each direction. The curve of convergence criterion w.r.t iterations is displayed by `Plot 10mc3_200.vol.KTensor.Error.Spreadsheet`.
- `10mc3_200.vol.VelocityX`, `10mc3_200.vol.VelocityY`,

`10mc3_2.vol.VelocityZ` : three vector fields representing the tensor $\overrightarrow{\overrightarrow{D}}$ solution to the tensorial problem derived from the Stokes equations with the volume averaging approach.

The scalar fields results representing the vector $\overrightarrow{d}$ solution of the same problem are not shown in this tutorial, since their visualization usually does not provide relevant information.

The spreadsheet containing the tensor and all the relevant information can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.25.

	Geometry file	Region of interest	kx [um^2]	ky [um^2]	kz [um^2]	Eigen values	Eigen vector x	Eigen vector y	Eigen vector z
1	10mc3_200.vol....	285 285 285 / 47...	2.098943	0.24279919	-0.096874624	2.06106	-0.9896614	0.13364789	-0.052043647
2			0.24285112	3.9993839	0.35955411	3.9030085	0.14336637	0.92743146	-0.34542292
3			-0.096958332	0.35966703	4.8284421	4.9627004	-0.0020899624	0.34921917	0.93703872

Figure 12.25: Spreadsheet containing the main results of the *Absolute Permeability Tensor Calculation* computation.

A spreadsheet can be exported into several formats (CSV, XML, txt for example).

12.1.9.2   Visualizing output fields

To visualize the velocity fields:

- Hide *Height Map Slice* by clicking on its viewer toggle.
- Right-click on `10mc3_200.vol.VelocityX` and select *Compute > Magnitude*.
- Right-click on `10mc3_200.vol.VelocityX` and select *Display > Illuminated Streamlines* and set its parameters as described in Figure 12.26.
- Click on *Apply*.

The resulting visualization and Project View are described by Figure 12.27.
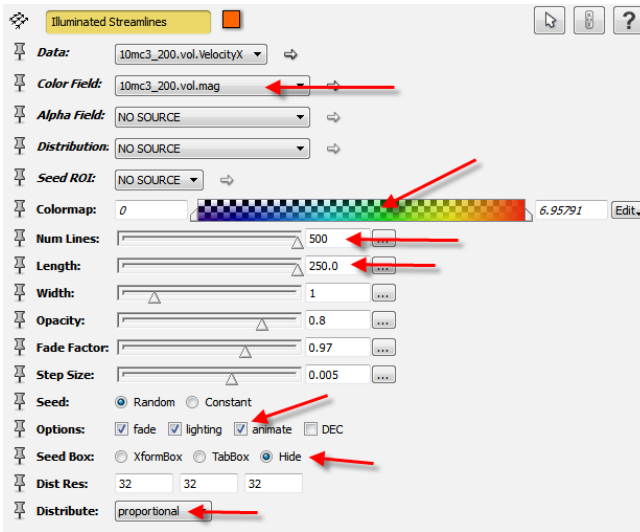
Figure 12.26: Modified parameters for *DisplayISL* module for velocity field result of absolute permeability tensor calculation.
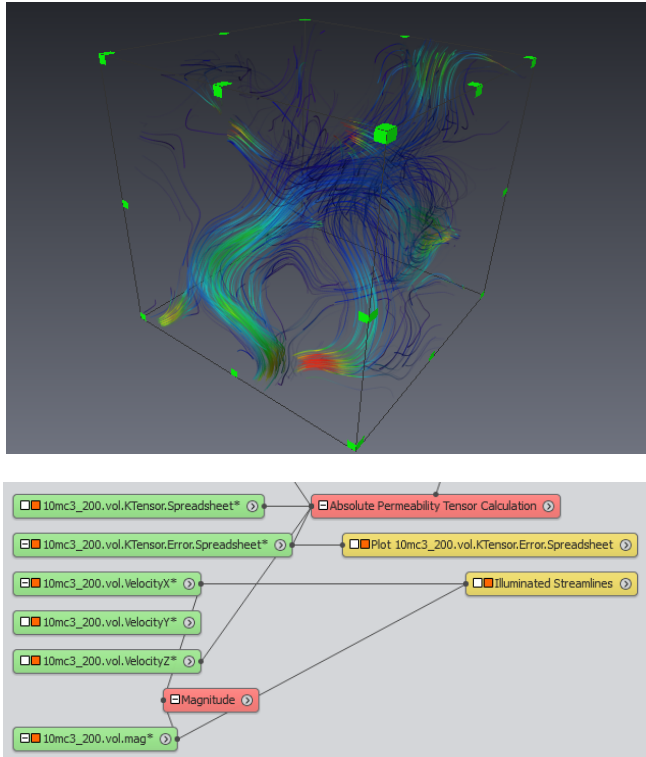
Figure 12.27: Visualization of the *Illuminated Streamlines* representing the velocity field in the calculation of one line of the intrinsic permeability tensor. The viewer and Project View should look like in this picture.

The other velocity fields can be visualized by replaying the same procedure for each.

12.1.10   Step 9 - Permeability validation with Kozeny-Carman equation

Purpose	Use Kozeny-Carman equation to check the accuracy of the result.
Mandatory step	No.

The data set used in this tutorial is a random packing of spheres. For such material, the Kozeny-Carman equation can be used to estimate permeability from porosity and sphere diameters. Kozeny-Carman equation can be written as:

$$k_{KC} = \frac{1}{180} \frac{\epsilon^3}{(1-\epsilon)^2} d^2$$

where:

- ϵ is the porosity of the sphere packing;

- d is the diameter of the spheres.

We work here with the complete volume, high-resolution version of the data set. It can be found in the same directory as the resampled data set used in this tutorial: `data/tutorials/xlab/10mc3_400.vol.am`.

Mean diameter of the spheres could be determined using Avizo toolset, following a procedure similar to tutorial *Example 4: Further Image Analysis - Distribution of Pore Diameters in Foam.* Using the experimental parameters, the spheres diameters are between 100 and $120\mu m$.

Porosity can also be estimated with Avizo, and the value is 36.35%.

From these two elements, the permeability value calculated from the Kozeny-Carman equation is between $6.59d$ and $8.00d$.

These values can be compared to the permeabilities obtained with absolute permeability simulations modules on the complete geometry (without the region of interest used in this tutorial). Experiment simulation in each direction gives the following values:

$$\begin{aligned} k_X &= 7.80d \\ k_Y &= 7.85d \\ k_Z &= 7.75d \end{aligned}$$

The intrinsic permeability tensor of the material is:

$$\begin{bmatrix} 7.57 & 0.25 & -0.23 \\ 0.25 & 7.54 & -0.18 \\ -0.23 & -0.18 & 7.15 \end{bmatrix}$$

The simulated experimental values and the diagonal values of the tensor are all in the range of permeabilities estimated with Kozeny-Carman equation. We remind the user that these values were computed with the complete volume, high-resolution version of the data set: `data/tutorials/xlab/10mc3_400.vol.am`. A second demo script is also provided, which runs the steps of this tutorial on the full resolution data set, restricted to a region of interest.

12.2 Getting started with molecular diffusitivity computation

The purpose of this step-by-step tutorial is to help you become more familiar with the usage of molecular diffusion computation modules provided with the Avizo XLabSuite Extension for Avizo. The following subjects will be addressed in this chapter:

- *theory basics about molecular diffusion*
- *data preparation for simulation (step 1 to step 5)*
- *experiment simulation (step 6)*: define parameters, run simulation, interpret and visualize results
- *effective property calculation (step 7)*: define parameters, run simulation, interpret and visualize results

- *validate results with several empirical laws (step 8).*

As the subject of data preparation for simulation has been addressed in the previous tutorial for *absolute permeability computation*, it will only be shortly evoked here:

- setting voxel size and units,
- segmenting void space,
- removing non-percolating space with Avizo,
- defining a region of interest.

A demo script is also provided (for Microsoft Windows only). This script will automatically perform all the steps detailed in the tutorial.

12.2.1 Theoretical elements

Fick's first law: definition of molecular diffusion

Molecular diffusion is a process whereby dissolved mass is passively transported from a higher chemical energy state to a lower chemical energy state through random molecular motion. Steady state diffusion of a chemical species in free solution can be described empirically using Fick's first law:

$$\vec{j} = -D.\vec{\nabla}c$$

where:

- $\vec{j}$ is the solute mass flux (in $mol.m^{-2}.s^{-1}$),
- D is the diffusion coefficient of the solute in the solvent (in $m^2.s^{-1}$),
- c is the concentration of the solute in the solvent (in $mol.m^{-3}$).

Fick's second law

The partial differential equation describing transient diffusion in a homogeneous (only one solid phase), saturated (the void space of the material is filled by the solvent), porous medium can be developed from the Fick's first law and conservation of mass. This equation is called Fick's second law and is written as follows:

$$\frac{\partial c}{\partial t} - D.\nabla^2 c = 0$$

In the *Molecular Diffusivity Experiment Simulation* module, a classical experiment is suggested, which is based on the double reservoir test. Two reservoirs having the same volume V_R are positioned on each side of the sample in a chosen direction. The other directions are closed with hermetic planes, so that no diffusion occurs. The initial concentrations of the reservoirs are different: $C_{in}(t)$ and $C_{out}(t)$. The sample is initially filled with the solution at $C_{in}(t_0)$, t_0 being the instant when the

experiment starts. At time $t = t_0$, the reservoirs are connected to the sample and the diffusion process starts. The influence of gravity is neglected, only passive diffusion is considered, not advection.

Considering these boundary conditions, Fick's second law governs the diffusion and defines the concentration field in the sample. The concentration of the reservoirs also evolves since they have a finite volume V_R. By default, V_R is supposed to be 100 times higher than the void space volume in the sample. Let's note $\beta = \frac{V_{voidspace}}{V_R}$ the ratio of the void space volume and the reservoir volume.

The following equations govern the concentration in the reservoirs:

$$V_R \frac{\partial C_{in}(t)}{\partial t} = D \int_{S_{in}} \vec{\nabla} c . \vec{n} dS$$

$$V_R \frac{\partial C_{out}(t)}{\partial t} = -D \int_{S_{out}} \vec{\nabla} c . \vec{n} dS$$

where S_{in} and S_{out} the faces of the sample where the reservoirs are connected.

Once the diffusion process starts, the concentration in the sample quickly evolves and the exchanges with the reservoirs are dissymmetrical. This transient state is then replaced by an established state, when the exchanges with the reservoirs are equal.

This established state is characterized by the fact that $\frac{\partial C_{in}(t)}{\partial t} = -\frac{\partial C_{out}(t)}{\partial t}$. Once this state is attained, the concentration in the reservoirs will continue to vary until they reach the equilibrium concentration c_∞. The difference of these concentrations, $C_{out}(t) - C_{in}(t)$, follows an exponential law:

$$C_{out}(t) - C_{in}(t) = p.exp\left(-\lambda^2 t\right)$$

where p and λ^2 are constant coefficients to be determined.

An analytic solution to this problem is suggested:

$$c(X,t) = A \left[\left[\frac{cos\left(\frac{\lambda}{\sqrt{D_{app}}} - 1\right)}{sin\left(\frac{\lambda}{\sqrt{D_{app}}}\right)} \right] cos\left(\frac{\lambda}{\sqrt{D_{app}}} X\right) + sin\left(\frac{\lambda}{\sqrt{D_{app}}} X\right) \right] exp\left(-\lambda^2 t\right) + c_\infty$$

where $c(X,t)$ is the local concentration at position X and time t, A is a constant coefficient.

Knowing this solution must verify the previous hypothesis of flux equality ($\frac{\partial C_{in}(t)}{\partial t} = -\frac{\partial C_{out}(t)}{\partial t}$), the following equation is derived:

$$-\lambda^2 \frac{cos\left(\frac{\lambda}{\sqrt{D_{app}}} - 1\right)}{sin\left(\frac{\lambda}{\sqrt{D_{app}}}\right)} = D_{app}\beta \frac{\lambda}{\sqrt{D_{app}}}$$

which links the λ^2 coefficient to the apparent diffusivity D_{app} of the sample.

To sum up:

Getting started with molecular diffusitivity computation

1. A first transient state during which the diffusion process starts must be achieved before an established state appears.
2. Once the established state has begun, the difference of the reservoirs concentration follows an exponential law. Therefore, the slope of the linear curve followed by $ln\left(C_{out}\left(t\right) - C_{in}\left(t\right)\right)$ can be estimated easily.
3. This slope is λ^2, the exponential coefficient, which is related to the apparent diffusivity D_{app}.

Volume averaged form of Fick's law

The *effective* molecular diffusivity tensor gives global information about the diffusion capabilities of the material and is computed by the *Molecular Diffusivity Tensor Calculation* module.

A change of scale is necessary to get equations valid on the entire volume. The method of volume averaging is a technique that accomplishes a change of scale. Its main goal is to spatially smooth equations by averaging them on a volume. The interested reader will find relevant details and references in [Whitaker, S., *The Method of Volume Averaging*, Kluver Academic Publishers, 1999].

This theory performs to develop a closure problem that transforms the Fick's equations to a vectorial problem; closure variable $\vec{b}$ is used to state the concentration perturbation in a new problem:

$$\nabla^2 \vec{b} = 0$$

When the problem is solved, it is possible to compute the dimensionless diffusivity tensor defined as:

$$\epsilon \frac{\overrightarrow{\overrightarrow{D}}}{D_{solution}} = \epsilon \left(\overrightarrow{\overrightarrow{I}} + \frac{1}{V_f} \int_{S_{fs}} \overrightarrow{n_{fs}} \overrightarrow{b} \, ds \right)$$

where:

- ϵ is the porosity,
- $D_{solution}$ is the bulk solution diffusivity,
- V_f is the volume of fluid,
- S_{fs} is the area of the fluid-solid interface,
- $\overrightarrow{n_{fs}}$ is the normal to the fluid-solid interface directed from the fluid to the solid phase.

Boundary conditions

Avizo XLabSuite Extension provides two approaches to estimate molecular diffusivity.

The first is an experiment simulation based on Fick's equations resolution. As said previously, this is done in the *Molecular Diffusivity Experiment Simulation* module. The rate of reaction of the solid is assumed to be zero: there is no reaction occuring at the fluid-solid interface. Then the boundary

condition at fluid-solid interface is:

$$-\overrightarrow{n_{fs}}.\overrightarrow{\nabla}c = 0$$

where $\overrightarrow{n_{fs}}$ is the normal to the fluid-solid interface directed from the fluid to the solid phase.

Besides this fluid-solid interface condition, a one-voxel-wide plane of solid is added on the faces of the image that are not perpendicular to the main diffusion direction. This allows isolation of the sample from the outside.

Boundary conditions at inlet and outlet require knowledge of the concentrations in the reservoirs. These concentrations evolve over time:

$$V_r\frac{\partial C_{in}}{\partial t} = \int_{S_{in}} \overrightarrow{n_{S_{in}}}.\overrightarrow{\nabla}c\,ds$$

$$V_r\frac{\partial C_{out}}{\partial t} = -\int_{S_{out}} \overrightarrow{n_{S_{out}}}.\overrightarrow{\nabla}c\,ds$$

when S_{in} and S_{out} are respectively the input and output face of the sample; $\overrightarrow{n_{S_{in}}}$ and $\overrightarrow{n_{S_{out}}}$ are respectively the normal to the input and output face.

The second approach solves the closure problem derived from the Fick's equation by volume averaging. This is done in the *Molecular Diffusivity Tensor Calculation* module. The vectorial problem that is solved in this case is closed by imposing periodic boundary conditions to $\vec{b}$ and the geometry. The fluid-solid interface condition has the following similar form:

$$-\overrightarrow{n_{fs}}.\overrightarrow{\nabla}\,\vec{b} = \overrightarrow{n_{fs}}$$

Discretization of the equation system

Avizo XLabSuite Extension uses a finite volume method to solve the equation systems.

The discretization scheme assumes that the voxel is isotropic (cubic).

System resolution for the experiment simulation

The boundary conditions at the inlet and outlet of the experiment simulation require knowledge of the concentrations in the reservoirs. These concentrations evolve over time, which makes it mandatory to have an explicit resolution of the problem and not a direct resolution as considered for effective diffusivity or both apparent and effective electrical conductivity.

System resolution for periodic boundary conditions

Once discretized, the closure equation system can be written as $Ax = b$, A being a sparse, symmetric matrix.

The equation system is solved using a fully implicit method (matrix inversion). PETSc (Portable, Extensible Toolkit for Scientific Computation) library is used for the direct resolution of the linear system.

An iterative resolution with a conjugate gradient and ILU preconditioner is performed. The convergence criterion used is the relative decrease of the residual l_2-norm.

12.2.2 Step 1 - Load the data set

Purpose	Load the data set in Avizo.
Mandatory step	Yes.

The data set that is used in this tutorial is a random glass sphere packing. It was scanned by Dominique Bernard, Research Director at ICMCB-CNRS. The spherical particles were sieved to have diameters in the 100-120 microns range. The sample was sintered for 10 minutes at 700° Celsius prior to scanning. Figure 12.28 shows the data set.

The complete 3D data set used in this tutorial is a 200×200×200 cube.

- Start a new Project *File > New Project*
- Open the *File > Open Data...* dialog.
- Open the `data/tutorials/xlab/10mc3_200.vol.am` file from `AVIZO_ROOT`directory.
- Click on *Open*.
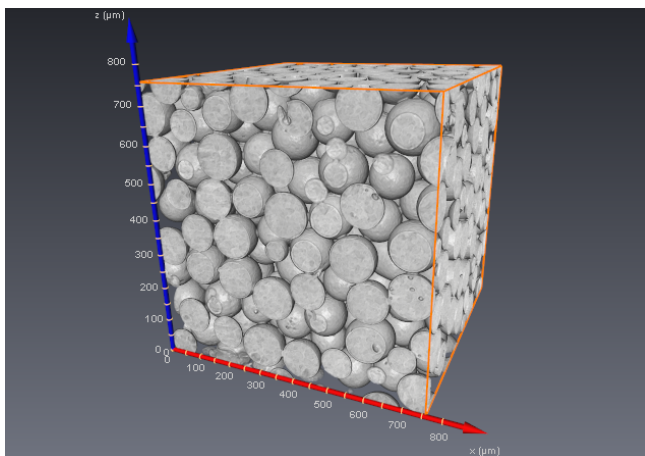
Figure 12.28: 3D visualization of the sphere pack used in this tutorial.

As mentioned in the previous step, unit management is not mandatory here. However, refer to *step 2* of absolute permeability simulations tutorial to learn more about editing the length units.

12.2.3 Step 2 - Set the voxel size

Purpose	Set the edge size of the voxels in the uniform data set.
Mandatory step	No.

Again, as mentioned in the previous steps, correctly setting the dimension of the voxels is not mandatory here. However, to learn more about using the *Crop Editor* to set the voxel size, please refer to *step 3* of absolute permeability simulations tutorial.

Note: Isotropic, i.e., cubic, voxels are mandatory for Avizo XLabSuite Extension computations.

12.2.4 Step 3 - Create a label image from the data set

Purpose	Use basic segmentation tools to create a label image from the data set.
Mandatory step	Yes.

This step consists of creating a label image from the initial data set that is smoothed and thresholded, using some of the segmentation tools provided by Avizo.

Please follow the instructions described in *step 4* of absolute permeability simulations tutorial.

Figure 12.29: Visualization of the thresholded data set.

12.2.5 Step 4 - Remove non-percolating void space

Purpose	Use advanced segmentation tools to remove non-percolating void space in the thresholded image.
Mandatory step	No. Only available with Avizo.

The percolation test can be performed before running the computation. An advantage of this test is that is removes all isolated void spaces from the label image (we remind the user that the diffusion co-efficient as well as the rate of reaction of the solid phase are assumed to be zero). Computation should then be a little faster. The necessary connectivity, from a discrete point of view, in 6-connectivity: a concentration can be estimated over a non-nil surface only, i.e., a face between two neighbor voxels.

If desired, you can follow the instructions described in *step 5* of absolute permeability simulations tutorial to perform the percolation test.
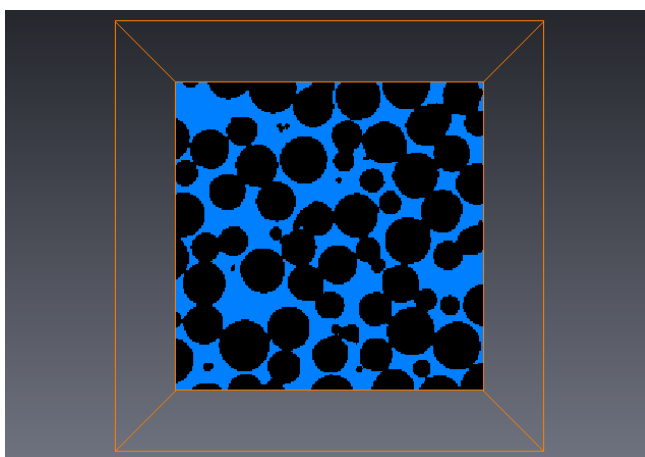
Figure 12.30: Visualization of the percolating porosity in data set.

12.2.6 Step 5 - Selection of a sub-region

Purpose	Define a region of interest for the molecular diffusivity computation.
Mandatory step	Yes.

There is an easy way to compute molecular diffusivity on a sub-region of the loaded data set without cropping it. A region of interest (ROI) can be defined and connected to Avizo XLabSuite Extension modules, so that the computation is only performed in the ROI.

Please follow the instructions given in *step 6* of absolute permeability simulations tutorial and define a ROI cube with 50 voxels on a side centered in the sample (see Figure 12.31). The ROI will be used in this tutorial to compute properties on small volumes. It can be moved through the data set at will

during the tutorial, although its initial position will be used for illustration purposes.

For convenience and because it will save time during this tutorial phase, we compute the molecular diffusivity on a sub-region of the sample. However, we will see in the validation phase how resampling and using ROI affects the quality of the computed results.
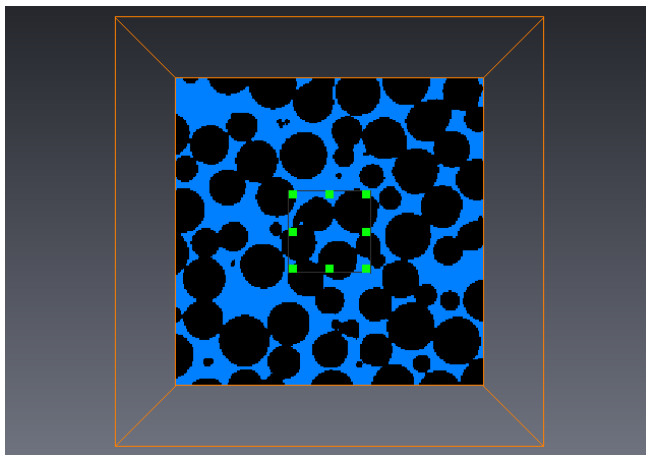
Figure 12.31: Visualization of the cube representing the region of interest.

12.2.7 Step 6 - Molecular diffusivity experiment simulation

Purpose	Simulate a molecular diffusivity laboratory measurement.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.Labels` if step 4 was not completed) and select *XLab Simulations > Molecular Diffusivity Experiment Simulation*.
- Connect the *ROI Box* to the ROI input connection of the *Molecular Diffusivity Experiment Simulation* module.
- If step 4 has been completed, check the box referring to label 1 in the *Pore space* port, since the label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 4 was not completed, check the box with referring to label 0 in the *Pore space*, since the label of the void space in `10mc3_200.vol.Labels` is 0.
- Click on *Apply*.

The default parameters simulate an experiment along the Z axis with a concentration in the input reservoir at initial time of 1711 $mol.m^{-3}$ and the concentration in the output reservoir at initial time being

null. The default solution bulk diffusivity is 1 $m^2.s^{-1}$. The options can be modified to simulate several experiments. For example, the direction of the molecular diffusion can be adjusted to X, Y, or Z direction (default is Z). If several directions are selected, the computations will be done successively. The concentration values used as boundary conditions of the experiment can also be modified. Modifying these values will not change the molecular diffusivity, which is intrinsic to the porous medium. It will only modify the output concentration field.

12.2.7.1 Retrieving and interpreting results

Four outputs appeared in the Project View:

- 10mc3_200.DExp.Spreadsheet : a spreadsheet containing the most relevant results of the computation:
  - name of the data set,
  - region of interest, on which the computation is done,
  - apparent molecular diffusivity in $m^2.s^{-1}$,
  - concentration imposed at initial time in the input reservoir in $mol.m^{-3}$,
  - concentration imposed at initial time in the output reservoir in $mol.m^{-3}$,
  - solution bulk diffusivity in $m^2.s^{-1}$.

- 10mc3_200.vol.DExp.ResConcentration.Spreadsheet: a spreadsheet containing the uniform concentrations of the input and output reservoirs at each iteration.
- 10mc3_200.vol.DExp.Error.Spreadsheet: a spreadsheet containing the estimation of the convergence criterion at each iteration. The curve of convergence criterion w.r.t. iterations is displayed by Plot 10mc3_200.vol.DExp.Error.Spreadsheet.
- 10mc3_200.vol.ConcentrationZ: a scalar field representing the concentration field (in $mol.m^{-3}$), solution of the Fick's equation system solved in the geometry restricted by the ROI.

The spreadsheet 10mc3_200.vol.DExp.Spreadsheet containing all the relevant information can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.32. A spreadsheet can be exported into several formats (CSV, XML, txt for example).

Geometry file	Region of interest	Apparent molecular diffusivity [m^2.s^-1]	Input concentration [mol.m^-3]	Output concentration [mol.m^-3]	Bulk solution diffusivity [m^2.s^-1]
1 10mc3_200...	284.525 284.525...	0.15075056	1711	0	1

Figure 12.32: Spreadsheet containing the main results of the *Molecular Diffusivity Experiment Simulation* computation.

12.2.7.2 Visualizing the reservoirs concentration evolution

To visualize the evolution of the concentration of the reservoirs:

- Select `10mc3_200.vol.DExp.ResConcentration.Spreadsheet` and select *Plot Spreadsheet* among the macro buttons.
- In *Plot Spreadsheet* properties area, select both `DExp.Z.Input concentration` and `DExp.Z.Output concentration` in the Y port.
- Press on the *Show* button.

The resulting visualization should look like Figure 12.33. One can notice that the input and output concentrations are far from having reached the end concentration (described as C_∞ in the *theory pages*). As explained in the theory pages, the computation is stopped once the behavior of the logarithm of the difference between the input and end concentrations becomes linear w.r.t. time.
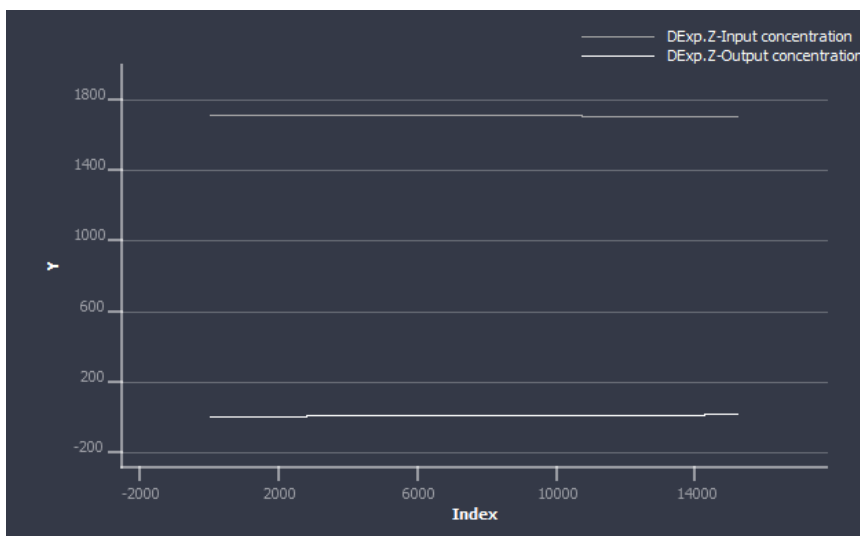
Figure 12.33: Visualization of the evolution of both input and output reservoirs concentration.

12.2.7.3 Visualizing the output concentration field

To visualize the concentration field:

- Hide all current display modules, such as *Bounding Box* or *Ortho Slice*, by clicking on their viewer toggle.
- Right-click on `10mc3_200.vol.ConcentrationZ` and select *Display > Ortho Slice*.
- In the *Ortho Slice* properties, change the selected colormap to *temperature.icol* from the *Edit*

menu.

- Select the *Edit* menu of the colormap again, and then select *Adjust range to > Data min-max*.
- In the *Orientation* port, select the *yz* orientation.
- Zoom on the slice to visualize it.

The resulting visualization should look like Figure 12.34. You can observe the decrease of the concentration from the input reservoir (at the bottom of the slice, in yellow), to the output reservoir (at the top of the slice, in light blue).
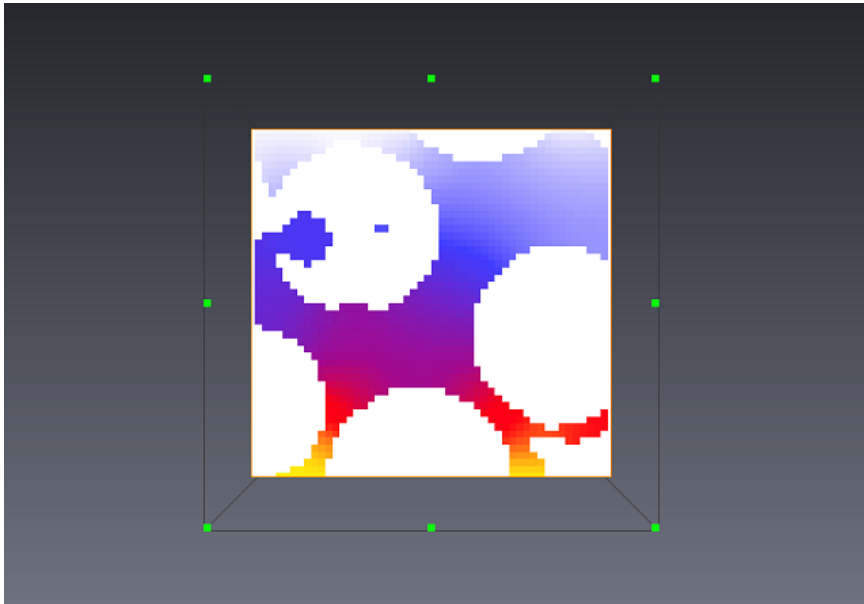
Figure 12.34: Visualization of concentration field in an experiment simulation with molecular diffusion in Z direction.

12.2.8 Step 7 - Molecular diffusivity tensor calculation

Purpose	Calculate the intrinsic molecular diffusivity tensor.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.Labels` if step 4 was not completed) and select *XLab Simulations > Molecular Diffusivity Tensor Calculation*.
- Connect the *ROI Box* to the ROI input connection of the *Molecular Diffusivity Tensor Calculation* module.

- If step 4 was completed, check the box referring to label 1 in the *Pore space* port, since the label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 4 was not completed, check the box referring to label 0 in the *Pore space*, since the label of the void space in `10mc3_200.vol.Labels` is 0.
- Click on *Apply*.

With these parameters, the module will compute the full intrinsic diffusivity tensor. A full tensor computation requires three computations, each equivalent in time and memory consumption to one experiment simulation.

Note that the *concentration field* output is not selected by default. This output corresponds to the $\vec{b}$ vector, solution of the vectorial problem derived from the Fick's equation with the volume averaging approach (see *molecular diffusion simulations theory pages*). It is used in the computation of the effective molecular diffusivity but its visualization is hard to interpret.

12.2.8.1 Retrieving and interpreting results

Only the spreadsheet `10mc3_200.vol.DTensor.Spreadsheet` output is generated in the Project View. The spreadsheet can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.35. It contains information about the computation results gathered in two tables.

- name of the data set,
- region of interest, on which the computation occured,
- full molecular diffusivity tensor in a 3 by 3 matrix form (dimensionless),
- the eigen system solutions for the tensor. The eigenvalues and their associated eigenvectors are described on a line.

A spreadsheet can be exported into several formats (CSV, XML, txt for example).

	Geometry file	Region of interest	D_x	D_y	D_z	Eigen values	Eigen vector x	Eigen vector y	Eigen vector z
1	10mc3_200...	284.525 284.525...	0.11944524	0.0044346885	-0.0048923502	0.11821773	-0.98314011	0.12698269	-0.13157111
2			0.0041252556	0.15591197	0.0055545033	0.15290774	0.18940859	0.79791749	-0.57223433
3			-0.0047252621	0.0050846385	0.15843366	0.1626654	-0.023946581	0.62627238	0.7792365

Full tensor

Figure 12.35: Tables of the spreadsheet containing the main results of the *Molecular Diffusivity Tensor Calculation* computation.

12.2.9 Step 8 - Molecular diffusivity computation validation

Purpose	Use experimental results and empirical laws to check the accuracy of the computed molecular diffusivity.
Mandatory step	No.

We base our validation on several studies reporting the results of molecular diffusivity experimental measurements and empirical laws aimed at computing the molecular diffusivity of a material.

Vrettos *et al.* (1989) reported the measurements of Carman (1956) on different types of samples (glass beadpacks, white sand, glass sand, quartz sand, sand).

Kim *et al.* (1987) studied isotropic systems of packed beds of glass spheres and reported the measurements of Currie (1960) and Hoogschagen (1955) and compared them to his own measurements.

Saez *et al.* (1991) and Whitaker (1999) reported the following analytical estimations of molecular diffusivity with respect to the porosity ϵ:

- Maxwell (1881) : $\epsilon \frac{D}{D_{solution}} = \frac{2\epsilon}{3-\epsilon}$
- Weissberg (1963) : $\epsilon \frac{D}{D_{solution}} = \frac{\epsilon}{1-ln(\epsilon)/2}$
- Torquato (1985) : $\epsilon \frac{D}{D_{solution}} = \frac{\epsilon-0,5\epsilon\xi}{1,5-0,5\epsilon-0,5\epsilon\xi}$ with $\xi = 0.21068(1-\epsilon) - 0.04693(1-\epsilon)^2 + 0.00247(1-\epsilon)^3$.

Finaly, Vrettos (1989) and Quintard (1993) obtained numerical results on Voronoi networks and on cubic configurations.

All these results are displayed on Figure 12.36.

We compare these values to the values computed with Avizo XLabSuite Extension modules on the data set used in this tutorial, which is a random packing of spheres. We work here with the complete geometry of the complete volume, high-resolution version of the data set (that is to say a label image obtained from `data/tutorials/xlab/10mc3_400.vol.am`, without any ROI Box).

Porosity can be estimated with Avizo (with *Measure And Analyze > Global Measures > Intensity Integral in 3D interpretation* connected to `10mc3_400.vol.Axis-Connectivity`), and the value is 36.35%.

Experiment simulation in each direction gave the following results:

- $\epsilon \frac{D_X}{D_{solution}} = 0.249$
- $\epsilon \frac{D_Y}{D_{solution}} = 0.255$
- $\epsilon \frac{D_Z}{D_{solution}} = 0.257$

The effective molecular diffusivity tensor computed is:

$$\epsilon \frac{\overrightarrow{\overrightarrow{D}}}{D_{solution}} = \begin{bmatrix} 0.213 & 0.002 & -0.003 \\ 0.002 & 0.213 & -0.001 \\ -0.003 & -0.001 & 0.209 \end{bmatrix}$$

These values are also displayed in Figure 12.36. One can see that Avizo XLabSuite Extension results are in good agreement with the different studies results.
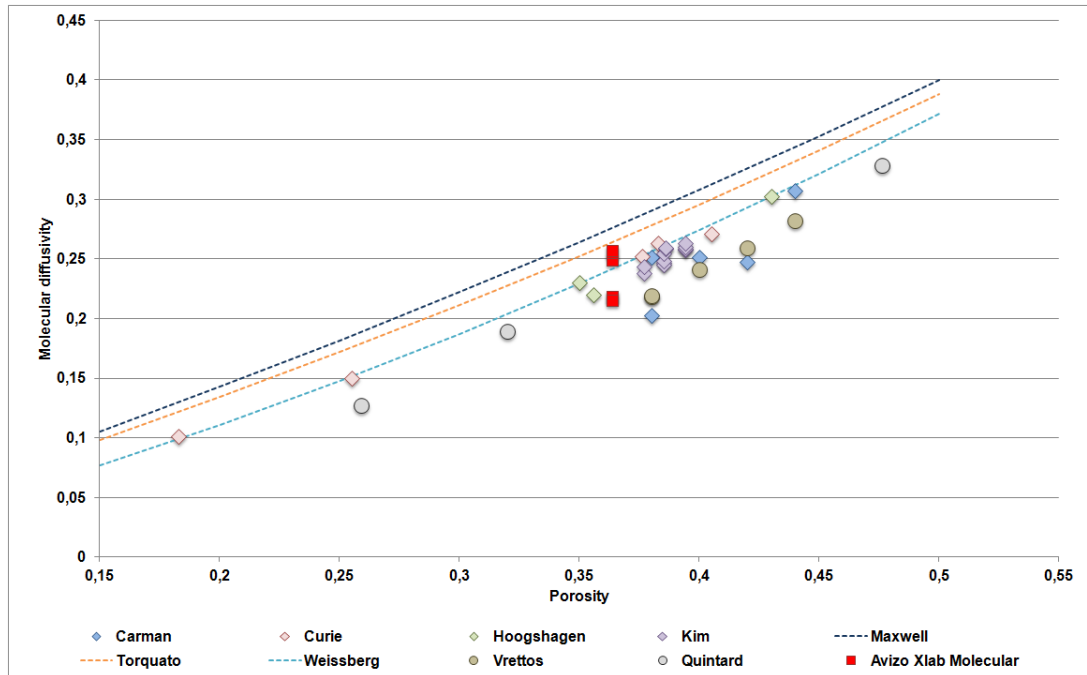
Figure 12.36: Comparison of experimental measurements, empirical laws and numerical simulations (among which Avizo XLabSuite Extension modules) for the determination of molecular diffusivity with respect to the porosity.

A second demo script is also provided (for Microsoft Windows only), which runs the steps of this tutorial on the full resolution data set, restricted to a region of interest.

Bibliography:

1. Carman P.C., *Flow of Gases through Porous Media*, Butterworths, London, 1956
2. Currie J.A., *Gaseous diffusion in porous media. Part I - a non-steady state method*, Brit. J. Appl. Phys., II, 314-324, 1960

Getting started with molecular diffusitivity computation **703**

3. Hoogschagen J., *Diffusion in porous catalysts and absorbents*, Ind. Eng. Chem., 47, 906-913, 1955

4. Kim J.-H., Ochoa J.A., Whitaker S., *Diffusion in Anisotropic Porous Media*, Transport in Porous Media, 2, 327-356, 1987

5. Maxwell J.C., *Treatise on Electricity and Magnetism*, Vol. I, 2nd edn., Clarendon Press, Oxford, 1881

6. Quintard M., *Diffusion in Isotropic and Anisotropic Porous Systems: Three-Dimensional Calculations*, Transport in Porous Media, 11, 187-199, 1993

7. Saez A.E., Perfetti J.C., Rusinek I., *Prediction of Effective Diffusivities in Porous Media using Spatially Periodic Models*, Transport in Porous Media, 6, 143-157, 1991

8. Torquato S., *Effective electrical conductivity of two-phase disordered composite media*, J. Appl. Phys., 58, 3790-3797, 1985

9. Vrettos N.A., Imakoma H., Okazaki M., *Transport Properties of Porous Media from the Microgeometry of a Three-dimensional Voronoi Network*, Chem. Eng. Process, 26, 237-246, 1989

10. Weissberg H.L., *Effective diffusion coefficients in porous media*, J. Appl. Phys., 34, 2636-2639, 1963

11. Whitaker S., *The method of volume averaging*, Theory and applications of transport in porous media Vol. 13, Kluwer Acad. Pub., Dordrecht, 1999

12.3 Getting started with formation factor and electrical conductivity computation

The purpose of this step-by-step tutorial is to become more familiar with the usage of electrical conductivity and formation factor computation modules provided with the Avizo XLabSuite Extension for Avizo. The following subjects will be addressed in this chapter:

- *theory basics about electrical conductivity*
- *data preparation for simulation (step 1 to step 5)*
- *experiment simulation (step 6)*: define parameters, run simulation, interpret and visualize results
- *effective property calculation (step 7)*: define parameters, run simulation, interpret and visualize results
- *validate results with several empirical laws (step 8).*

As the subject of data preparation for simulation has been addressed in the previous tutorial for *absolute permeability computation*, it will only be shortly evoked here:

- setting voxel size and units;
- segmenting void space;
- removing non-percolating space with Avizo;

- defining a region of interest.

A demo script is also provided (for Microsoft Windows only). This script will automatically perform all the steps detailed in the tutorial.

12.3.1 Theoretical elements

Ohm's law: definition of electrical conduction
Electrical conduction is a process whereby electrical charges are transported under the effect of an electrical field. In an electrolyte, the electrical energy is transferred by the free ions. Electrical conduction in a homogeneous material is described by Ohm's law:

$$\vec{j} = -\sigma \vec{\nabla} v$$

where:

- $\vec{j}$ is the current density (in $A.m^{-2}$),
- σ is the electrical conductivity of the material (in $S.m^{-1}$ or $A.V^{-1}.m^{-1}$),
- v is the electrical potential (in V).

Ohm's law in a porous material
In a porous material, the electrical conductivity is modified by the presence of the material around an electrolyte. Generally, in the most common applications of formation factor, the solid phase is considered as an insulator, since its conductivity is orders of magnitude lower than the electrolyte conductivity. We consider a porous media

- homogeneous: there is only one solid phase,
- saturated: the void space of the material is filled by the solvent.

As we consider conductive solution, there is no charges accumulation. Ohm's law and conservation of charge lead to the following equation:

$$\vec{\nabla}.\vec{j} = 0$$

and as we consider only one homogeneous fluid phase, the electrical conductivity σ does not vary in space, which leads to:

$$\nabla^2 v = 0$$

The experiment that is simulated in the *Formation Factor Experiment Simulation* module consists of applying a constant electrical potential difference between two opposite faces of the material sample (direct current is used). The other faces of the sample are enclosed with an electrical insulator. When the final state is attained, input and output current fluxes are equal, and by using the Ohm's law on the

entire volume, the apparent electrical conductivity is estimated by:

$$\frac{j_{total}}{S} = \sigma \frac{V_{in} - V_{out}}{L}$$

where

- j_{total} is the total electrical flux going through the input face,
- S is the area of the input face,
- σ is the electrical conductivity of the material,
- V_{in} and V_{out} are the input and output imposed potentials,
- L is the length of the material sample.

The total electrical flux going through the input face j_{total} can be computed by locally applying Ohm's law :

$$j_{total} = \int_S -\sigma_{solution} \vec{\nabla} v . \vec{ds}$$

where $\sigma_{solution}$ is the electrical conductivity of the free solution.

Important remark: in the experiment, we consider direct current (there is no transient phenomenon) and we neglect the skin effect and the concentration variations. An experimenter will usually have to use correction models to remove those experimental phenomena, which we avoid here with this experiment simulation in *ideal* conditions.

The formation factor is directly linked with the electrical conductivity through its inverse: the electrical resistivity. Indeed, it is "the ratio of the resistivity of a rock filled with water to the resistivity of that water" (*Schlumberger Oilfield Glossary: formation factor, 2009*).

Volume averaged form of Ohm's law

The *effective* electrical conductivity tensor gives global information about the electrical conduction capabilities of the material. A change of scale is necessary to get equations valid on the entire volume. The method of volume averaging is a technique that accomplishes a change of scale. Its main goal is to spatially smooth equations by averaging them on a volume. The interested reader will find relevant details and references in [Whitaker, S., *The Method of Volume Averaging*, Kluver Academic Publishers, 1999].

This very general theory leads to develop a closure problem which transforms the Ohm's equations into a vectorial problem. Closure variable $\vec{b}$ is used to express the electrical potential perturbation in a new problem:

$$\nabla^2 \vec{b} = 0$$

When the problem is solved, it is possible to compute the dimensionless electrical conductivity tensor defined as:

$$\epsilon \frac{\overrightarrow{\overrightarrow{\sigma}}}{\sigma_{solution}} = \epsilon \left(\overrightarrow{\overrightarrow{I}} + \frac{1}{V_f} \int_{S_{fs}} \overrightarrow{n_{fs}} \overrightarrow{b} \, ds \right)$$

where:

- ϵ is the porosity,
- $\sigma_{solution}$ is the solution electrical conductivity,
- V_f is the volume of fluid
- S_{fs} is the area of the fluid-solid interface,
- $\overrightarrow{n_{fs}}$ is the normal to the fluid-solid interface directed from the fluid to the solid phase.

The inverse of the conductivity tensor is the formation factor tensor and the formation factor scalar is the average value of the eigenvalues of this last tensor. All this is computed by the *Effective Formation Factor Calculation* module.

Boundary conditions

Avizo XLabSuite Extension proposes two approaches to estimate electrical conductivity.

The first is an experiment simulation based on Ohm's equations resolution. As said previously, this is done in *Formation Factor Experiment Simulation* module. The material is generally composed of rocks, which can be considered as insulator. The boundary condition at fluid-solid interface is:

$$-\overrightarrow{n_{fs}}.\overrightarrow{\nabla} v = 0$$

where $\overrightarrow{n_{fs}}$ is the normal to the fluid-solid interface directed from the fluid to the solid phase. Besides this fluid-solid interface condition, boundary conditions are:

- one-voxel-wide plane of electrical insulator added on the faces of the image that are not perpendicular to the electrical flux main direction. This allows isolation of the sample from the outside.
- the input and output (the faces that are perpendicular to the flux main direction) designed as one-voxel-wide plane where the potential is imposed.

The second approach solves the closure problem derived from Ohm's equation by volume averaging. This is done in *Effective Formation Factor Calculation* module. The vectorial problem that is solved in this case is closed by imposing periodic boundary conditions to $\vec{b}$ and the geometry. The fluid-solid interface condition has the following similar form:

$$-\overrightarrow{n_{fs}}.\overrightarrow{\nabla} \overrightarrow{b} = \overrightarrow{n_{fs}}$$

Discretization of the equation system

Avizo XLabSuite Extension uses a finite volume method to solve the equation systems.

The discretization scheme supposes that the voxel is isotropic (cubic).

System resolution

Once discretized, the equation systems can be written as $Ax = b$, A being a sparse, symmetric matrix.

The equation systems are solved using a fully implicit method (matrix inversion). PETSc (Portable, Extensible Toolkit for Scientific Computation) library is used for the direct resolution of the linear systems.

An iterative resolution with conjugate gradient and ILU preconditioner is performed. The convergence criterion used is the relative decrease of the residual l_2-norm.

12.3.2 Step 1 - Load the data set

Purpose	Load the data set in Avizo.
Mandatory step	Yes.

The data set that is used in this tutorial is a random glass spheres packing. It was scanned by Dominique Bernard, Research Director at ICMCB-CNRS. The spherical particles were sieved to have diameters in the 100-120 microns range. The sample was sintered for 10 minutes at 700° Celsius prior to scanning. Figure 12.37 shows the data set.

The complete 3D data set used in this tutorial is a $200 \times 200 \times 200$ cube.

- Start a new Project *File > New Project*
- Open the *File > Open Data...* dialog.
- Open the `data/tutorials/xlab/10mc3_200.vol.am` file from `AVIZO_ROOT` directory.
- Click on *Open*.
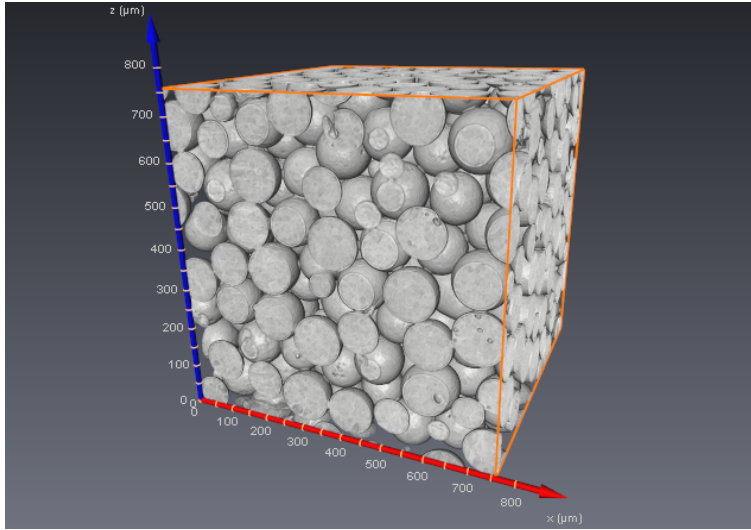
Figure 12.37: 3D visualization of the sphere pack used in this tutorial.

As mentioned in the previous step, unit management is not mandatory here. However, refer to *step 2* of absolute permeability simulations tutorial to learn more about editing the units of length.

12.3.3   Step 2 - Set the voxel size

Purpose	Set the edge size of the voxels in the uniform data set.
Mandatory step	No.

Again, as mentioned in the previous steps, correctly setting the dimensions of the voxels is not mandatory here. However, to learn more about using the *Crop Editor* to set the voxel size, please refer to *step 3* of absolute permeability simulations tutorial.

Note: Isotropic, i.e., cubic, voxels are mandatory for Avizo XLabSuite Extension computations.

12.3.4   Step 3 - Create a label image from the data set

Purpose	Use basic segmentation tools to create a label image from the data set.
Mandatory step	Yes.

This step consists of creating a label image from the initial data set that is smoothed and thresholded, using some of the segmentation tools provided by Avizo.

Please follow the instructions described in *step 4* of absolute permeability simulations tutorial.

Figure 12.38: Visualization of the thresholded data set.

12.3.5 Step 4 - Remove non-percolating void space

Purpose	Use advanced segmentation tools to remove non-percolating void space in the thresholded image.
Mandatory step	No. Only available with Avizo.

The percolation test can be performed before running the computation. An advantage of this test is to remove all isolated void spaces from the label image (we remind the user that the solid material is considered as an electrical insulator and so the current will not reach the solution in isolated spaces). Computation should then be a little faster. The necessary connectivity, from a discrete point of view, in 6-connectivity: a current density can be estimated over a non-nil surface only, i.e., a face between two neighbor voxels.

You might follow the instructions described in *step 5* of absolute permeability simulations tutorial to perform the percolation test.
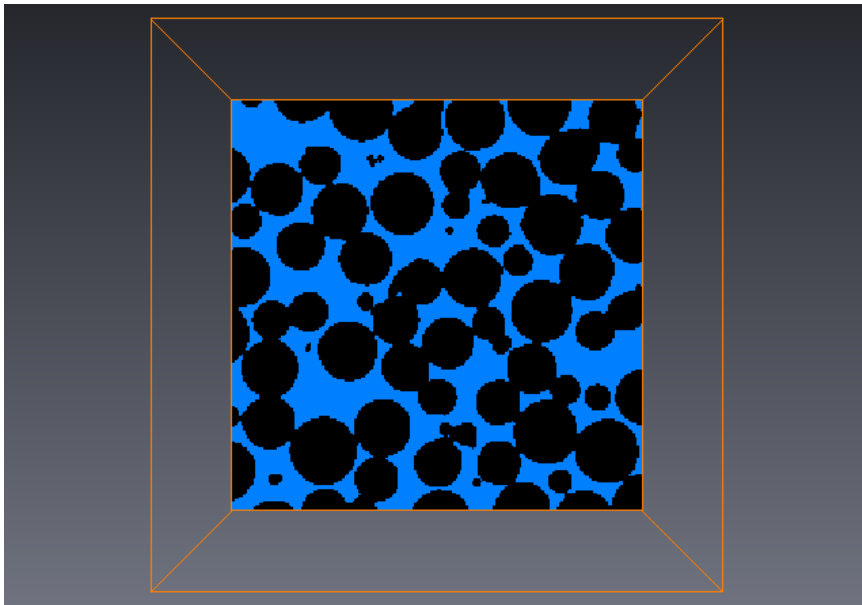
Figure 12.39: Visualization of the percolating porosity in data set.

12.3.6 Step 5 - Selection of a sub-region

Purpose	Define a region of interest for the electrical conductivity computation.
Mandatory step	Yes.

There is an easy way to compute electrical conductivity on a sub-region of the loaded data set without cropping it. A region of interest (ROI) can be defined and connected to Avizo XLabSuite Extension modules, so that the computation is only performed in the ROI.

Please follow the instructions given in *step 6* of absolute permeability simulations tutorial and define a ROI cube with 50-voxel edges centered in the sample (see Figure 12.40). The ROI will be used in this tutorial to compute properties on small volumes. It can be moved through the data set at will during the tutorial, although its initial position will be used for illustrating purposes.

For convenience and because it will save time during this tutorial phase, we compute the electrical conductivity and formation factor on a sub-region of the sample. However, we will see in the validation phase how resampling and using ROI affects the quality of the computed results.
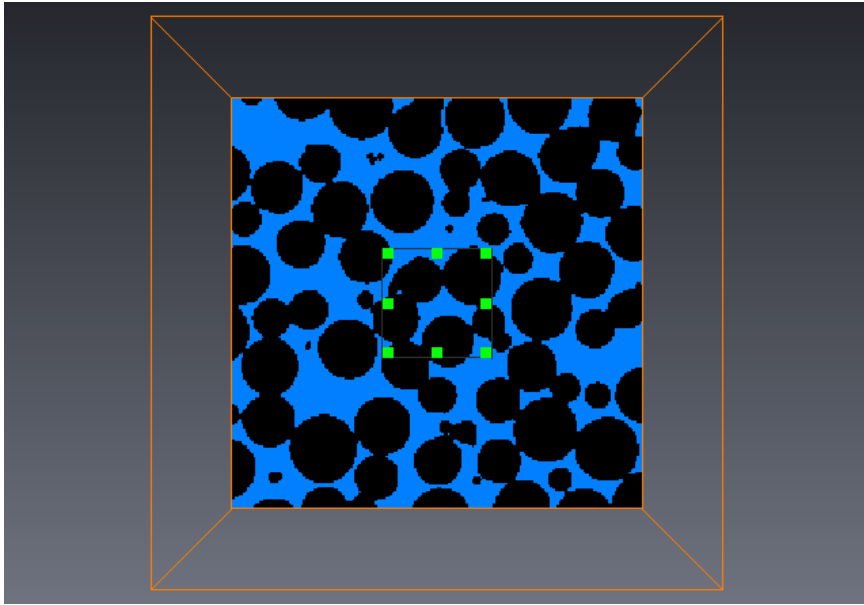
Figure 12.40: Visualization of the cube representing the region of interest.

12.3.7 Step 6 - Formation factor experiment simulation

Purpose	Simulate a formation factor laboratory measurement.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.Labels` if step 4 was not completed) and select *XLab Simulations > Formation Factor Experiment Simulation*.
- Connect the *ROI Box* to the ROI input connection of the *Formation Factor Experiment Simulation* module.
- If step 4 has been completed, check the box refering to label 1 in the *Pore space* port, since the label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 4 was not completed, check the box with refering to label 0 in the *Pore space*, since the label of the void space in `10mc3_200.vol.Labels` is 0.
- Click on *Apply*.

The default parameters simulate an experiment along the Z axis with an input potential of 1 V and the output potential being null. The default solution electrical conductivity is 0.0001 $S.m^{-1}$. The options can be modified to simulate several experiments. For example, the direction of the electrical flux can

be adjusted to X, Y or Z direction (default is Z). If several directions are selected, the computations will be done successively. The potential values used as boundary conditions of the experiment can also be modified. Modifying these values will not change the electrical conductivity or the formation factor, which are intrinsic to the porous media. It will only modify the output potential field.

12.3.7.1 Retrieving and interpreting results

Two outputs appeared in the Project View:

- `10mc3_200.vol.FFExp.Spreadsheet` : a spreadsheet containing the most relevant results of the computation:

  - name of the data set;
  - region of interest, on which the computation is done;
  - apparent electrical conductivity in $S.m^{-1}$;
  - apparent formation factor;
  - potential imposed at input of the experimental setup in V;
  - potential imposed at output of the experimental setup in V;
  - solution electrical conductivity in $S.m^{-1}$.

- `10mc3_200.vol.PotentialZ` : a scalar field representing the potential field (in V), solution of the Ohm's equation system solved in the geometry restricted by the ROI.

The spreadsheet containing all the relevant information can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.41. A spreadsheet can be exported into several formats (CSV, XML, txt for example).

	Geometry file	Region of interest	Apparent electrical conductivity [S.m^-1]	Apparent formation factor	Solution electrical conductivity [S.m^-1]
1	10mc3_200.Axis-Connectivity	284.525 284.525 ...	1.6564074e-005	6.0371618	9.9999997e-005

Z

Figure 12.41: Spreadsheet containing the main results of the *Formation Factor Experiment Simulation* computation.

12.3.7.2 Visualizing output potential field

To visualize the potential field:

- Hide all current display modules, such as *Bounding Box* or *Ortho Slice*, by clicking on their viewer toggle.

- Right-click on `10mc3_200.vol.PotentialZ` and select *Display > Ortho Slice*.
- In the *Ortho Slice* properties, change the selected colormap to *temperature.icol* from the *Edit* menu.
- Select the *Edit* menu of the colormap again, and then select *Adjust range to > Data min-max*.
- In the *Orientation* port, select the *yz* orientation.
- Zoom on the slice to visualize it.

The resulting visualization should look like Figure 12.42. You can observe the decrease of the potential from the input of the device (at the bottom of the slice, in yellow), to the output of the device (at the top of the slice, in light blue).
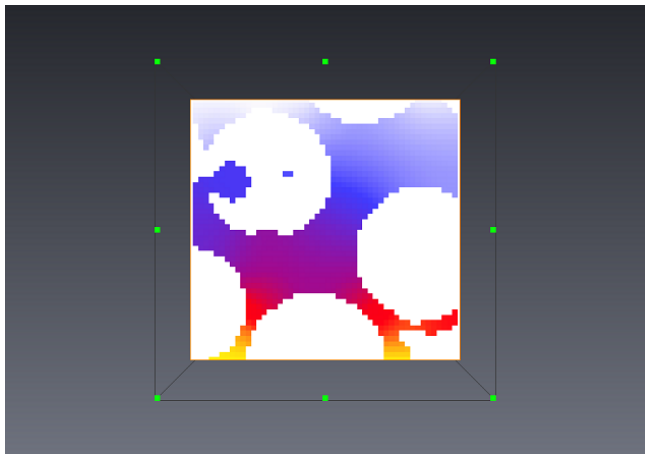
Figure 12.42: Visualization of potential field in an experiment simulation with electrical flux in Z direction.

12.3.8   Step 7 - Effective formation factor calculation

Purpose	Calculate the intrinsic electrical conductivity tensor.
	Compute the intrinsic formation factor.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Axis-Connectivity` (or `10mc3_200.vol.Labels` if step 4 was not completed) and select *XLab Simulations > Effective Formation Factor Calculation*.
- Connect the *ROI Box* to the ROI input connection of the *Effective Formation Factor Calculation* module.
- If step 4 was completed, check the box refering to label 1 in the *Pore space* port, since the

label of the void space in `10mc3_200.vol.Axis-Connectivity` is 1. If step 4 was not completed, check the box refering to label 0 in the *Pore space*, since the label of the void space in `10mc3_200.vol.Labels` is 0.

- Click on *Apply*.

With these parameters, the module will compute the full intrinsic conductivity tensor. A full tensor computation requires three computations, each equivalent in time and memory consumption to one experiment simulation.

Note that the *potential field* output is not selected by default. This output corresponds to the $\overrightarrow{b}$ vector, solution of the vectorial problem derived from the Ohm's equations with the volume averaging approach (see *electrical conductivity simulations theory pages*). It is used in the computation of the effective electrical conductivity but its visualization is hard to interpret.

12.3.8.1 Retrieving and interpreting results

Only the spreadsheet `10mc3_200.vol.FFTensor.Spreadsheet` output is generated in the Project View. The spreadsheet can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.43. It contains information about the computation results gathered in two tables.

- Conductivity tensor table:
    - name of the data set;
    - region of interest, on which the computation occured;
    - full electrical conductivity tensor in a 3 by 3 matrix form (dimensionless);
    - the eigen system solutions for the tensor. The eigenvalues and their associated eigenvector are described on a line.

- Formation factor tensor table:
    - name of the data set;
    - region of interest, on which the computation occured;
    - full formation factor tensor in a 3 by 3 matrix form (dimensionless);
    - the eigen system solutions for the tensor. The eigenvalues and their associated eigenvector are described on a line.

As the formation factor tensor is computed as the inverse of the conductivity tensor, the second table is generated only if all the three directions have been selected and the full conductivity tensor has been computed.

A spreadsheet can be exported into several formats (CSV, XML, txt for example).

	Geometry file	Region of interest	sigma_x	sigma_y	sigma_z	Eigen values	Eigen vector x	Eigen vector y	Eigen vector z
1	10mc3_200...	284.525 284.525...	0.11944524	0.0044346885	-0.0048923502	0.11821773	-0.98314011	0.12698269	-0.13157111
2			0.0041252556	0.15591197	0.0055545033	0.15290774	0.18940859	0.79791749	-0.57223433
3			-0.0047252621	0.0050846385	0.15843366	0.1626654	-0.023946581	0.62627238	0.7792365

	Geometry file	Region of interest	ff_x	ff_y	ff_z	Eigen values	Eigen vector x	Eigen vector y	Eigen vector z	Formation factor
1	10mc3_200...	284.525 284.525...	8.3911762	-0.23120138	0.25768563	8.4589691	0.98314011	-0.12698269	0.13157108	7.0488167
2			-0.24740799	6.4280348	-0.21367492	6.5398912	-0.18940857	-0.79791766	0.57223403	0
3			0.26778907	-0.23249894	6.3272386	6.1475892	-0.02394663	0.62627208	0.77923667	0

Figure 12.43: Tables of the spreadsheet containing the main results of the *Effective Formation Factor Calculation* computation.

12.3.9   Step 8 - Formation factor computation validation

Purpose	Use several empirical laws to check the accuracy of the computed formation factor.
Mandatory step	No.

We base our validation on the study of Lemaitre *et al* (1988). They measured the conductivity (and deduced the formation factor) of porous media made of dense binary mixtures of glass spheres and filled with a conducting fluid. The experiment was performed on mixtures with two diameter ratios of spheres and several porosity values.

It showed that the formation factor depends almost only on porosity and not on the packing structure.

They also reviewed the main results obtained by some of the numerous authors that have tried to relate the formation factor F and the porosity ϵ:

- **Archie's law** (1942): $F = A\epsilon^{-m}$ where A is often assumed to be 1 and m depends on the geometry of the porous medium (m usually belongs to range [1,3]),
- **Sen** *et al* (1981): $F = \epsilon^{-3/2}$ which is a good approximation for sintered packings of glass spheres for $\epsilon > 0.2$.
- **Berryman** (1983): $F = 2/(\epsilon(1 + \epsilon))$ which is well adapted to materials with a large porosity.

It appeared that the experimental values measured on mixtures of glass spheres by Lemaitre *et al* were in a range defined by the values obtained from the theories of Sen *et al* and Berryman and were well fit by Archie's law with m=1.46.

The data set used in this tutorial is a random packing of spheres. We work here with the complete geometry of the complete volume, high-resolution version of the data set (that is to say a label image

obtained from `data/tutorials/xlab/10mc3_400.vol.am`, without any ROI Box).

Porosity can be estimated with Avizo (with *Measure And Analyze > Global Measures > Intensity Integral in 3D interpretation* connected to `10mc3_400.vol.Axis-Connectivity`), and the value is 36.35%.

The formation values calculated from the different empirical laws are:

- **Sen** *et al*: F = 4.563
- **Berryman**: F = 4.035

We compared these values to the values computed with electrical conductivity simulations modules on `data/tutorials/xlab/10mc3_400.vol.am`.

Experiment simulation in each direction gave the following results:

- F_X = 4.599
- F_Y = 4.602
- F_Z = 4.623

The effective formation factor computed is F = 4.714.

Note: we can notice here a considerable difference with the values obtained on a ROI of the subsampled case `10mc3_200.vol.am`. This shows how resampling and using ROI affects the quality of the computed results.

The simulated experimental values and effective value are all of the same order of magnitude as those obtained from the theories of Sen *et al* and Berryman. If we apply Archie's law, the computed values correspond to a m coefficient varying from 1.51 to 1.53, which is very reasonable.

A second demo script is also provided (for Microsoft Windows only), which runs the steps of this tutorial on the full resolution data set, restricted to a region of interest.

Bibliography:

1. Archie G.E., *The electrical resistivity log as an aid in determining some reservoir characteristics*, Petroleum Transactions of AIME, 146, 54-62, 1942
2. Berryman J.G., *Effective conductivity by fluid analogy for a porous insulator filled with a conductor*, Phys. Rev. B 27, 7789-7792, 1983
3. Lemaitre J., Troadec J.P., Bideau D., Gervois A. and Bougault E., *The formation factor of the pores space of binary mixtures of spheres*, J. Phys. D: Appl. Phys., 21, 1589-1592, 1988
4. Sen P.N., Scala C. and Cohen M.H., *A self-similar model from sedimentary rocks with application to dielectric constant of fused glass beads*, Geophysics, 46, 781-795, 1981

12.4   Getting started with thermal conductivity computation

The purpose of this step-by-step tutorial is to become more familiar with the usage of thermal conductivity computation modules provided with the Avizo XLabSuite Extension for Avizo. The following subjects will be addressed in this chapter:

- *theory basics about thermal conductivity*
- *data preparation for simulation (step 1 to step 5)*
- *experiment simulation (step 6)*: define parameters, run simulation, interpret and visualize results
- *effective property calculation (step 7)*: define parameters, run simulation, interpret and visualize results
- *validate results with several empirical laws (step 8).*

As the subject of data preparation for simulation has been addressed in the previous tutorial for *absolute permeability computation*, it will only be shortly evoked here:

- setting voxel size and units,
- segmenting void space,
- removing non-percolating space with Avizo,
- defining a region of interest.

A demo script is also provided (for Microsoft Windows only). This script will automatically perform all the steps detailed in the tutorial.

12.4.1   Theoretical elements

Fourier's law: definition of thermal conduction
Thermal conduction is the ability of a material to conduct heat from high temperature areas to lower temperature areas. Under steady state conditions, heat conduction in a homogeneous material is described by Fourier's law:

$$\vec{\varphi} = -\lambda \vec{\nabla} T$$

where:

- $\vec{\varphi}$ is the heat flux (in $W.m^{-2}$),
- λ is the thermal conductivity of the material (in $W.m^{-1}.K^{-1}$),
- T is the temperature (in K).

Fourier's law in a non-homogeneous material
Most materials are non-homogeneous and contain more than one solid phase. As a consequence, the heat conduction in those materials is more complicated than expressed in Fourier's law.

Transient heat conduction in a given phase named α is described by the partial differential equation:

$$(\rho c_p)_\alpha \frac{\partial T_\alpha}{\partial t} - \lambda_\alpha \nabla^2 T_\alpha = 0$$

where:

- $(\rho c_p)_\alpha$ is the heat capacity of the α phase (in $J.m^{-3}.K^{-1}$);
- ρ_α is the density of the α phase (in $kg.m^{-3}$);
- c_{p_α} is the specific heat of the α phase (in $J.kg^{-1}.K^{-1}$);
- λ_α is the thermal conductivity of the α phase (in $W.m^{-1}.K^{-1}$);
- T_α is the temperature of the α phase (in K).

We consider that the steady state is reached. Then the equation we now want to solve is:

$$\lambda_\alpha \nabla^2 T_\alpha = 0$$

The experiment that is simulated in the *Thermal Conductivity Experiment Simulation* module consists of applying a constant heat flux between two opposite faces of the material sample. For example, the input and output temperatures might be maintained constant by a heating resistor for the input and a freezing bath for the outut. The other faces of the sample are perfect thermal insulator planes. When the final state is attained, input and output heat fluxes are equal and Fourier's law writes on the entire volume:

$$\frac{\varphi_{total}}{S_{in}} = \lambda \frac{T_{in} - T_{out}}{L}$$

where

- φ_{total} is the total heat flux through the input face,
- S_{in} is the area of the input face,
- λ is the apparent thermal conductivity of the material,
- T_{in} and T_{out} are the input and output imposed temperatures,
- L is the length of the material sample.

The apparent thermal conductivity λ can be computed thanks to the previous expression, provided that we know the other terms. The experimenter controls the temperatures T_{in} and T_{out} and it is easy to determine the total heat flux through the input face by locally using Fourier's law:

$$\varphi_{total} = \int_{S_{in}} -\lambda_\alpha \vec{\nabla} T_\alpha . \vec{ds}$$

where λ_α and T_α are the thermal conductivity and temperature of any phase α of the material in contact with the input surface S_{in}.

Multiscale homogeneization theory applied to Fourier's law

The *effective* thermal conductivity tensor gives global information about the thermal conduction capabilities of the material.

The homogeneization theory consists in considering the problem on both a macroscopic domain and a microscopic domain. Here the macroscopic domain is the periodic domain, with a characteristic length L and the microscopic domain is the period which can be considered as representative elementary volume (R.E.V.). The characteristic local length l is the period length and $L >> l$. The characteristic dimensionless space variables x^* and y^* are introduced such that $y^* = X/l$ and $x^* = X/L$ where X is the physics space variable.

Due to the two characteristic variables, the space derivative becomes $\nabla_{x^*} + \epsilon^{-1}\nabla_{y^*}$ where $\epsilon = x^*/y^*$ and so $\epsilon << 1$.

The unknown T is written as an asymptotic development with respect to $\epsilon = x^*/y^*$.

Fourier's partial differential equation is rewritten considering the new space derivative and the asymptotic development of T. Finaly the identification of the terms with ϵ to the same power leads to the resolution of several successive problems among which the following canonical equation on the R.E.V.:

$$\nabla(\lambda_\alpha(\overrightarrow{\nabla}\,\overrightarrow{b}_\alpha + \overrightarrow{\overrightarrow{I}})) = 0$$

where $\overrightarrow{b}$ might be considered as a perturbation of the temperature field. $\overrightarrow{b}$ also verifies:

$$\overrightarrow{\overrightarrow{\lambda_{eff}}} = \frac{1}{V}\sum_\alpha \int_{V_\alpha} \lambda_\alpha \left(\overrightarrow{\nabla}\overrightarrow{b}_\alpha + \overrightarrow{\overrightarrow{I}}\right) dv$$

where:

- $\overrightarrow{\overrightarrow{\lambda_{eff}}}$ is the effective thermal conductivity tensor,
- V is the total volume of the sample,
- α is a conducting phase,
- V_α is the volume occupied by each phase α.

The interested reader will find relevant details and references in [Auriault, J.-L., Boutin, C. and Geindreau, C., *Homogénéisation de phénomènes couplés en milieux hétérogènes 1*, Lavoisier, 2009].

Boundary conditions

As said previously, most materials are not homogeneous and contain more than one phase. Characteristics of each phase can be completely different and the interfaces between the components of the material play a major role in the global thermal conductivity.

Lets consider the case of a two phases material, composed of an α phase and a β phase. This can be generalized to any number of phases. Boundary conditions must be defined for each interface between phases. The system to solve is then:

$$\begin{cases} \lambda_\xi \nabla^2 T_\xi & = 0 & \text{for } \xi = \alpha, \beta \\ T_\alpha & = T_\beta & \text{at } \alpha - \beta \text{ interface} \\ -\overrightarrow{n_{\alpha\beta}}.\lambda_\alpha \overrightarrow{\nabla} T_\alpha & = -\overrightarrow{n_{\alpha\beta}}.\lambda_\beta \overrightarrow{\nabla} T_\beta & \text{at } \alpha - \beta \text{ interface} \end{cases}$$

where $\overrightarrow{n_{\alpha\beta}}$ is the unit vector normal to the interface between the two phases and oriented from α to β.

These boundary conditions specify that the temperature and the normal component of the heat flux are continuous at the interface between the phases.

Avizo XLabSuite Extension proposes two approaches to estimate thermal conductivity.

The first is an experiment simulation based on Fourier's equations resolution. As said previously, this is done in *Thermal Conductivity Experiment Simulation* module. Besides the phases interface condition, boundary conditions are:

- one voxel wide plane of thermal insulator is added on the faces of the image that are not perpendicular to the heat flux main direction. This allows isolating the sample from the outside.
- the input and output (the faces that are perpendicular to the flux main direction) are designed as one voxel wide planes where the temperature is imposed.
- two among the following three conditions can be imposed by the user, the third being estimated from the chosen two: input temperature, output temperature, heat flux.

The second approach solves the canonical problem derived from Fourier's equation by homogeneization on an infinite periodic domain. This is done in the *Thermal Conductivity Tensor Calculation* module. Two boundary conditions result from the condition of continuity of the temperature and of the normal component of the heat flux at a two-phases interface:

$$\begin{cases} \overrightarrow{b}_\alpha & = \overrightarrow{b}_\beta \\ -\lambda_\alpha \overrightarrow{n}_{\alpha\beta}.(\overrightarrow{\nabla}\overrightarrow{b}_\alpha + \overrightarrow{\overrightarrow{I}}) & = -\lambda_\beta \overrightarrow{n}_{\alpha\beta}.(\overrightarrow{\nabla}\overrightarrow{b}_\beta + \overrightarrow{\overrightarrow{I}}) \end{cases}$$

A periodic boundary condition is imposed to $\overrightarrow{b}_\alpha$ and to the geometry.

Discretization of the equation system

Avizo XLabSuite Extension uses a finite volume method to solve the equation systems.

The discretization scheme supposes that the voxel is isotropic (cubic).

System resolution

Once discretized, the equation systems can be written as $Ax = b$, A being a sparse, symmetric matrix.

The equation systems are solved using a fully implicit method (matrix inversion). PETSc (Portable, Extensible Toolkit for Scientific Computation) library is used for the direct resolution of the linear systems.

An iterative resolution with conjugate gradient and ILU preconditioner is performed. The convergence criterion used is the relative decrease of the residual l_2-norm.

12.4.2   Step 1 - Load the data set

Purpose	Load the data set in Avizo.
Mandatory step	Yes.

The data set that is used in this tutorial is a random glass spheres packing. It was scanned by Dominique Bernard, Research Director at ICMCB-CNRS. The spherical particles were sieved to have diameters in the 100-120 microns range. The sample was sintered for 10 minutes at $700°$ Celsius prior to scanning. Figure 12.44 shows the data set.

The complete 3D data set used in this tutorial is a $200{\times}200{\times}200$ cube.

- Start a new Project *File > New Project*
- Open the *File > Open Data...* dialog.
- Open the `data/tutorials/xlab/10mc3_200.vol.am` file from `AVIZO_ROOT` directory.
- Click on *Open*.
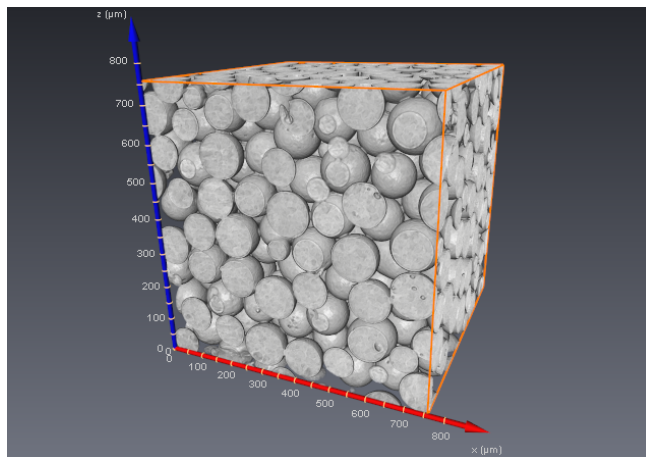
Figure 12.44: 3D visualization of the sphere pack used in this tutorial.

As mentioned in the previous step, unit management is not mandatory here. However, refer to *step 2* of absolute permeability simulations tutorial to learn more about editing the length units.

12.4.3 Step 2 - Set the voxel size

Purpose	Set the edge size of the voxels in the uniform data set.
Mandatory step	No.

Again, as mentioned in the previous steps, correctly setting the dimensions of the voxels is not mandatory here. However, to learn more about using the *Crop Editor* to set the voxel size, please refer to *step 3* of absolute permeability simulations tutorial.

Note: Isotropic, i.e., cubic, voxels are mandatory for Avizo XLabSuite Extension computations.

12.4.4 Step 3 - Create a label image from the data set

Purpose	Use basic segmentation tools to create a label image from the data set.
Mandatory step	Yes.

This step consists of creating a label image from the initial data set that is smoothed and thresholded, using some of the segmentation tools provided by Avizo.

Please follow the instructions described in *step 4* of absolute permeability simulations tutorial. The resulting visualization should look like Figure 12.45.
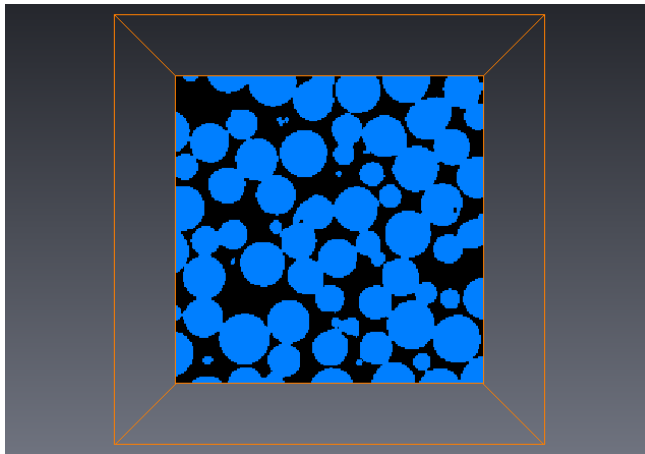
Figure 12.45: Visualization of the thresholded data set.

12.4.5 Step 4 - Remove isolated conducting materials parts

Purpose	Use advanced segmentation tools to remove parts of the conducting materials that are surrended by isolating materials.
Mandatory step	No. Only available with Avizo.

In case one or several phases of the material are considered as thermal insulators (and only in this case), it is strongly advised to run a test that will detect and remove parts of the conducting materials that are completely surrounded by insulators. The removed parts will be considered as insulators as shown on 12.46.

This will avoid the risk of building a singular matrix for the transport equation resolution. The test can be performed before running the computation. The necessary connectivity, from a discrete point of view, in 6-connectivity: heat can be transmitted through a non-nil surface only, i.e., a face between two neighbor voxels.

The underlying idea of this test is the same as the idea of the non-percolating test performed in *step 5* of absolute permeability simulations tutorial. Therefore, the same tools can be used to perform both tests.

In the present tutorial, we will consider that all the phases of our material are conducting heat but in a case where you have to consider both isolating and conducting phases, you might follow the instructions described in *step 5* of absolute permeability simulations tutorial. The equivalent of the void space where the fluid flows in absolute permeability simulations tutorial is here the heat conducting material and the equivalent of the porous material is here the insulator.
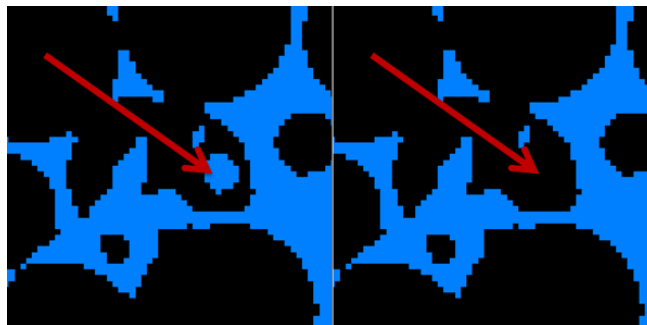
Figure 12.46: Removal of voxels belonging to a conductig material but surrended with insulator material.

12.4.6 Step 5 - Selection of a sub-region

Purpose	Define a region of interest for the thermal conductivity computation.
Mandatory step	Yes.

There is an easy way to compute thermal conductivity on a sub-region of the loaded data set without

cropping it. A region of interest (ROI) can be defined and connected to Avizo XLabSuite Extension modules, so that the computation is only performed in the ROI.

Please follow the instructions given in *step 6* of absolute permeability simulations tutorial and define a ROI cube with 50-voxel edges centered in the sample (see Figure 12.47). The ROI will be used in this tutorial to compute properties on small volumes. It can be moved through the data set at will during the tutorial, although its initial position will be used for illustrating purposes.

For convenience and because it will save time during this tutorial phase, we compute the thermal conductivity on a sub-region of the sample. However, we will see in the validation phase how resampling and using ROI affects the quality of the computed results.
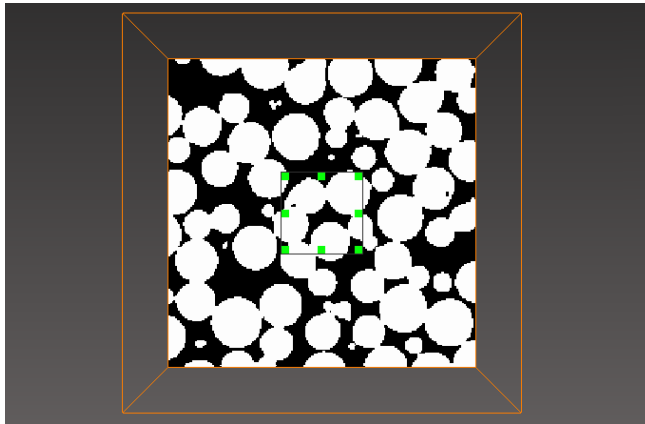
Figure 12.47: Visualization of the cube representing the region of interest.

12.4.7 Step 6 - Thermal conductivity experiment simulation

Purpose	Simulate a thermal conductivity laboratory measurement. Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Labels` and select *XLab Simulations > Thermal Conductivity Experiment Simulation*.
- Connect the *ROI Box* to the ROI input connection of the *Thermal Conductivity Experiment Simulation* module.
- In the *Conducting Materials* port, check the boxes for both of the two phases.
- Set the thermal conductivity of phase Inside to 1000 $W.m^{-1}.K^{-1}$ in the *Thermal Conductivity* port, and leave the thermal conductivity of phase Exterior (corresponding to the void space) to the default value 1 $W.m^{-1}.K^{-1}$.

- Click on *Apply*.

The default parameters simulate an experiment along the Z axis with an input temperature of 298 K and an output temperature of 273 K. The options can be modified to simulate several experiments. For example, the direction of the thermal flux can be adjusted to X, Y or Z direction (default is Z). If several directions are selected, the computations will be done successively. The boundary conditions of the experiment can also be modified. Two values among three must be imposed: input temperature, output temperature, heat flux. Modifying these values will not change the thermal conductivity, which is intrinsic to the material. It will only modify the output temperature field.

12.4.7.1 Retrieving and interpreting results

Two outputs appeared in the Project View:

- `10mc3_200.vol.TCExp.Spreadsheet` : a spreadsheet containing the most relevant results of the computation:

    - name of the data set;
    - region of interest, on which the computation is done;
    - apparent thermal conductivity of the material restricted to the ROI in $W.m^{-1}.K^{-1}$;
    - temperature at input of the experimental setup in K;
    - temperature at output of the experimental setup in K;
    - heat flux through the sample in $W.m^{-2}$;
    - thermal conductvities of the phases of the material in $W.m^{-1}.K^{-1}$.

- `10mc3_200.vol.TemperatureZ` : a scalar field representing the temperature field (in K), solution of the Fourier's equation system solved in the geometry restricted by the ROI.

The spreadsheet containing all the relevant information can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.48. A spreadsheet can be exported into several formats (CSV, XML, txt for example).

	Geometry file	Region of interest	Apparent thermal conductivity [W.m^-1.K^-1]	Exterior thermal conductivity [W.m^-1.K^-1]	Inside thermal conductivity [W.m^-1.K^-1]
1	10mc3_200.Labels	284.525 284.525...	296.4754	1	1000

Z

Figure 12.48: Spreadsheet containing the main results of the *Thermal Conductivity Experiment Simulation* computation.

12.4.7.2 Visualizing output temperature field

To visualize the temperature field:

- Hide all current display modules, such as *Bounding Box* or *Ortho Slice*, by clicking on their viewer toggle.
- Right-click on `10mc3_200.vol.TemperatureZ` and select *Display > Ortho Slice*.
- In the *Ortho Slice* properties, change the selected colormap to *temperature.icol* from the *Edit* menu.
- Select the *Edit* menu of the colormap again, and then select *Adjust range to > Data min-max*.
- In the *Orientation* port, select the *yz* orientation.
- Zoom on the slice to visualize it.

The resulting visualization should look like Figure 12.49. You can observe the decrease of the temperature from the input of the device (at the bottom of the slice, in yellow), to the output of the device (at the top of the slice, in light blue). You can also guess that the heat front is moving forward faster in the phase with the greatest thermal conductivity.
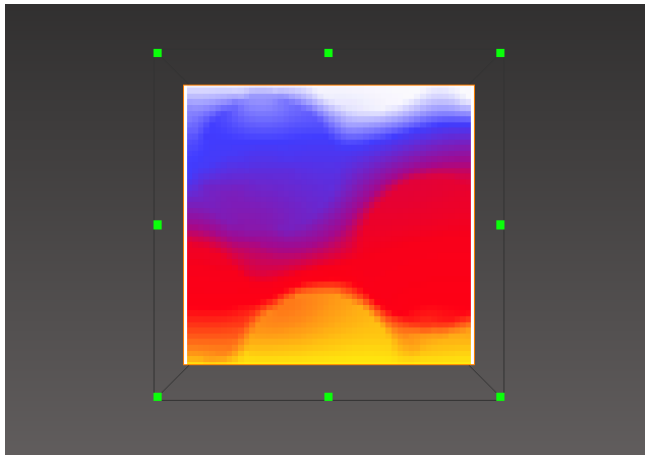
Figure 12.49: Visualization of temperature field in an experiment simulation with thermal flux in Z direction.

To emphasize this phenomenon, we will use some other display tools.

- Attach a *Color Wash* module to the *Ortho Slice*.
- In the *Data* port of the *Color Wash* module, select `10mc3_200.vol.Labels`.
- Set the *Weight Factor* to 0.75.

Now it is easier to visualize the particules of phase Inside and the influence of their higher thermal conductivity on the heat front.

- Attach an *Isosurface* module to `10mc3_200.vol.TemperatureZ` (*Display > Isosurface*).
- Check the *auto-refresh* box at the bottom of the *Properties area*.
- Move the *Threshold* cursor to visualize the isosurface for different temperature values.

The aspect of the isosurface contours highlights the fact that the heat front is moving forward faster in the particules of phase Inside. (If this is not obvious, you can use a *Volume Rendering* to help visualize the particules in 3D).

- Attach an *Isocontour Slice* module to `10mc3_200.vol.TemperatureZ` (*Display > Isocontour Slice*).
- Set the *Clipping Plane Orientation* to *yz* and the position to 51 in the *Translate* port.
- In the *Properties area* of the *Isocontour Slice*, set the *Values* range to 273, 298 and the number of lines to 30.
- Set the *line width* to 1 from the *Parameters* port.

Again, the heat front, represented by the isolines, obviously moves faster in the particules of phase Inside.

The resulting visualization should look like Figure 12.50.
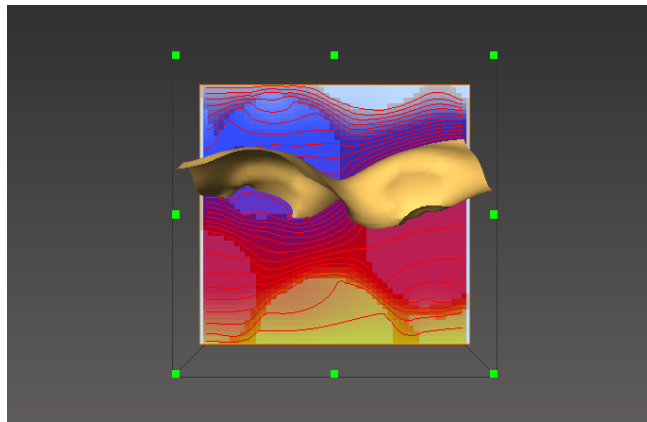
Figure 12.50: Visualization of the temperature isosurface T = 284 K and of 30 temperature isolines on an Ortho Slice with Color Wash rendering.

12.4.8   Step 7 - Effective thermal conductivity calculation

Purpose	Calculate the intrinsic thermal conductivity tensor.
	Visualize and interpret the results of the calculation.
Mandatory step	Yes.

- Right-click on `10mc3_200.vol.Labels` and select *XLab Simulations > Thermal Conductivity Tensor Calculation*.
- Connect the *ROI Box* to the ROI input connection of the *Thermal Conductivity Tensor Calculation* module.
- In the *Conducting Materials* port, check the boxes for both of the two phases.
- Set the thermal conductivity of phase Inside to 1000 $W.m^{-1}.K^{-1}$ in the *Thermal Conductivity* port, and leave the thermal conductivity of phase Exterior (corresponding to the void space) to the default value 1 $W.m^{-1}.K^{-1}$.
- Click on *Apply*.

With these parameters, the module will compute the full intrinsic thermal conductivity tensor. A full tensor computation requires three computations, each equivalent in time and memory consumption to one experiment simulation.

Note that the *temperature field* output is not selected by default. This output corresponds to the $\vec{b}$ vector, solution of the vectorial problem derived from the Fourier's equation with the homogeneization approach (see *thermal conductivity simulations theory pages*). It is used in the computation of the effective thermal conductivity but its visualization is hard to interpret.

12.4.8.1   Retrieving and interpreting results

Only the spreadsheet `10mc3_200.vol.TCTensor.Spreadsheet` output is generated in the Project View. The spreadsheet can be visualized by selecting it in the Project View and clicking on the *Show* button (in the *Properties area*). See Figure 12.51. It contains information about the computation results gathered in two tables.

- name of the data set,
- region of interest, on which the computation occured,
- full thermal conductivity tensor in a 3 by 3 matrix form (in $W.m^{-1}.K^{-1}$),
- the eigen system solutions for the tensor: the eigenvalues and their associated eigenvectors are described on a line,
- the thermal conductivities of the phases of the material in $W.m^{-1}.K^{-1}$.

A spreadsheet can be exported into several formats (CSV, XML, txt for example).

	Geometry file	Region of interest	tc_x	tc_y	tc_z	Eigen values	Eigen vector x	Eigen vector y	Eigen vector z	Exterior thermal conductivity [W.m^-1.K^-1]	Inside therm
1	10mc3_200....	284.525 284.525...	392.76132	-7.8175116	-2.1636078	394.7778	0.97024798	-0.2268604	-0.084576368	1	1000
2			-7.8133755	354.77756	17.659267	360.56644	-0.24167193	-0.88535875	-0.39715815		
3			-2.1698551	17.654089	322.53171	314.72635	-0.015326614	-0.40589601	0.91379076		

Full tensor

Figure 12.51: Tables of the spreadsheet containing the main results of the *Thermal Conductivity Tensor Calculation* computation.

12.4.9   Step 8 - Thermal conductivity computation validation

Purpose	Use experimental results and empirical laws to check the accuracy of the computed thermal conductivity.
Mandatory step	No.

We base our validation on several empirical laws aimed at computing the thermal conductivity of materials with several very specific geometries. The "inside" material (or geometry) is refered to as β-phase with thermal conductivity λ_β and volume fraction ϵ_β. The material surrending it is refered to as the α-phase with thermal conductivity λ_α and volume fraction ϵ_α. We define the ratio $\Lambda = \lambda_\beta/\lambda_\alpha$ and $\Lambda_{eff} = \lambda_{eff}/\lambda_\alpha$ where λ_{eff} is the effective conductivity of the whole material.

12.4.9.1   Periodic array of non-touching 3d cylinders

The first model we study is a periodic array of non-touching 3d cylinders, as displayed on Figure 12.52. The cylinders are refered to as the β-phase and the material surrending the cylinders as the α-phase.
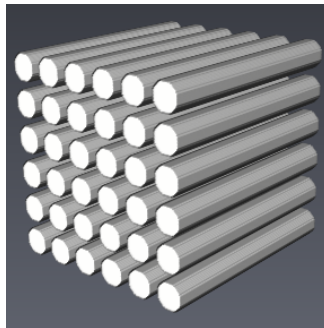
Figure 12.52: Example of periodic array of non-touching 3d cylinders.

In Ochoa-Tapia *et al.* (1994), the following analytical estimation of the effective thermal conductivity

of an array of non-touching cylinders in series distribution (where the two phases are thermally in series with respect to the direction of the heat flux) is given:

$$\Lambda_{eff} = \frac{2\Lambda - \epsilon_\alpha(\Lambda - 1)}{2 + \epsilon_\alpha(\Lambda - 1)}$$

In Perrins *et al.* (1979), analytical estimations of the effective thermal conductivity are gathered in a table of values of conductivities for square arrays of cylinders with various conductivities ratio Λ and volume fractions.

We apply the formula by Ochoa-Tapia *et al.* to our model, where $\epsilon_\alpha = 0.49$, and we compare it with the results obtained from both Avizo experiment simulation and tensor computation and with the values of thermal conductivity given in Perrins *et al.* for $\epsilon_\alpha = 0.5$ (as 0.49 is not in the table):

	Ochoa-Tapia *et al.*	Perrins *et al.*	Avizo TCExp	Avizo TCTensor
Λ	Λ_{eff}	Λ_{eff}	Λ_{eff}	Λ_{eff}
2	1.406	1.401	1.397	1.397
3.5	1.783	1.776	1.765	1.765
5	2.020	2.013	1.999	1.999
10	2.416	2.416	2.395	2.395
20	2.692	2.701	2.677	2.677
50	2.897	2.915	2.890	2.890
100	2.972	na	2.970	2.970
1000	3.045	na	3.046	3.045
10000	3.053	na	3.054	3.054
∞	3.053	3.080	na	na

We observe a good agreement between the analytical estimations and the simulations.

In Meredith and Tobias (1960), the following analytical estimation of the effective thermal conductivity of an array of non-touching cylinders in parallel distribution (where the two phases are thermally in parallel with respect to the direction of the heat flux) is given:

$$\Lambda_{eff} = \epsilon_\alpha + (1 - \epsilon_\alpha)\Lambda$$

We apply the formula to our model, where $\epsilon_\alpha = 0.49$, and we compare it with the results obtained from Avizo experiment simulation and tensor computation:

	Meredith and Tobias	Avizo TCExp	Avizo TCTensor
Λ	Λ_{eff}	Λ_{eff}	Λ_{eff}
10	5.559	5.474	5.474
100	51.152	50.218	50.217
1000	507.08	497.65	497.65
10000	5066.4	4972.0	4971.9

Again, we observe a good agreement between the analytical estimation and the simulations.

Whitaker (1999) reports the results computed by Nozad *et al.* (1985) for a simple two-dimensional square unit cells periodic array and shows that the results obtained are similar to those of Perrins *et al.* (1979). On Figure 12.53 we compare Nozad *et al.* results of Λ_{eff} w.r.t. Λ for various ϵ_α values with Avizo simulations and see that this last curve is well located and has a correct behavior. Only one curve is displayed for both Avizo experiment simulation and tensor computation as their results are almost equal.
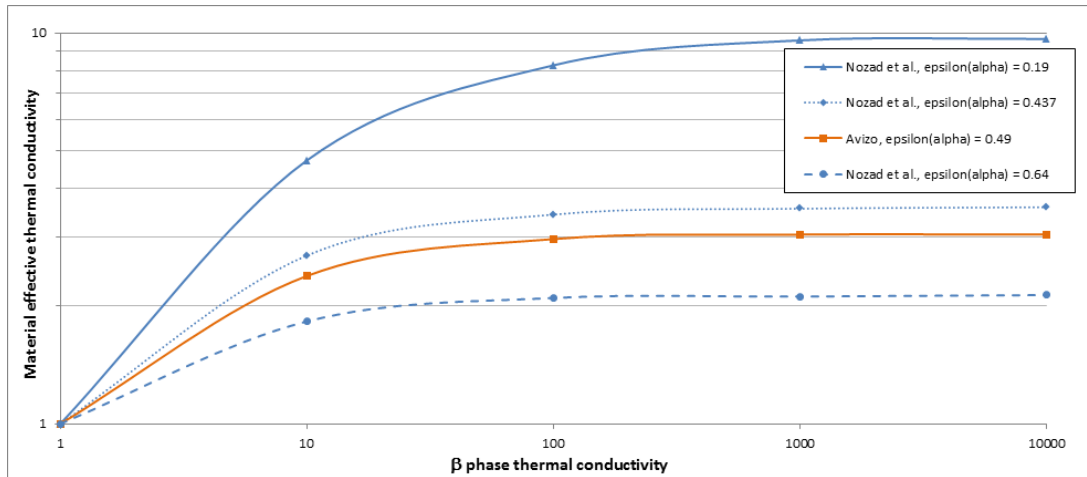
Figure 12.53: Comparison between Avizo simulation and Nozad *et al.* computation of thermal conductivity.

12.4.9.2 Periodic array of non-touching 3d spheres

The second model we study is a periodic array of non-touching 3d spheres, as displayed on Figure 12.54. The spheres are refered to as the β-phase and the material surrending the spheres as the α-phase.
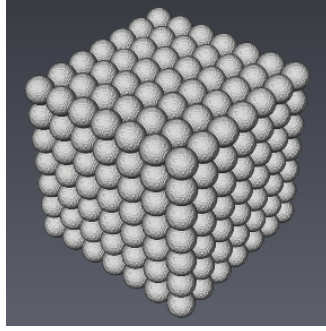
Figure 12.54: Example of periodic array of non-touching 3d spheres.

Maxwell (1881) obtained the following expression for the thermal conductivity of a packed-sphere bed where the spheres are sufficiently apart that they do not interact:

$$\Lambda_{eff} = \frac{3\Lambda - 2\epsilon_\alpha(\Lambda - 1)}{3 + \epsilon_\alpha(\Lambda - 1)}$$

Meredith and Tobias (1960) gave an analytical expression for the same problem:

$$\Lambda_{eff} = \frac{\dfrac{2+\Lambda}{1-\Lambda} - 2\epsilon_\beta + 0.409\dfrac{6+3\Lambda}{4+3\Lambda}\epsilon_\beta^{7/3} - 2.133\dfrac{3-3\epsilon_\beta}{4+3\epsilon_\beta}\epsilon_\beta^{10/3}}{\dfrac{2+\Lambda}{1-\Lambda} + \epsilon_\beta + 0.409\dfrac{6+3\Lambda}{4+3\Lambda}\epsilon_\beta^{7/3} - 0.906\dfrac{3-3\Lambda}{4+3\Lambda}\epsilon_\beta^{10/3}}$$

We apply the two formula to our model, where $\epsilon_\alpha = 0.64$, and we compare it with the results obtained from both Avizo experiment simulation and tensor computation:

	Maxwell	Meredith and Tobias	Avizo TCExp	Avizo TCTensor
Λ	Λ_{eff}	Λ_{eff}	Λ_{eff}	Λ_{eff}
10	2.110	2.243	2.138	2.138
100	2.611	2.884	2.732	2.732
1000	2.680	2.976	2.820	2.820
10000	2.687	2.986	2.830	2.830

We observe a good agreement between the analytical estimations and the simulations.

12.4.9.3 Periodic array of 3d spheres with particle-particle contact

The third model we study is a periodic array of 3d spheres with particle-particle contact, as displayed on Figure 12.55. The contact surface is controled by the ratio between the contact size and the spheres

radius. The spheres are refered to as the β-phase and the material surrending the spheres as the α-phase.
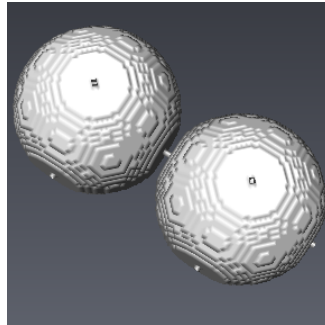
Figure 12.55: Two spheres in particle-particle contact extracted from a periodic array.

Whitaker (1999) reports the results computed by Nozad *et al.* (1985) on a two-dimensional square array with particle-particle contact. In there study Nozad *et al.* compared those results with experimental measurements on a narrow range of porosities (ϵ_α in [0.39, 0.41]). The theory and experiment agreed well, despite the important differences between their characteristics. Based on this observation, we compare Nozad *et al.* computational results with Avizo simulations on our model, where $\epsilon_\alpha = 0.51$. Only one curve is displayed for both Avizo experiment simulation and tensor computation as their results are almost equal.
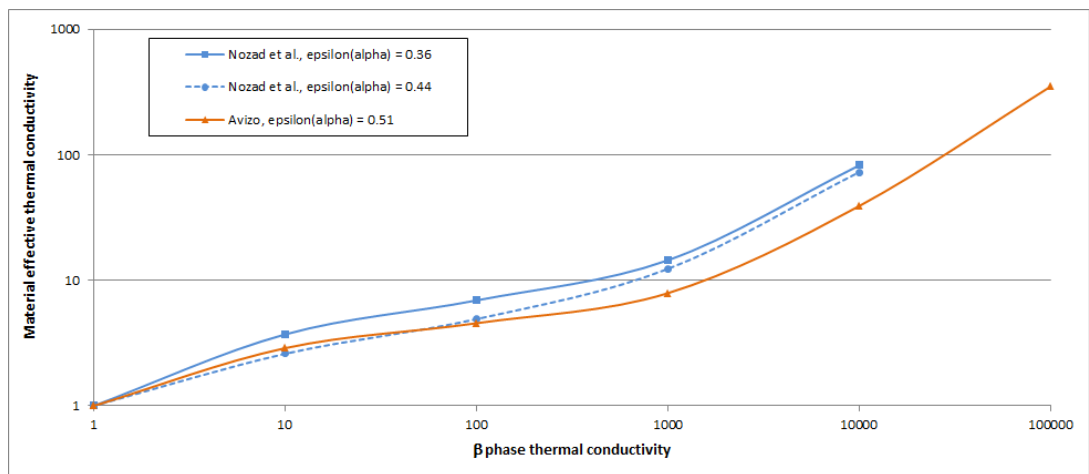
Figure 12.56: Comparison between Avizo simulation and Nozad *et al.* computation of thermal conductivity.

Getting started with thermal conductivity computation **734**

We observe that Avizo curve behavior is correct and its position w.r.t. the other curves is mostly acceptable.

12.4.9.4 Periodic bilaminar composit material

The last model we study is a periodic bilaminar composit material, as displayed on Figure 12.57. The thinest layers material is refered to as the β-phase and the other material as the α-phase. We define r, the thickness ratio between the layers. The stratification is supposed to be in the Z direction.
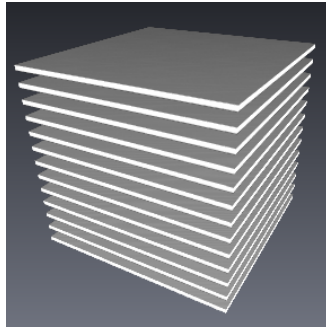
Figure 12.57: Example of periodic bilaminar composit material.

Auriault (2009) presents an analytical expression of the effective thermal conductivity tensor for such a bilaminar material:

$$\overrightarrow{\overrightarrow{\lambda}}_{eff} = \begin{pmatrix} r\lambda_\alpha + (1-r)\lambda_\beta & 0 & 0 \\ 0 & r\lambda_\alpha + (1-r)\lambda_\beta & 0 \\ 0 & 0 & \frac{\lambda_\alpha \lambda_\beta}{r\lambda_\alpha + (1-r)\lambda_\beta} \end{pmatrix}$$

We compare the analytical expression with the result of Avizo tensor computation for two pairs of thickness ratio and thermal conductivity ratio:

r	Λ	Auriault $\overrightarrow{\overrightarrow{\lambda}}_{eff}$			Avizo TCTensor $\overrightarrow{\overrightarrow{\lambda}}_{eff}$		
$\frac{1}{3}$	0.02	$\begin{pmatrix} 33.667 & 0 & 0 \\ 0 & 33.667 & 0 \\ 0 & 0 & 2.885 \end{pmatrix}$			$\begin{pmatrix} 33.667 & 0 & 0 \\ 0 & 33.667 & 0 \\ 0 & 0 & 2.885 \end{pmatrix}$		
$\frac{1}{4}$	0.01	$\begin{pmatrix} 75.25 & 0 & 0 \\ 0 & 75.25 & 0 \\ 0 & 0 & 3.883 \end{pmatrix}$			$\begin{pmatrix} 75.25 & 0 & 0 \\ 0 & 75.25 & 0 \\ 0 & 0 & 3.884 \end{pmatrix}$		

The results obtained through Avizo tensor computation are in perfect agreement with the analytical expression.

Bibliography:

1. Maxwell J.C., *Treatise on Electricity and Magnetism*, Vol. I, 2nd edition, Clarendon Press, Oxford, 1881

2. Meredith R.E., Tobias C.W., *Resistance to potential flow through a cubical array of spheres*, J. Applied Phys., Vol. 31, 1270-1273, 1960

3. Nozad I., Carbonell R.G., Whitaker S., *Heat conduction in multiphase systems I: Theory and experiment for two-phase systems*, Chem. Engng. Sci. 40, 843-855, 1985

4. Ochoa-Tapia J.A., Stroeve P., Whitaker S., *Diffusive transport in two-phase media: Spatially periodic models and Maxwell's theory for isotropic and anisotropic systems*, Chem. Engng. Sci. 49, 709-726, 1994

5. Perrins W.T., McKenzie D.R., McPhedran R.C., *Transport properties of regular arrays of cylinders*, Proc. Roy. Soc. Lond. A369, 207-225, 1979

6. Whitaker S., *The method of volume averaging*, Theory and applications of transport in porous media Vol. 13, Kluwer Acad. Pub., Dordrecht, 1999

7. Auriault, J.-L., Boutin, C., Geindreau, C., *Homogénéisation de phénomènes couplés en milieux hétérogènes 1*, Lavoisier, 2009

Chapter 13

Avizo XPand Extension

This document describes how to develop custom extensions for the Avizo visualization system. Such extensions may include file read and write routines, new visualization modules, data objects and other components. In order to be able to develop custom extensions, you will need the developer extension for Avizo, called Avizo XPand Extension for short. In addition, you will need a C++ development environment like Microsoft Visual Studio on Windows (for details see subsection 13.1.2).

To understand the document, you need some basic know-how in C++ programming. Also, you should be already familiar with the standard Avizo system. If you do not know Avizo yet, we recommend that you go through the tutorials in the *Avizo User's Guide*.

Avizo is based on a number of external libraries, such as *Open Inventor, OpenGL, Qt*, and *Tcl*. This document does not provide a documentation for these libraries. It merely gives some basic hints where it is needed to understand the examples. In general, this will be sufficient to write your own standard I/O routines and Avizo modules. For details, we refer to the original documentation of the external libraries.

- *Introduction*, including a *quick start guide* and *upgrade information*
- *The Development Wizard*, a tool for facilitating development tasks
- *File IO*, describing how to write read and write routines
- *Writing Modules*, covering compute and display modules
- *Data Classes*, how to use them and how to derive from them
- *Documenting Modules* and integration into the user's guide
- Documentation Markup Language, available in the online documentation, describing all commands that can be used when documenting Avizo resources
- *Miscellaneous*, resource file summary, saving Avizo projects, and more.

13.1 Introduction to Avizo XPand Extension

Avizo XPand Extension allows you to add to Avizo new components such as file read or write routines, modules for visualizing data or modules for processing data. New module classes and new data classes can be defined as subclasses of existing ones.

Note that it is not possible (or possible only to a very limited extent) to change or modify existing modules or parts of Avizo's graphical user interface.

In the following sections we

- present an *overview of Avizo XPand Extension*,
- discuss the *system requirements* for the different platforms,
- outline the *structure of the Avizo file tree*,
- show how to compile the example package in a *quick start tutorial*,
- provide additional hints on *compiling and debugging*,
- and mention how to *upgrade and maintain existing code*.

Note: an Avizo project is actually a set of interconnected modules and data objects. In Avizo XPand Extension programming interface, the Avizo project is often referred as the *network* or *module network*, or the *object pool*. The *pool* also refers to Avizo Project View, more specifically the Project Graph View.

13.1.1 Overview of Avizo XPand Extension

Avizo XPand Extension is an extension to the ordinary Avizo version. In addition to the files contained in the ordinary version, the developer version essentially provides all C++ header files needed to compile custom extensions.

13.1.1.1 Packages and Shared Objects

Avizo is an object-oriented software system. Besides the core components like the graphical user interface or the 3D viewers, it contains a large number of data objects, modules, readers and writers. Data objects and modules are C++ classes, readers and writers are C++ functions.

Instead of being compiled into a single static executable, these components are grouped into *packages*. A package is a shared object (usually called .so or .sl on Unix or .dll on Windows) which can be dynamically loaded by Avizo at runtime when needed. This concept has two advantages. On the one hand, the program remains small since only those packages are loaded which are actually needed by the user. On the other hand, it provides almost unlimited extensibility since new packages can be added any time without recompiling the main program.

Therefore, in order to add custom components to the Avizo developer version, new packages or shared objects must be created and compiled. A package may contain an arbitrary number of modules and it

is left up to the developer whether he wants to organize his modules into several packages or just in one.

13.1.1.2   Package Resource Files

A *resource file* is stored along with each package. This file contains information about the components being defined in a particular package. When Avizo starts, it first scans the resource files of all available packages and thus knows about all the components which may be used at runtime.

The resource files of the standard Avizo packages are located under share/resources in the directory where Avizo is installed. Details about registering read and write routines or different kinds of modules in a resource file are provided in sections 13.3 and 13.4.

13.1.1.3   The Local Avizo Directory

Usually Avizo will be installed by the system administrator at a location where ordinary users are not allowed to create or modify files. Therefore it is recommended that every user creates new packages in his own personal *local Avizo directory*. The local Avizo directory has essentially the same structure as the directory where Avizo is installed. A new local Avizo directory can most easily be created by using the *Development Wizard*, a special-purpose dialog box described in detail in section 13.2.

Once a local Avizo directory has been set up, resource files located in it will also be scanned by Avizo when started. In this way, new components can be added or existing ones redefined.

13.1.1.4   External Libraries

Avizo is based on a number of industry standard libraries. The most important ones are *Open Inventor*, *OpenGL*, *Qt*, and *Tcl*.

Avizo's 3D graphics is based on OpenGL and Open Inventor. OpenGL is the industry standard for professional 3D graphics. Open Inventor is a C++ library using OpenGL which provides an object-oriented scene description layer. Writing new visualization modules for Avizo essentially means creating an Open Inventor scene from the input data. If you already have code doing this, it will be straightforward to turn it into an Avizo module. While the Open Inventor headers are included in Avizo XPand Extension, OpenGL must already be installed on your system.

Qt is a platform-independent C++ library for building graphical user interfaces (GUIs). Avizo is built with Qt. However, the user interface elements used in standard Avizo modules are encapsulated by special Avizo classes called *ports*. Therefore, you can develop your own modules without knowing Qt. You only need Qt if you plan to add completely new user interface components such as special purpose dialogs. Note also, that in this case Qt headers and libraries are included in Avizo XPand Extension under LGPL licensing. Avizo 9.4 is linked against Qt 5.4 under Microsoft Windows and Qt 4.8 under Linux and Mac OS X.

Finally, Tcl is a C library providing an extensible scripting language used by Avizo. All required header files are included in Avizo XPand Extension. Avizo programmers usually need not know details of the Tcl API but merely derive their code from existing examples.

13.1.2 System Requirements

In order to develop new Avizo components as described in this document, you need the developer extension for Avizo (called Avizo XPand Extension) and the debugging files for Avizo. Both can be installed with the Avizo installer. You also need the appropriate C++ development environment. C++ compilers are generally not compatible, therefore the compilers and compiler versions listed in section *System Requirements* should be used. Other compiler versions may work too, but this is not guaranteed. In particular, it is not possible to use the GNU gcc compiler except on Linux or Mac.

On all Unix platforms, the GNU make utility (`gmake`) is needed in order to use the GNUmakefiles provided with Avizo XPand Extension. To proceed, you should create a link in a directory already listed in your path, e.g., in `/usr/bin`.

On Mac OS X platforms, the GNU make utility (`gmake`) is basically needed in order to use the GNUmakefiles provided with Avizo XPand Extension. You can also modify and build your code by using other development tools based on GNU gcc compilers as Xcode.

More general hardware requirements such as installed memory or special graphics adapters are listed in the Avizo *User's Guide*. On all systems, an OpenGL library together with the OpenGL header files must be installed.

13.1.3 Structure of the Avizo File Tree

Like the ordinary version, the developer version of Avizo (with Avizo XPand Extension) is installed in a single directory called the *AvizoRoot Directory*. This directory contains all the binaries, shared objects, and resource files required to run Avizo, as well as all the C++ header files required to compile new components. Note that the installation of debug binaries must be enabled during Avizo installation to compile new components in debug. New components themselves are stored independently in the *AvizoLocal Directory*. Every user may define his/her own local Avizo directory. The local Avizo directory has a structure very similar to the AvizoRoot Directory. The structure of these two directories is described in more detail in the following two sections.

13.1.3.1 The Avizo Root Directory

The contents of the Avizo root directory may differ slightly from platform to platform. For example, on Windows, there will be no subdirectory `lib`. Instead, the compiled shared objects are located under `bin/arch-*-Optimize`. The online documentation directory (`share/devref/oiv`) of *Open Inventor* C++ classes. The online documentation directory (`share/devrefAvizo/`) of Avizo C++ classes does not exist on Windows. Instead, a compressed archive file `Avizo.chm` is provided and is accessible by shortcut. This is how a typical Avizo installation directory looks like:

```
AvizoRoot/..........................................................Avizo installation directory
│  bin/
│  │  arch-*-Debug/......................................Avizo debug binaries (Windows only)
│  │  │  Avizo.exe........................................Avizo debug executable (Windows only)
│  │  arch-*-Optimize/..............................................Avizo binaries (Windows)
│  │  │  Avizo.exe..............................................Avizo executable (Windows)
│  │  start..............................................................Avizo start script (Unix)
│  include/..............................................................C++ header files
│  make/....................................................Make environment for Unix systems
│  share/
│  │  resources/...........................................Resource files of all standard packages
│  │  devref/..........................................Documentation of Open Inventor C++ classes
│  │  devrefAvizo/.......................................Documentation of Avizo C++ classes
│  │  doc/..................................................................Avizo documentation
```

13.1.3.2 The Local Avizo Directory

The local Avizo directory contains the source code and object files of custom modules, the resource files of custom packages, and the compiled custom packages themselves. The packages can be compiled either in a debug version or in an optimized version. The corresponding object files and compiled shared objects reside in different subdirectories called `arch-*-Debug` and `arch-*-Optimize`, respectively. Here the asterisk denotes the particular architecture, e.g., `Win64VC12` for Windows systems or `Linux, LinuxAMD64` for Linux platforms.

The *Development Wizard* should be used to create a new local Avizo directory. For details please refer to section 13.2. Subdirectories like `AvizoLocal/lib` or `AvizoLocal/share/resources` are created automatically the first time a custom package is compiled. Again, the contents of the local Avizo directory may differ slightly from platform to platform. For example, on Windows compiled shared objects are located under `bin` instead of `lib`.

```
AvizoLocal/
│  AvizoLocal.vc120.sln...................................Visual Studio solution file (Windows)
│  GNUmakefile.......................................................Global makefile (Unix)
│  src/.......................................Directory containing source code of custom packages
│  │  mypackage/.......................Directory containing source code of one particular package
│  │  │  Package............................Configuration file used to generate make / project files
│  │  │  GNUmakefile............................................................Unix makefile
│  │  │  mypackage.vc120.vcproj...................................Visual Studio project file
│  │  │  api.h................................................Windows storage-class specification
│  │  │  MyModule.h.............................................Header file of a custom module
│  │  │  MyModule.cpp.......................................Implementation of a custom module
│  │  │  MyReader.cpp....................................Implementation of a custom read routine
│  │  │  share/
│  │  │  │  resource/
│  │  │  │  │  mypackage.rc.............................................Package resource file
```

```
├── obj/
│   ├── arch-*-Debug/ .......................................... Object files on Unix (debug version)
│   └── arch-*-Optimize/ ......................................... Object files on Unix (optimized)
├── lib/
│   ├── arch-*-Debug/ ...................................... Compiled debug shared objects (Unix)
│   └── arch-*-Optimize/ ............................... Compiled optimized shared objects (Unix)
├── bin/
│   ├── arch-Win*-Debug/ ..................................... Compiled debug binaries (Windows)
│   └── arch-Win*-Optimize/ .............................. Compiled optimized binaries (Windows)
└── share/
    └── resources/ .......................................... Package resource files are copied here
```
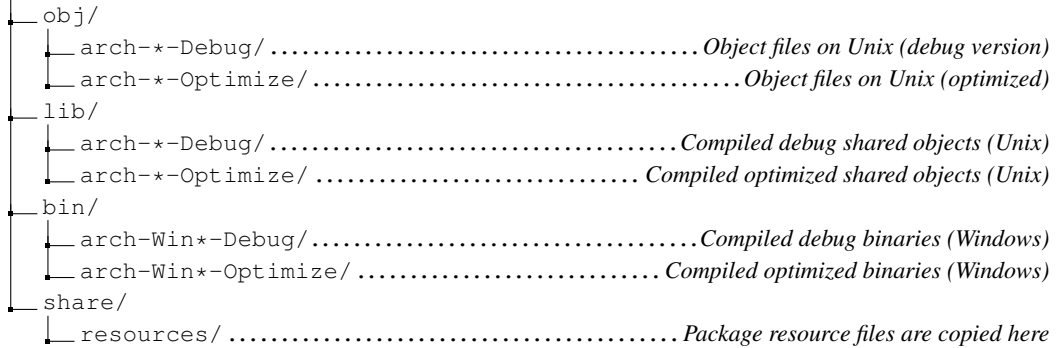
13.1.4 Quick Start Tutorial

This section contains a short tutorial on how to compile and execute the example package provided with Avizo XPand Extension. The example package contains the source code of the example modules and IO routines described elsewhere in this manual. At this point, you should just get a rough idea about the basic process required to develop your own modules and IO routines. Details will be discussed in the following sections.

For the development of custom Avizo packages, a dedicated directory, the local Avizo directory, is required. Initially, this directory should be created using the *Development Wizard*. Lets see how this is done:

- Start Avizo and choose the Development Wizard from the *XPand* menu of the Avizo main window.
- Make sure that the item *Set local Avizo directory* is selected in the wizard's dialog window.
- Press the *Next* button.

You can now enter a directory name for the local Avizo directory. For example, you may choose `AvizoLocal` in your home directory. The directory must be different from the directory where Avizo has been installed.

- Enter the name for the local Avizo directory.
- Select the button *copy example package*.
- Press the *OK* button.

If the directory did not yet exist, Avizo asks you if it should be created. The name of the directory is stored in the Windows registry or in the `.AvizoRegistry` file in the Unix or MacOS home directory, so that the next time Avizo is started, all modules or IO routines defined in this directory will be available.

The next step is to create the Visual Studio project files for Windows or the GNUmakefiles for Unix and MacOS. These files will be generated from a *Package* file which must be present in each custom package directory. The syntax of the Package file is described in section 13.2.10. The example package already contains a Package file, so there is no need to create one here.

- Select *Create build system* on the main page of the Development Wizard.
- Press the *Next* button.
- Choose *all local packages* as target.
- Choose which kind of build system you want to create.
- Press the *OK* button.

The files for the selected build system will now be created automatically. The advantage of the automatic generation is that the include and library paths are always set correctly. Also, any dependencies between local packages are taken into account.

Once the build system has been created, you can close the Development Wizard and exit Avizo. We are now ready to compile the example package. Yet, as a reminder, you will not be able to compile in debug if you have not installed the "debug binaries" during Avizo installation. The compilation procedure is different for each platform:

Windows Visual Studio

- Start Visual Studio and load the solution file `AvizoLocal.vc120.sln` from the local Avizo directory. If your local Avizo directory is not called `AvizoLocal`, the solution file also has some other name.
- Build all local packages in debug mode by choosing *Build Solution* from the *Build* menu.

Unix GNUmakefile system

- Change into the local Avizo directory in a shell.
- Type in `gmake` to build all local packages in debug mode. If `gmake` is not already installed on your system, you can find it in the subdirectory `bin` in the Avizo root directory. Either add this directory in your path variable or create a link in a directory already listed in your path, e.g., `/usr/bin`.

Mac GNUmakefile system

- Open a `Terminal` window and change into the local Avizo directory you created before.
- Type in `gmake` to build all local packages.

Important note: please refer to *system requirements for Mac OSX* if an error message like `ld: framework not found CUDA` is displayed by the compiler.

We are now ready to start Avizo in order to test the example package. However, because we have compiled the example package in debug mode, we also need to start Avizo in debug mode. Otherwise, Avizo would not find the correct DLLs or shared libraries. For Linux, start Avizo with the command line option `-debug`. On Windows, use the *Avizo (debug)* shortcut in the Start menu. On Mac systems, you can start the application immediately, the new path will be automatically recognized.

In order to check if the example package has been successfully compiled and can be loaded by Avizo, you can, for example, choose the entry *Other / DynamicColormap* from the *Project >Create Object...* menu of the Avizo main window. Then a new colormap object should be created. You can find the source code of this new object in the local Avizo directory under `src/mypackage/MyDynamicColormap.cpp`. In the same directory, there is also the header

file for this class.

If you want to compile the example package in release mode, you must change the active configuration in Visual Studio and recompile the code. On Unix, you have to call `gmake MAKE_CFG=Optimize`. You can also define `MAKE_CFG` as an environment variable.

13.1.5 Compiling and Debugging

This section provides additional information not covered by the *quick start guide* on how to compile and debug custom Avizo packages. You may skip it the first time you are reading this manual. The information will not become relevant until you are actually developing your own code.

It has already been mentioned that the development of custom Avizo packages should take place in a local Avizo directory. Initially, such a directory should be created using the Development Wizard described in section 13.2. The name of the local Avizo directory is stored in the Windows registry or in the file `.AvizoRegistry` in your Unix home directory. On both Windows and Unix, the name of the local Avizo directory can be overridden by defining the environment variable `AVIZO_LOCAL`. This might be useful if you want to switch between different local Avizo directories. However, in general it is recommended not to set this variable.

For each local package, there is a resource file stored in the subdirectory `share/resources` in the local Avizo directory. This file contains information about all modules and IO routines provided by that package. A local package can be compiled in debug mode suitable for debugging or in release mode with compiler optimization turned on. In the first case, the DLLs or shared libraries are stored under `bin/arch-*-Debug` on Windows and `lib/arch-*-Debug` on Unix, provided you have installed "debug binaries" during Avizo installation. In the second case, they are stored under `bin/arch-*-Optimize` or `lib/arch-*-Optimize`. Here the '*' indicates the actual architecture name. In the following, it will be describe how to compile local packages in both modes on the different platforms and how to debug the code using a debugger.

13.1.5.1 Windows Visual Studio

Note: Mixing code generated with different versions of Visual Studio (Visual Studio 2008, 2010, 2012, etc.) is not officially supported. So, generally you need the same Visual Studio version with which Avizo was compiled (see *system requirements*). However, compiling your own modules using another version of Visual Studio *may* work if you install the corresponding Visual Studio runtime together with Avizo on any Windows PC that makes use of your custom modules. Be aware this is not a recommended usage.

The workspace and project files for Visual Studio are generated automatically from the Avizo Package files by the Development Wizard. There should be no need to change the project settings manually.

By default, Visual Studio will compile in debug mode. In order to generate optimized code, you need to change the active configuration. This is done by choosing *Configuration Manager...* from the *Build* menu. In the *Active Solution Configuration* pulldown menu, select *Release*.

In order to execute the debug mode version of your local packages, you must start Avizo with the debug `Avizo.exe` located in the `bin/arch-*-Debug` folder. For convenience, a link *Avizo (Debug)* is

provided in the start menu. However, if you want to debug your code, you need to start Avizo from Visual Studio. Thus, you need to specify the correct executable in the project properties dialog.

You can bring up the project properties dialog by choosing *Properties* from the *Project* menu. Select *Debugging* from the left pane. In the field *Command*, choose the file `bin/arch-<version>-Debug/Avizo.exe` located in the Avizo root directory (see Figure 13.1). Replace `<version>` with the version of Avizo you have (see *system requirements*).

Figure 13.1: Specifying the name of the executable in Visual Studio.

You can now start Avizo from Visual Studio by pressing F5 or by choosing *Start* from the *Debug* menu. In order to debug your code, you may set breakpoints at arbitrary locations in your code. For example, if you want to debug a read routine, set a breakpoint at the beginning of the routine, execute Avizo and invoke the read routine by loading some file.

13.1.5.2 Linux

In order to compile a local package under Linux, you need to change into the package directory and execute `gmake` in a shell. The `gmake` utility is provided in the `bin` subdirectory of the Avizo root directory. Either add this directory to your path or create a link in a directory already listed in your path, e.g., in `/usr/bin`.

The required GNUmakefiles will be generated automatically from the Avizo Package files by the Development Wizard. There should be no need to edit the GNUmakefiles manually. Depending on the contents of the Package file, all source files in a package directory will be compiled, or a subset only. By default, all files will be compiled. The Development Wizard will put the name of the Avizo root directory into the file `GNUmakefile.setroot`. You may overwrite the name by defining the environment variable `AVIZO_ROOT`. For example, this might be useful when working simultaneously with two different Avizo versions.

By default, `gmake` will compile debug code. In order to compile optimized code, invoke `gmake MAKE_CFG=Optimize`. Alternatively, you may set the environment variable `MAKE_CFG` to `Optimize`.

If you have a multi-processor machine, you may compile multiple files at once by invoking `gmake` with the option `-j<n>`. Here `<n>` denotes the number of compile jobs to be run in parallel. Usually twice the number of processors is a good choice.

If you have compiled debug code, you must invoke Avizo with the command line option `-debug`. Otherwise, the optimized version will be executed. If no such version exists, an error occurs. Instead of specifying `-debug` at the command line, you may also set the environment variable `MAKE_CFG` to `Debug`.

In order to run Avizo in a debugger, invoke the Avizo start script with the -gdb or -ddd command line option.

Note that usually you cannot set a breakpoint in a local package right after starting the debugger. This is because the package's shared object file will not be linked to Avizo until the first module or read or write routine defined in the package is invoked. In order to force the shared object to be loaded without invoking any module or read or write routine, you may use the command `dso open lib<name>.so`, where `<name>` denotes the name of the local package. Once the shared object has been successfully loaded, breakpoints may be set. It depends on the debugger whether these breakpoints are still active when the program is started the next time.

13.1.6 Maintaining Existing Code

This section is directed to programmers who have already developed custom modules using a previous version of Avizo XPand Extension. In particular, we describe

- *how to upgrade to Avizo XPand Extension 9*, and
- *how to rename an existing package.*

13.1.6.1 Upgrading to Latest Version of Avizo XPand Extension

Avizo is an evolving interactive software product. The Avizo XPand Extension API is subject to change. While we do our best to maintain compatibility, some incompatible changes may be intro-

duced and require adaptation of your existing code, such as code relating to modules and the user interface.

The *XPand Reference Manual* contains all classes and methods that you can safely use since the source compatibility will be ensured between versions. We do our best to maintain compatibility between versions, however, to improve our API it is sometimes necessary to break compatibility. We do not guarantee any compatibility if you use internal classes or methods, and if used, you will get a compatibility warning at the compilation step. A Porting Guide tab that indicates the compatibility breaks in Avizo 9.4 is accessible in the *Avizo Programmer's Reference* document (`$AVIZO_ROOT/share/devrefAvizo/Avizo.chm`).

Moreover, you may use headers and libraries for Open Inventor and Qt provided by the Avizo XPand Extension. Those APIs follow themselves via their own path and may introduce specific incompatibilities. Please refer to their respective release notes for more details.

13.1.6.2 Renaming an Existing Package

Sometimes you may want to rename an existing Avizo package, for example when using an existing package as a template for a new custom package. In order to do so, the following changes are required:

- Rename the package directory:
  `AvizoLocal/src/oldname` becomes `AvizoLocal/src/newname`
- Rename the following files in the package directory:
  `oldnameAPI.h` becomes `newnameAPI.h`
  `share/resources/oldname.rc` becomes `share/resources/newname.rc`
- In the package resource file `share/resources/newname.rc` and in the `Package` file, replace `oldname` by `newname`.
- In the file `newnameAPI.h`, replace `OLDNAME_API` by `NEWNAME_API`.
- In all header and source files of the package, adjust the included directives, if necessary, i.e., instead of
  `#include <oldname/SomeClass.h>`
  now write `#include <newname/SomeClass.h>`

All replacements can be performed using an arbitrary text editor. After all files have been modified as necessary, a new Visual Studio project file or a new GNUmakefile should be created using the Avizo development wizard.

13.2 The Development Wizard

The development wizard is a special tool which helps you to set up a local Avizo directory tree so that you can write custom extensions for Avizo. In addition, the development wizard can be used to create templates of Avizo modules or read or write routines. The details of developing such components are treated in other sections. At this point, we want to give a short overview about the functionality of the development wizard.
In particular, we discuss

- *how to invoke the development wizard*
- *how to set or create the local Avizo directory*
- *how to add a package to the local Avizo directory*
- *how to add components to an existing package*
- *how to create the files for the build system*

Finally, a section describing the *Package file syntax* is provided.

13.2.1 Starting the Development Wizard

In order to invoke the development wizard, first start Avizo. Then select *Development Wizard* from the main window's *XPand* menu. Note that this menu option will only be available if you are running the developer version of Avizo (with Avizo XPand Extension installed).

The layout of the development wizard is shown in Figure 13.2. Initially, the wizard informs you about the local Avizo directory currently being used. If no local Avizo directory is defined, this is indicated too. Furthermore, the wizard lets you select between four different tasks to be performed. These are

- *setting the local Avizo directory (or creating a new one)*
- *adding a new package to the local Avizo directory*
- *adding a component to an existing package*
- *creating the files for the build system*

The first option is always available. A new package can only be added if a valid local Avizo directory has been specified. For the local Avizo directory to be valid, among others, it must contain a subdirectory called src. If at least one package exists in the src directory of the local Avizo directory, a new component, i.e., a module or a read or write routine, can be added to a package. Finally, the last option allows you to create all files required by the build system, i.e., Visual Studio project files or GNUmakefiles for Unix platforms.

13.2.2 Setting Up the Local Avizo Directory

The local Avizo directory contains your source code and the binaries of all your custom extensions. You can easily specify the name of this directory using the development wizard (see Figure 13.3). Since all users can write his/her own extensions for Avizo, it is recommended that you create the local Avizo directory in your home directory.

If the specified directory does not exist, the development wizard asks you whether it should be created. If you confirm, the directory and its subdirectories will be created. You may also specify an existing empty directory in the text field. Then the subdirectories will be created in that location.

Finally, you may choose the existing directory previously created by the development wizard. In this case, a simple check validates the specified directory.

To unset the local Avizo directory, clear the text field and click **OK**. The directory will not be deleted, but the next time Avizo is started, modules and IO routines defined in the local Avizo directory will
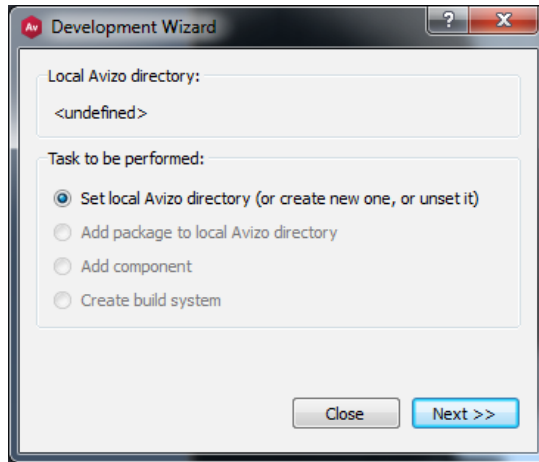
Figure 13.2: Initial Layout of the Avizo Development Wizard.
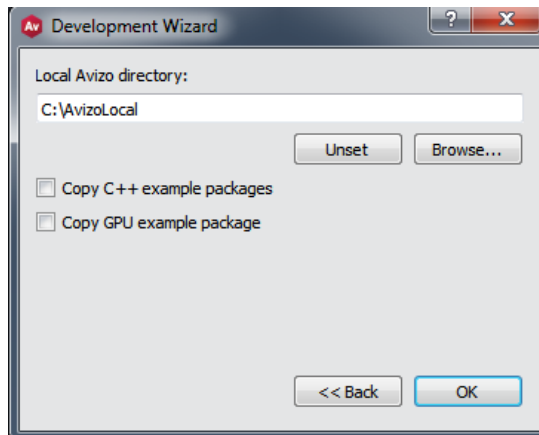
Figure 13.3: Setting the Local Avizo Directory.

not be available anymore.

Once you have set up the local Avizo directory, the name of the directory is stored permanently. This means that the next time Avizo is started the .rc-files located in the subdirectory `share/resources` of the local Avizo directory can be read. In this way, custom components are made known to Avizo. On Windows, the name of the local Avizo directory is stored in the Windows registry. On Unix systems, it is stored in the file `.AvizoRegistry` in your home directory. In both cases, these settings can be overridden by defining the environment variable `AVIZO_LOCAL`.

The development wizard provides a toggle for copying an example package to the local Avizo directory. You will get a warning if this button is activated and an existing local Avizo directory already containing the example package has been specified. The example package is copied to the subdirectory `src/mypackage` in the local Avizo directory. It contains all read and write routines plus modules presented as examples in this guide.

You can select toggle *Copy GPU example package* to provide another example package. To build this package, install the CUDA toolkit first. This example is copied in the local Avizo directory in the subdirectorie `src/gaussianfiltercudac`.

13.2.3 Adding a New Package

All Avizo components are organized in *packages*. Each package will be compiled into a separate shared object (or DLL file on Windows). Therefore, before any components can be defined, at least one package must be created in the local Avizo directory. In order to do so, choose *add package to local Avizo directory* on the first page of the wizard and press *Next*. On the next page you can enter the name of the new package (see Figure 13.4).
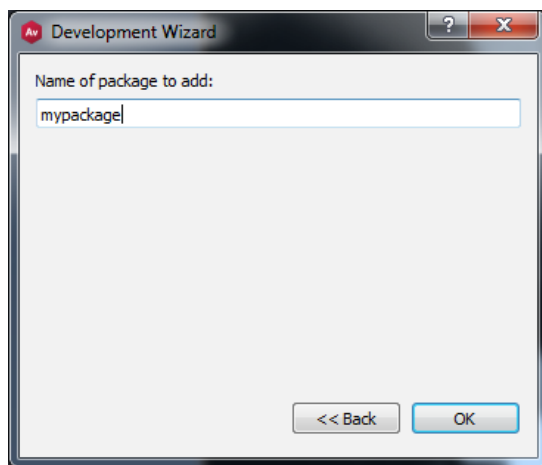
Figure 13.4: Adding a new package to the local Avizo directory.

The name of a package must not contain any white spaces or punctuation characters. When a package is added, a subdirectory of the same name is created under `src` in the local Avizo directory. In this directory, the source code and header files of all the modules and IO routines of the package are stored. In addition, in each package directory, there must be a *Package* file from which the build system files can be generated.

Initially, a default *Package* file will be copied into a new package directory. This default file adds the most common Avizo libraries for linking. It also selects all C++ source files in the package directory

to be compiled. In order to generate the build system from the *Package* file, please refer to subsection 13.2.9.

In addition to the *Package* file, the file `version.cpp` will also be copied into a new package directory. This file allows you to put version information into your package, which can later be viewed in the Avizo system information dialog. Finally, an empty file `package.rc` will also be copied into `share/resources`. Later modules and IO routines will be registered in this file.

13.2.4 Adding a New Component

If you choose the *add component* option on the first page of the development wizard, you will be asked what kind of component should be added to which package. Remember that the *add component* option will only be available if a valid local Avizo directory with at least one existing package is found. In particular, templates

- of an *ordinary module*,
- of a *compute module*,
- of a *read routine*,
- or of a *write routine*

may be created (see Figure 13.5). The option menu in the lower part of the dialog box lets you specify the package to which the component should be added. After you press the *Next* button, you will be asked to enter more specific information about the particular component you want to add. Up to this point no real operation has been performed, i.e., no files have been created or modified.
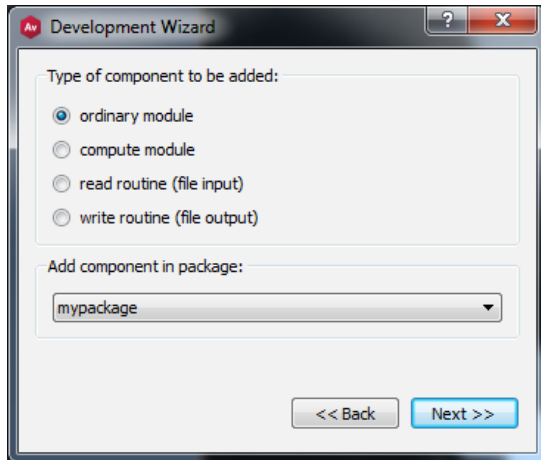
Figure 13.5: Adding a new component to an existing package.
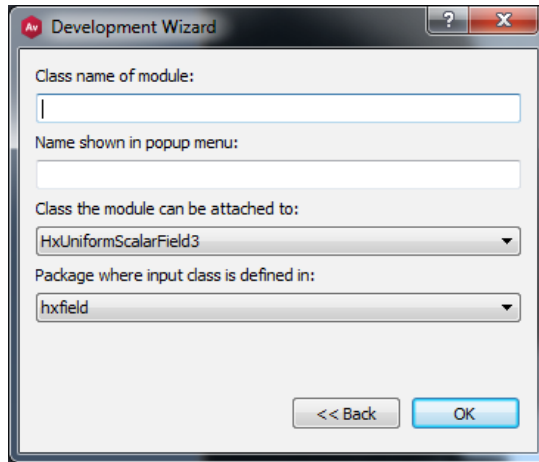
Figure 13.6: Creating the Template of a Custom Module

13.2.5   Adding an Ordinary Module

An ordinary module in Avizo directly visualizes the data object to which it is attached. Examples include the *Isosurface* module, the *Volume Render* module, and the *Surface View* module. Such modules, sometimes also called *display modules*, are represented by yellow icons in the Project View.

To create the template for an ordinary module using the development wizard, you must enter the C++ class name of the module, the name to be shown in the popup menu of possible input data objects, the C++ class name of possible input data objects, and the package where the input class is defined (see Figure 13.6).

Once you click **OK**, two files are created in the package directory: a header file and a source code file for the new module. In addition, a new module statement is added to the package resource file located under `share/resources` in the package directory. After you have added a new module to a package, you need to recreate the build system files before you can compile the module. Details are described in subsection 13.2.9.

13.2.6   Adding a Compute Module

A *compute module* in Avizo usually takes one or more input data objects, performs some kind of computation, and puts back a resulting data object in the Project View. Compute modules are represented by red icons in the Project View.
The only difference between an ordinary module and a compute module is that the former is directly derived from `HxModule` while the latter is derived from `HxCompModule`. When creating a template for a compute module using the development wizard, the same input fields must be filled in as for an ordinary module. The meaning of these input fields is described in subsection 13.2.5.

13.2.7 Adding a Read Routine

As will be explained in more detail in subsection 13.3.2, read routines are global C++ functions used to create one or more Avizo data objects from the contents of a file stored in a certain file format. To create the template of a new read routine, first the name of the routine must be specified (see Figure 13.7). The name must be a valid C++ name. It must not contain blanks or any other special characters.
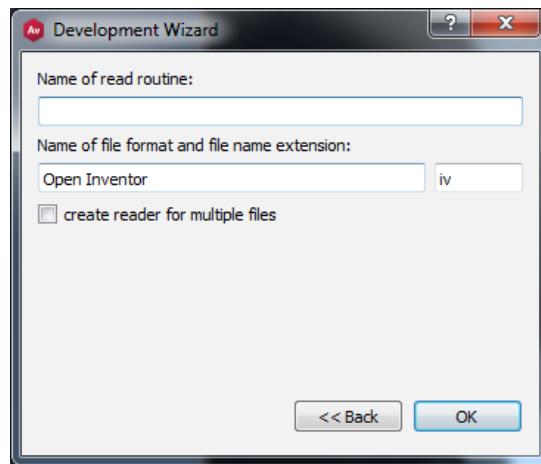
Figure 13.7: Creating the template of a read routine.

Moreover, the name of the file format and the preferred file name extension must be specified. The extension will be used by the file browser in order to identify the file format. The format name will be displayed next to any matching file.

Finally, a toggle can be set in order to create the template of a read routine supporting the input of multiple files. Such a routine will have a slightly different signature. It allows you to create a single data object from multiple input files. For example, multiple 2D image files can be combined in a single 3D image volume. Details are provided in subsection 13.3.2.3.

After you press *OK* a new file `<name>.cpp` will be created in the package directory, where `<name>` denotes the name of the read routine. In addition, the read routine will be registered in the package resource file. Some file formats can be identified by a unique file header, not just by the file name extension. In such a case, you may want to modify the resource file entry as described in subsection 13.3.2.1.

Remember, that after you have added a new read routine to a package, you need to recreate the build system files before you can compile it. Details are described in subsection 13.2.9.

13.2.8 Adding a Write Routine

A write routine is a global C++ function which takes a pointer to some data object and writes the data to a file in a certain file format. The details are explained in subsection 13.3.3. In order to create the

template of a new write routine, the name of the routine must first be specified (see Figure 13.8). The name must be a valid C++ name. It must not contain blanks or any other special characters.
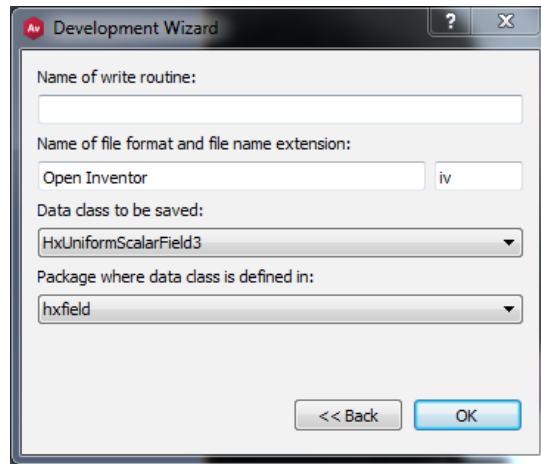
Figure 13.8: Creating the template of a write routine.

In addition, the name of the file format and the preferred file name extension must be specified. Before saving a data object, both the name and the extension will be displayed in the file format menu of the Avizo file browser.

Finally, the C++ class name of the data object to be saved must be chosen as well as the package in which this class is defined. Some important data objects such as a `HxUniformScalarField3` or a `HxSurface` are already listed in the corresponding combo box. However, any other class, including custom classes, may be specified here. Instead of the name of a data class, even the name of an interface class such as `HxLattice3` may be used (see subsection 13.5.2.1).

After you press *OK*, a new file `<name>.cpp` will be created in the package directory, where `<name>` denotes the name of the write routine. In addition, the write routine will be registered in the package resource file.

Remember, that after you have added a new write routine to a package, you need to recreate the build system files before you can compile it. Details are described in subsection 13.2.9.

13.2.9 Creating the Build System Files

Before you can actually compile your own packages, you need to create project files for Microsoft Visual Studio on Windows or GNUmakefiles for Unix. These files contain information such as the source code files to be compiled, or the correct include and library paths. Since it is not trivial to set up and edit these files manually, Avizo provides a mechanism to create them automatically. In order to do this, a so-called *Package* file must exist in each package. The Package files contain the name of the package and a list of dependent packages. It may also contain additional tags to customize the build

process. The syntax of the package file is described in subsection 13.2.10. However, usually there is no need to modify the default Package file created by the development wizard.

While the automatic generation of the build system files is a very helpful feature, it also means that you should not modify the resulting project or GNUmakefiles manually, because they might be easily overwritten by Avizo.
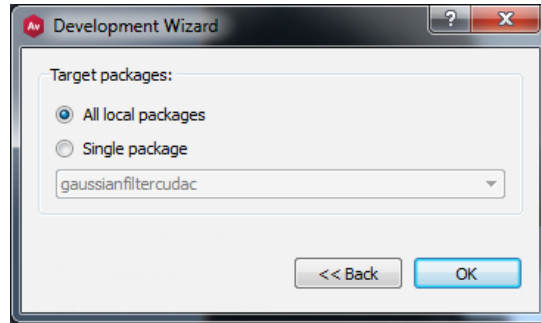
Figure 13.9: Creating the build system files.

If you select *Create build system* on the main page of the development wizard and then press the *Next* button, the controls shown in Figure 13.9 will be activated. You can choose if you want to create the build system files for all local packages or just for a particular one.

13.2.10   The Package File Syntax

The Package file contains information about a local package. From this file, Visual Studio project files or GNUmakefiles can be generated. The Package file is a Tcl file. It defines a set of Tcl variables indicating things like the name of the package, dependent libraries, or additional files to be copied when building the package. The default Package file created by the Development Wizard looks as follows:

```
set PACKAGE {mypackage}

set LIBS {
    hxplot hxtime hxsurface hxcolor hxfield
    hxcore amiramesh mclib oiv tcl qt
}

set SHARE {
    share/resources/mypackage.rc
}
```

In most cases, the default file works well and do not need to be modified. However, in order to accomplish special tasks, the default values of the variables can be changed or additional variables can be defined. Here is a detailed list describing the meaning of the different variables:

PACKAGE

The variable PACKAGE indicates the name of the package. This should be the same as the name of the package directory. The package name must not contain any characters other than letters or digits.

LIBS

Lists all libraries on which the package depends. By default, the most common Avizo packages are inserted here. You can modify this list as needed. For example, if you want to link against a library called *foo.lib* on Windows or *libfoo.so* on Unix, you should add *foo* to LIBS.

In addition to a real library name, you may use the following aliases in the LIBS variable:

oiv - for the Open Inventor libraries
tcl - for the Tcl library
opengl - for the OpenGL library
qt - for the Qt library

If you want to link against a library only on a particular platform, you can set a dedicated variable LIBS-arch, where arch denotes the platform. You may further distinguish between the debug and release version of the code. Here is an example:

```
set LIBS {mclib amiramesh schedule hxz qt oiv opengl tcl}
set LIBS-Unix {hxgfxinit}
set LIBS-Win {hxgfxinit}
set LIBS-Win-Debug {msvcrtd mpr}
set LIBS-Win-Optimize {msvcrt mpr}
```

SHARE

Lists all files which should be copied from the package directory into the local Avizo directory. By default, only the package resource file will be copied. However, you may add additional files here, if necessary. Instead of explicit file names, you may use wildcards. These will be resolved using the standard Tcl command glob. For example, if you have some demo scripts in your package, you could copy them in the following way:

```
set SHARE {
    share/resources/mypackage.rc
    share/demo/mydemos/*.hx
}
```

As for the LIBS variable, you may append an arch string here, i.e., SHARE-arch. The files then will only be copied on the specified platforms.

INCLUDES

This variable may contain a list of additional include paths. These paths are used by the compiler to locate header files. By default, the include path is set to $AVIZO_ROOT/include, $AVIZO_ROOT/include/oiv, $AVIZO_LOCAL/src, and the local package directory.

COPY

This may contain a list of files which are copied from a location other than the local package directory. You need to specify the name of the target file followed by the name of the destination file relative to the local Avizo directory. For example, you may want to copy certain data files from some archive into the Avizo directory. This can be achieved in the following way.

```
set COPY {
    D:/depot/data/28523763.dcm data/test
    D:/depot/data/28578320.dcm data/test
    D:/depot/data/28590591.dcm data/test
}
```

As for the `LIBS` variable, you may append an arch string here, i.e., `COPY-arch`. The files then will only be copied on the specified platforms. A common application is to copy external libraries required on a particular platform into the Avizo directory.

SRC

This variable specifies the source code files to be compiled for the package. The default value of this variable is

```
set SRC {*.cpp *.c}
```

This means, that by default, all .cpp and .c files in the local package directory will be compiled. Sometimes you may want to replace this default by an explicit list of source files.
Again, you may append an arch string to the `SRC` variable, so that certain files will only be compiled on a particular platform.

INCSRC

This variable specifies the header files to be included into the Visual Studio package project file. The default value of this variable is

```
set INCSRC {*.h *.hpp}
```

This means that, by default, all .h and .hpp files in the local package directory will be considered.
Again, you may append an arch string to the `INCSRC` variable, so that certain header files will only be considered on a particular platform.

13.3   File I/O

This section describes how user-defined read and write routines can be added to Avizo. The purpose of custom read and write routines is to add support for file formats not available in Avizo.
First, some general hints *on file formats* are given. Then we discuss how *read routines* are expected to look in Avizo. *Write routines* are treated subsequently. Finally, the *AmiraMesh API* is discussed. Using this API, file I/O for new custom data objects can be implemented rather easily.

13.3.1 On file formats

Before going into detail, let us clarify some general concepts. In Avizo, all data loaded into the system are encapsulated in C++ *data classes*. Section 13.5 provides more information about the standard data classes. For example, there is a class to represent tetrahedral grids (*HxTetraGrid*), a separate one for scalar fields defined on tetrahedral grids (*HxTetraScalarField3*), and another one for 3D image data (*HxUniformScalarField3*). Every instance of a data class is represented by a green icon in the Avizo Project View.

The way in which data are stored in a disk file is called a file format. Although there is a relationship between data classes and file formats, these are two different things. It is especially important to understand that there is no one-to-one correspondence between them.

Typically, a specific data class (like 3D image data) can be stored in many different file formats (3D TIFF, DICOM, a set of 2D JPEG files, and so on). On the other hand, a specific file format does not necessarily correspond to exactly one data class. For example, a simple data file in Fluent UNS format can contain hexahedral grids (*HxHexaGrid*) or tetrahedral grids (*HxTetraGrid*). A more complicated Fluent UNS case containing both tetrahedra, hexahedra, pyramids and wedges can be read in Avizo XWind Extension and the grid will be stored as an unstructured model (*HxUnstructuredModel*).

Note that there is also no one-to-one correspondence between the instance of a data class (a green icon in Avizo) and the instance of a file format (the actual file). Often multiple files correspond to a single data object, for example, 2D images forming a single 3D image volume. On the other hand, a single file can contain the data of multiple data objects. For example, an AVS UCD file can contain a tetrahedral grid as well as multiple scalar fields defined on it.

Finally, note that information may get lost when saving a data object to a file in a specific format. For example, when saving a 3D image volume to a set of 2D JPEG images, the bounding box information will be lost. Likewise, there are user-defined parameters or attributes in Avizo that cannot be encoded in most standard file formats. On the other hand, a file reader often does not interpret all information provided by a specific file format.

13.3.2 Read Routines

As already mentioned in the previous section, a read routine is a C++ function that reads a disk file, interprets the data, creates an instance of an Avizo data class, and fills that instance with the data read from the file.

In order to write a read routine, obviously two things are needed, namely a specification of the file format to be read as well as the information for which of Avizo's data classes is able to represent the data and how this class is used. More information about the standard Avizo data classes is given in section 13.5. The C++ interface of these classes is described in the online reference documentation.

A read routine may either be a static member function of a class or a global function. In addition to the function itself, an entry in the package resource file is needed. In this way Avizo is informed about the existence of the read routine and about the type of files that can be handled by the reader.

In the following discussion, the implementation of a user-defined read routine will be illustrated by two concrete examples, namely *a simple read routine for scalar fields* and *a read routine for surfaces and surface fields*. Some *more details about read routines* will be discussed subsequently.

13.3.2.1 A Reader for Scalar Fields

In this section, we present a simple read routine designed for reading image volumes, i.e., 3D scalar fields, from a very simple file format, which we have invented for this example. The file format is called PPM3D (because it is similar to the ppm 2D image format). The PPM3D format will be an ASCII file format containing a header, three integer numbers specifying the size of the 3D image volume, and the pixel data as integer numbers in the range 0 to 255. An example file could look like this:

```
# PPM3D
4 4 3
43 44 213 9 23 234 3 3 3 44 213 9 23 234 36 63
44 213 9 23 234 35 3 5 44 213 9 23 234 31 13 12
44 213 9 23 234 35 3 5 44 213 9 23 234 31 13 12
```

The full source code of the read routine is contained in the example package provided with Avizo XPand Extension. In order to follow the example below, first create a local Avizo directory using the *Development Wizard*. Be sure that the toggle *copy example package* is activated, as described in subsection 13.2.2. The read routine can then be found in the local Avizo directory under `src/mypackage/readppm3d.cpp`.

Let us first take a look at the commented source code of the reader. Some general remarks follow below.

```
////////////////////////////////////////////////////////////////////
//
// Read routine for the PPM3D file format
//
////////////////////////////////////////////////////////////////////

#include "api.h"                          // storage-class specification
#include <hxcore/HxMessage.h>             // for output in console
#include <hxfield/HxUniformScalarField3.h> // class representing 3D images

MYPACKAGE_API int
readppm3d(const QString& filename)
{
    FILE* f = fopen(filename.toLocal8Bit(), "r"); // open the file

    if (!f)
    {
        theMsg->ioError(filename);
        return 0; // indicate error
    }

    // Skip header (first line). We could do some checking here:
    char buf[80];
    fgets(buf, 80, f);

    // Read size of volume:
    int dims[3];
    dims[0] = dims[1] = dims[2] = 0;
    fscanf(f, "%d %d %d", &dims[0], &dims[1], &dims[2]);

    // Do some consistency checking.
    if (dims[0] * dims[1] * dims[2] <= 0)
    {
        theMsg->printf(QString("Error in file %1.").arg(filename));
        fclose(f);
        return 0;
    }

    // Now create 3D image data object. The constructor takes the
    // dimensions and the primary data type. In this case we create
    // a field containing unsigned bytes (8 bit).
    HxUniformScalarField3* field =
        new HxUniformScalarField3(dims, McPrimType::MC_UINT8);

    // The HxUniformScalarField3 stores its data in a member variable
    // called lattice. We know, that the data is unsigned 8 bit,
    // because we specified this in the constructor.
    unsigned char* data = (unsigned char*)field->lattice().dataPtr();

    // Now we have to read dims[0]*dims[1]*dims[2] data values
    for (int i = 0; i < dims[0] * dims[1] * dims[2]; i++)
    {
        int val = 0;
```

```
        fscanf(f, "%d", &val);
        data[i] = (unsigned char)val;
    }

    // We are done with reading, close the file.
    fclose(f);

    // Register the data object to make it visible in the object pool. The
    // name for the new object is automatically generated from the filename.
    HxData::registerData(field, filename);

    return 1; // indicate success
}
```

The source file starts with some includes.

First, the file *api.h* is included. This file provides import and export storage-class specifiers for Windows systems. These are encoded in the macro `MYPACKAGE_API`. On Unix systems this macro is empty and can be omitted.

Next, the file *HxMessage.h* is included. This header file provides the global pointer *theMsg* which allows us to print out text messages in the Avizo console window. In our read routine, we use *theMsg* to print out error messages if a read error occurred.

Finally, the header file containing the declaration of the data class to be created is included, i.e., *HxUniformScalarField3.h*. As a general rule, every class in Avizo is declared in a separate header file. The name of the header file is identical to the name of the C++ class.

The read routine itself takes one argument, the name of the data file to be read. It should return 1 on success, or 0 if an error occurred and no data object could be created. The body of the read routine is rather straightforward. The file is opened for reading. The size of the image volume is read. A new data object of type *HxUniformScalarField3* is created and the rest of the data is written into the data object. Finally, the file is closed again and the data object is put into the Project View by calling `HxData::registerData`. In principle, all read routines look like this example. Of course, the type of data object being created and the way that this object is initialized may differ.

In order to make the new read routine known to Avizo, an entry must be added to the package resource file, i.e., to the file `mypackage/share/resources/mypackage.rc`. In our case, this entry looks as follows:

```
dataFile -name "PPM3D Demo Format"  \
    -header "PPM3D"                  \
    -load "readppm3d"                \
    -package "mypackage"
```

The `dataFile` command registers a new file format called *PPM3D Demo Format*. The option `-header` specifies a regular expression which is used for automatic file format detection. If the first 64 bytes of a file match this expression, the file will be automatically loaded using this read routine. Of course, some data formats do not have a unique file header. In this case, the format may also be detected from a standard file name extension. Such an extension may be specified using the `-ext` option of the `dataFile` command. Multiple extensions can be specified as a comma-separated list.

The actual C++ name of the read routine is specified via `-load`. Finally, the package containing the read routine must be specified using the `-package` option.

If you have compiled the example in the mypackage example package, you can try to load the example file `mypackage/data/test.ppm3d`. As you will see, the file browser automatically detects the file format and displays *PPM3D Demo Format* in its file list.

13.3.2.2   A Reader for Surfaces and Surface Fields

Now that you know what a read routine looks like in principle, let us consider a more complex example. In this section, we discuss a read routine which creates more than one data object. In particular, we want to read a triangular surface mesh from a file. In addition to the surface description, the file may also contain data values for each vertex of the surface. Data defined on a surface mesh are represented by separate classes in Avizo. Therefore, the reader must first create a data object representing the surface only. Then appropriate data objects must be created for each surface field.

Again, the file format is quite simple and has been invented for the purpose of this example. We call it the *Trimesh* format. It is a simple ASCII format without any header. The first line contains the number of points and the number of triangles. Then the x-, y-, and z-coordinates of the points are listed. This section is followed by triangle specifications consisting of three point indices for each triangle, point count starts at one. The next section is for vertex data, starting with a line that contains an arbitrary number of integers. Each integer indicates that there is a data field with a certain number of variables defined on the surface's vertices, e.g., 1 for a scalar field or 3 for a vector field. The data values for each vertex follow in separate lines. Here is a small example containing a single scalar surface field:

```
4 2
0.0 0.0 0.0
1.0 0.0 0.0
0.0 1.0 0.0
1.0 1.0 0.0
1 2 4
1 4 3
1
0.0
0.0
1.0
1.0
```

You can find the full source code of the reader in the local Avizo directory under `src/mypackage/readtrimesh.cpp`. Remember that the example package must have been copied into the local Avizo directory before compiling. For details, refer to subsection 13.2.2. Let us now look at the complete read routine before discussing the details:

```
/////////////////////////////////////////////////////////////////
//
// Read routine for the trimesh file format
//
/////////////////////////////////////////////////////////////////

#include "api.h"
```

```
#include <hxcore/HxMessage.h>
#include <hxqt/QxStringUtils.h>
#include <hxsurface/HxSurface.h>
#include <hxsurface/HxSurfaceField.h>

MYPACKAGE_API int
readtrimesh(const QString& filename)
{
    FILE* fp = fopen(filename.toLocal8Bit(), "r");

    if (!fp)
    {
        theMsg->ioError(filename);
        return 0;
    }

    // 1. Read the surface itself

    char buffer[256];
    fgets(buffer, 256, fp); // read first line

    int i, j, k, nPoints = 0, nTriangles = 0;
    sscanf(buffer, "%d %d", &nPoints, &nTriangles); // get numbers

    if (nPoints < 0 || nTriangles < 0)
    {
        theMsg->printf("Illegal number of points or triangles.");
        fclose(fp);
        return 0;
    }

    HxSurface* surface = HxSurface::createInstance(); // create new surface
    surface->addMaterial("Inside", 0);                // add some materials
    surface->addMaterial("Outside", 1);

    HxSurface::Patch* patch = new HxSurface::Patch; // create surface patch
    surface->patches.append(patch);                  // add patch to surface
    patch->innerRegion = 0;
    patch->outerRegion = 1;

    surface->points().resize(nPoints);
    surface->triangles().resize(nTriangles);

    for (i = 0; i < nPoints; i++)
    { // read point coordinates
        McVec3f& p = surface->points()[i];
        fgets(buffer, 256, fp);
        sscanf(buffer, "%g %g %g", &p[0], &p[1], &p[2]);
    }

    for (i = 0; i < nTriangles; i++)
    { // read triangles
        int idx[3];
```

```
        fgets(buffer, 256, fp);
        sscanf(buffer, "%d %d %d", &idx[0], &idx[1], &idx[2]);

        Surface::Triangle& tri = surface->triangles()[i];
        tri.points[0] = idx[0] - 1; // indices should start at zero
        tri.points[1] = idx[1] - 1;
        tri.points[2] = idx[2] - 1;
        tri.patch = 0;
    }

    patch->triangles.resize(nTriangles);
    for (i = 0; i < nTriangles; i++)
        patch->triangles[i] = i; // add all triangles to one patch

    HxData::registerData(surface, filename); // add surface to object pool

    // 2. Check if file also contains data fields

    fgets(buffer, 256, fp);

    QStringList stringList = toQString(buffer).split(' ');
    McDArray<HxSurfaceField*> fields;
    foreach (QString string, stringList)
    { // are there any numbers here ?
        bool intValid = false;
        int n = string.toInt(&intValid);
        if (intValid)
        {
            // Create appropriate field, e.g. HxSurfaceScalarField if n==1
            HxSurfaceField* field =
                HxSurfaceField::create(surface,
                                       HxSurfaceField::ON_NODES,
                                       n);
            fields.append(field);
        }
    }

    if (fields.size())
    {
        for (i = 0; i < nPoints; i++)
        { // read data values of all fields
            fgets(buffer, 256, fp);
            QStringList stringList = toQString(buffer).split(' ');
            int indice = 0;
            for (j = 0; j < fields.size(); j++)
            {
                int n = fields[j]->nDataVar();
                float* v = &fields[j]->dataPtr()[i * n];
                for (k = 0; k < n; k++)
                {
                    if (indice < stringList.size())
                    {
                        v[k] = stringList[indice].toFloat();
```

```
                indice++;
            }
        }
    }
}

for (i = 0; i < fields.size(); i++)
{ // add fields to object pool
    HxData::registerData(fields[i], QString());
    fields[i]->composeLabel(surface->getLabel(), "data");
}
}

fclose(fp); // close file and return ok

// Fix the load command of all created objects
QString loadCmd = QString(
                    "set TMPIO [load -trimesh \"%1\"]\n"
                    "lindex $TMPIO 0")
                    .arg(filename);
surface->setLoadCmd(loadCmd, 1);

for (i = 0; i < fields.size(); i++)
{
    loadCmd = QString("lindex $TMPIO %1").arg(i + 1);
    fields[i]->setLoadCmd(loadCmd, 1);
}

return 1;
}
```

The first part of the read routine is very similar to the PPM3D reader outlined in the previous section. Required header files are included, the file is opened, the number of points and triangles are read, and a consistency check is performed.

Then an Avizo surface object of type *HxSurface* is created. The class *HxSurface* has been devised to represent an arbitrary set of triangles. The triangles are organized into *patches*. A patch can be thought of as the boundary between two volumetric regions, an "inner" and an "outer" region. Therefore, for each patch, an inner region and an outer region should be defined. In our case, all triangles will be inserted into a single patch. After this patch has been created and initialized, the number of points and triangles is set, i.e., the dynamic arrays *points* and *triangles* are resized appropriately.

Next, the point coordinates and the triangles are read. Each triangle is defined by the three points of which it consists. The point indices start at one in the file but should start at zero in the *HxSurface* class. Therefore, all indices are decremented by one. Once all triangles have been read, they are inserted into the patch we have created before. The surface is now fully initialized and can be added to the Project View by calling *HxData::registerData*.

The second part of the read routine is reading the data values. First, we check how many data fields are defined and how many data variables each field has.

For each group of data variables, a corresponding surface field is created. The fields are temporarily stored in the dynamic array *fields*. Instead of directly calling the constructor of the class *HxSur-*

faceField, the static method *HxSurfaceField::create* is used. This method checks the number of data variables and automatically creates an instance of a subclass such as *HxSurfaceScalarField* or *Hx-SurfaceVectorField*, if this is possible. In principle, surface fields may store data on a per-node or a per-triangle basis. Here we are dealing with vertex data, so we specify the encoding to be *HxSurface-Field::ON_NODES* in *HxSurfaceField::create*.

Finally, the data values are read into the surface fields created before. Afterwards, all the fields are added to the Project View by calling *HxData::registerData* again. In order to define a useful name for the surface fields, we call the method *composeLabel*. This method takes a reference name, In this case, the name of the surface, and replaces the suffix by some other string, in this case "data". Avizo automatically modifies the name so that it is unique. Therefore, we can perform the same replacement for all surface fields.

Like any other read routine, our *Trimesh* reader must be registered in the package resource file before it can be used. This is done by the following statement in `mypackage/share/resources/mypackage.rc`:

```
dataFile -name "Trimesh Demo Format"    \
    -ext "trimesh,tm"                   \
    -load "readtrimesh"                 \
    -package "mypackage"
```

Most of the options of the `dataFile` command have already been explained in the previous section. However, in contrast to the PPM3D format, the *Trimesh* format cannot be identified by its file header. Therefore, we use the `-ext` option to tell Avizo that all files with file name extensions *trimesh* or *tm* should be opened using the *Trimesh* reader.

13.3.2.3   More About Read Routines

The basic structure of a read routine should be clear from the examples presented in the previous two sections. Nevertheless, there are a few more things that might be of interest in some situations. These will be discussed in the following.

Reading Multiple Images At Once   The Avizo file browser allows you to select multiple files at once. Usually, all these files are opened one after the other by first determining the file format and then calling the appropriate read routine. However, in some cases the data of a single Avizo data object are distributed among multiple files. The most prominent example is 3D images where every slice is stored in a separate 2D image file. In order to be able to create a full 3D image, the file names of all the individual 2D images must be available to a read routine. To facilitate this, read routines in Avizo can have two different signatures. Besides the ordinary form

```
int myreader(const char* filename);
```

read routines can also be defined as follows:

```
int myreader(int n, const char** filenames);
```

In both cases exactly the same `dataFile` command can be used in the package resource file. Avizo automatically detects whether a read routine takes a single file name as an argument or multiple ones. In the latter case, the read routine is called with the names of all files selected in the file browser, provided all these files have the same file format (if multiple files with different formats are selected, the read routine for each format is called with the matching files only). You can create the template of a multiple files read routine by selecting the toggle *create reader for multiple files* in the Development Wizard (see subsection 13.2.7).

The Load Command   The current state of the Avizo project with all its data objects and modules can be stored in a script file. When executed, the script should restore the Avizo project again. Of course, this is a difficult task especially if data objects have been modified since they have been loaded from files. However, even if this is not the case, Avizo must know how to reload the data later on.

For this purpose, a special parameter called *LoadCmd* should be defined for the data object. This parameter should contain a Tcl command sequence which restores the data object on execution. Usually, the load command is simply set to `load <filename>` when calling `HxData::registerData`. However, this approach fails if the format of the file cannot be detected automatically, or if multiple data objects are created from a single file, e.g., as in our *Trimesh* example.

In such cases, the load command should be set manually. In case of the *Trimesh* reader, this could be done by adding the following lines of code at the very end of the routine just before the method's returning point:

```
QString loadCmd = QString(
                    "set TMPIO [load -trimesh \"%1\"]\n"
                    "lindex $TMPIO 0")
                    .arg(filename);
surface->setLoadCmd(loadCmd, 1);

for (i = 0; i < fields.size(); i++)
{
    loadCmd = QString("lindex $TMPIO %1").arg(i + 1);
    fields[i]->setLoadCmd(loadCmd, 1);
}
```

This code requires some explanation. The file is loaded and all data objects are created when the first line of the load command is executed. Note that we specified the `-trimesh` option as an argument of `load`. This ensures that the *Trimesh* reader will always be used. The format of the file to be loaded will not be determined automatically. The Tcl command `load` returns a list with the names of all data objects which have been created. This list is stored in the variable TMPIO. Later the names of the individual objects can be obtained by extracting the corresponding elements from this list. This is done using the Tcl command `lindex`.

Using Dialog Boxes in a Read Routine   In some cases, a file cannot be read successfully unless certain parameters are interactively specified by the user. Usually this means that a special-purpose dialog must be popped up within the read routine. This is done, for example, when raw data are read in Avizo. In order to write your own dialogs, you must use Qt, a platform-independent toolkit for

designing graphical user interfaces. Qt is included with Avizo XPand Extension under LGPL licensing, so that you can easily use it to create custom dialogs in Avizo.

If you don't want to use Qt, you may consider implementing your read routine within an ordinary module. Although this somewhat breaks Avizo's data import concept, it will work too, of course. You then can utilize ordinary ports to let the user specify required import parameters.

13.3.3 Write Routines

Like read routines, write routines in Avizo are C++ functions, either global ones or static member functions of an arbitrary class. In the following discussion, we present write routines for the same two formats for which reader codes have been explained in the previous section. First, *a writer for scalar fields* will be discussed, then *a writer for surfaces and surface fields.*

13.3.3.1 A Writer for Scalar Fields

In this section, we explain how to implement a routine for writing 3D images, i.e., instances of the class *HxUniformScalarField3*, to a file using the PPM3D format introduced in subsection 13.3.2.1. The writer is even simpler than the reader. Again, the source code is contained in the example package of Avizo XPand Extension. Once you have created a local Avizo directory using the *Development Wizard* and copied the example package into that directory, you will find the write routine in the local Avizo directory under `src/mypackage/writeppm3d.cpp`. Here it is:

```
///////////////////////////////////////////////////////////////////
//
// Sample write routine for the PPM3D file format
//
///////////////////////////////////////////////////////////////////

#include "api.h"                          // storage-class specification
#include <hxcore/HxMessage.h>             // for output in console
#include <hxfield/HxUniformScalarField3.h> // class representing 3D images
#include <hxqt/QxStringUtils.h>           // String conversion functions

MYPACKAGE_API
int
writeppm3d(HxUniformScalarField3* field, const char* filename)
{
    // For the moment we only want to support byte data
    if (field->primType() != McPrimType::MC_UINT8)
    {
        theMsg->printf("This format only supports byte data.");
        return 0; // indicate error
    }

    FILE* f = fopen(filename, "w"); // open the file

    if (!f)
    {
        theMsg->ioError(toQString(filename));
        return 0; // indicate error
```

```
    }

    // Write header:
    fprintf(f, "# PPM3D\n");

    // Write fields dimensions:
    const McDim3l& dims = field->lattice().getDims();
    fprintf(f, "%lli %lli %lli\n", dims[0], dims[1], dims[2]);

    // Write dims[0]*dims[1]*dims[2] data values:
    unsigned char* data = (unsigned char*)field->lattice().dataPtr();
    for (int i = 0; i < dims[0] * dims[1] * dims[2]; i++)
    {
        fprintf(f, "%d ", data[i]);
        if (i % 20 == 19) // Do some formatting:
            fprintf(f, "\n");
    }

    // Close the file.
    fclose(f);

    return 1; // indicate success
}
```

At the beginning, the same header files are included as in the reader.

api.h provides import and export storage-class specifiers for Windows systems. These are encoded in the macro `MYPACKAGE_API`. On Unix systems, this macro is empty and can be omitted.

HxMessage.h provides the global pointer *theMsg*, which allows us to print out text messages in the Avizo console window.

Finally, *HxUniformScalarField3.h* contains the declaration of the data class to be written to the file.

The signature of a write routine differs from that of a read routine. It takes two arguments, namely a pointer to the data object to be written to a file, as well as the name of the file. Before a write routine is called, Avizo always checks if the specified file already exists. If this is the case, the user is asked if the existing file should be overwritten. Therefore, such a check need not to be coded again in each write routine. Like a read routine, a write routine should return 1 on success, or 0 if an error occurred and the data object could not be saved.

The body of the write routine is almost self-explanatory. At the beginning, a check is made whether the 3D image really consists of byte data. In general, the type of data values of such an image can be 8-bit bytes, 16-bit shorts, 32-bit integers, floats, or doubles. If the image does contain bytes, a file is opened and the image contents are written into it. However, note that the data object also contains information which cannot be stored using our simple PPM3D file format. First of all, this applies to the bounding box of the image volume, i.e., the position of the center of the first and the last voxel in world coordinates. Also, all parameters of the object (defined in the member variable *parameters* of type *HxParamBundle*) will be lost if the image is written into a PPM3D file and read again.

Like a read routine, a write routine must be registered in the package resource file, i.e., in `mypackage/share/resources/mypackage.rc`. This is done by the following statement:

```
dataFile -name "PPM3D Demo Format"    \
    -save "writeppm3d"                \
    -type "HxUniformScalarField3"     \
    -package "mypackage"
```

The option `-save` specifies the name of the write routine. The option `-type` specifies the C++ class name of the data objects which can be saved using this format. Note that an export format may be registered for multiple C++ objects of different type. In this case, multiple `-type` options should be specified. However, for each type there must be a separate write routine with a different signature (polymorphism). For example, if we additionally want to register the PPM3D format for objects of type *HxStackedScalarField3*, we must additionally implement the following routine:

```
int writeppm3d(HxStackedScalarField3* field, const char* fname);
```

Besides the standard data classes, there are so-called *interface classes* that may be specified with the `-type` option. For example, in this way it is possible to implement a generic writer for n-component regular 3D fields. Such data is encapsulated by the interface *HxLattice3*. For more information about interfaces, refer to section 13.5.

At this point, you may try to compile and execute the write routine by following the instructions given in subsection 13.1.5 (Compiling and Debugging).

13.3.3.2 A Writer for Surfaces and Surface Fields

For the sake of completeness, a writer for the *Trimesh* format introduced in subsection 13.3.2.2 is described in this subsection. Remember that the *Trimesh* format is suitable for storing a triangular mesh as well as an arbitrary number of data values defined on the vertices of the surface. In Avizo, surfaces and data fields defined on surfaces are represented by different objects. This also has some implications when designing a write routine.

In our example, we actually implement two different write routines, one for the surface and one for the surface field. If the user selects the surface and exports it using the `Trimesh` writer, the surface mesh as well as all attached data fields will be written to file. On the other hand, if the user selects a particular surface field, the corresponding surface and just the selected field will be written.

The source code of the writer can be found in the local Avizo directory under `src/mypackage/writetrimesh.cpp`. Remember that the example package must be copied into the local Avizo directory before compiling. For details, refer to subsection 13.2.2. Again, let us start by looking at the code:

```
////////////////////////////////////////////////////////////////
//
// Write routine for the trimesh file format
//
////////////////////////////////////////////////////////////////

#include "api.h"
#include <hxcore/HxMessage.h>
#include <hxsurface/HxSurface.h>
#include <hxsurface/HxSurfaceField.h>
```

```
#include <hxqt/QxStringUtils.h>

static int
writetrimesh(HxSurface* surface,
             McDArray<HxSurfaceField*> fields,
             const char* filename)
{
    FILE* f = fopen(filename, "w");

    if (!f)
    {
        theMsg->ioError(toQString(filename));
        return 0;
    }

    int i, j, k;
    McDArray<McVec3f>& points = surface->points();
    McDArray<Surface::Triangle>& triangles = surface->triangles();

    // Write number of points and number of triangles
    fprintf(f, "%ld %ld\n", points.size(), triangles.size());

    // Write point coordinates
    for (i = 0; i < points.size(); i++)
    {
        McVec3f& v = points[i];
        fprintf(f, "%g %g %g\n", v[0], v[1], v[2]);
    }

    // Write point indices of all triangles
    for (i = 0; i < triangles.size(); i++)
    {
        int* idx = triangles[i].points;
        fprintf(f, "%d %d %d\n", idx[0] + 1, idx[1] + 1, idx[2] + 1);
    }

    // If there are data fields write them out, too.
    if (fields.size())
    {
        for (j = 0; j < fields.size(); j++)
            fprintf(f, "%d ", fields[j]->nDataVar());
        fprintf(f, "\n");

        for (i = 0; i < points.size(); i++)
        {
            for (j = 0; j < fields.size(); j++)
            {
                int n = fields[j]->nDataVar();
                float* v = &fields[j]->dataPtr()[i * n];
                for (k = 0; k < n; k++)
                    fprintf(f, "%g ", v[k]);
            }
            fprintf(f, "\n");
```

```
        }
    }

    fclose(f); // done
    return 1;
}

MYPACKAGE_API
int
writetrimesh(HxSurface* surface, const char* filename)
{
    // Temporary array of surface data fields
    McDArray<HxSurfaceField*> fields;

    // Check if there are data fields attached to surface
    for (int i = 0; i < surface->downStreamConnections.size(); i++)
    {
        HxObject* field = surface->downStreamConnections[i]->getObject();
        if (field->isOfType(HxSurfaceField::getClassTypeId()) &&
            ((HxSurfaceField*)field)->getEncoding() == HxSurfaceField::ON_NODES)
            fields.append((HxSurfaceField*)field);
    }

    // Write surface and all attached data fields
    return writetrimesh(surface, fields, filename);
}

MYPACKAGE_API
int
writetrimesh(HxSurfaceField* field, const char* filename)
{
    // Check if data is defined on nodes
    if (field->getEncoding() != HxSurfaceField::ON_NODES)
    {
        theMsg->printf("Data must be defined on nodes.");
        return 0;
    }

    // Store pointer to field in dynamic array
    McDArray<HxSurfaceField*> fields;
    fields.append(field);

    // Write surface and this data field
    return writetrimesh(field->surface(), fields, filename);
}
```

In the upper part of the code, first a static utility method is defined which takes three arguments: a pointer to a surface, a dynamic array of pointers to surface fields, and a file name. This is the function that actually writes the data to a file. Once you have understood the *Trimesh* reader presented in subsection 13.3.2.2, it should be no problem to follow the writer code too.

In the lower part of the code, two write routines mentioned above are defined, one for surfaces and the other one for surface fields. Since these routines are to be exported for external use, we need to apply

the package macro `MYPACKAGE_API`, at least on Windows.

Let us now look more closely at the surface writer. This routine first collects all surface fields attached to the surface in a dynamic array. This is done by scanning `surface->downStreamConnections` which provides a list of all objects attached to the surface. The class type of each object is checked using the method `isOfType`. This sort of dynamic type-checking is the same as in Open Inventor. If a surface field has been found and if it contains data defined on its nodes, it is appended to the temporary array `fields`. The surface itself, as well as the collected fields, are then written to file by calling the utility method defined in the upper part of the writer code.

The second write routine, the one adapted to surface fields, is simpler. Here a dynamic array of fields is used too, but this array is filled with data representing the original surface field only. Once this has been done, the same utility method can be called as in the first case.

Although actually two write routines have been defined, only one entry in the package resource file is required. This entry looks as follows (see `mypackage/share/resources/mypackage.rc`):

```
dataFile -name "Trimesh Demo Format"    \
    -ext "trimesh"                      \
    -type "HxSurface"                   \
    -type "HxSurfaceField"              \
    -save "writetrimesh"                \
    -package "mypackage"
```

In order to compile and execute the write, please follow the instructions given in subsection 13.1.5 (Compiling and Debugging).

13.3.4 Use the AmiraMesh API to read and write files in Avizo data format

Besides many standard file formats, Avizo also provides its own native format called Avizo data format. The Avizo data file format is very flexible. It can be used to save many different data objects including image data, finite-element grids, and solution data defined on such grids. Among other features, it supports ASCII or binary data encoding, data compression, and storage of arbitrary parameters. The format itself is described in more detail in the reference section of the users guide. In this section, we want to discuss how to save custom data objects in Avizo format. For this purpose, a special C++ utility class called `AmiraMesh` is provided. Using this class, reading and writing files in Avizo format becomes very easy.

Below we will first provide an *overview* of the AmiraMesh API. After that, we present two simple examples. In the first one, we show how colormaps are *written* in Avizo format. In the second one, we show how such colormaps are *read back* again.

13.3.4.1 Overview

The AmiraMesh API consists of a single C++ class. This class is called `AmiraMesh`. It is defined in the header file `include/amiramesh/AmiraMesh.h` located in the Avizo root directory. The class is designed to completely represent the information stored in an Avizo file in memory. When reading a file, first an instance of an `AmiraMesh` class is created. This instance can then be interpreted and the data contained in it can be copied into a matching Avizo data object. Likewise, when writing

a file, first an instance of an `AmiraMesh` class is created and initialized with all required information. Then this instance is written to file simply by calling a member method.

If you look at the header file or at the `AmiraMesh` class documentation, you will notice that there are four public methods called `setParameters`, `setLocationList`, `setDataList`, and `setFieldList`. These methods completely store the information contained in a file. The first method store the parameter bundle (with a `HxParamBundle` parameter). Like in an Avizo data object, it is used to store an arbitrary hierarchy of parameters. The other three methods manipulate dynamic arrays of pointers to locally defined classes. The most important local classes are `Location` and `Data`, which are stored in `locationList` and `dataList`, respectively. Four accessors `getLocationList`, `setLocationList`, `getDataList` and `setDataList` are created to manipulate this two lists.

A `Location` defines the name of a single- or multi-dimensional array. It does not store any data by itself. This is done by a `Data` class. Every `Data` class must refer to some `Location`. For example, when writing a tetrahedral grid, we may define two different one-dimensional locations, one called *Nodes* and the other one called *Tetrahedra*. On the nodes, we define a `Data` instance for storing the x-, y-, and z-coordinates of the nodes. Likewise, on the tetrahedra we define a `Data` instance for storing the indices of the four points of a tetrahedron.

As stated in the `AmiraMesh` class documentation, the *Data* class can take a pointer to some already existing block of memory. In this way, it is prevented the copy of all data before it is written to file. In order to write compressed data, the member method `setCodec` has to be called. Currently, two different compression schemes are supported. The first one, called *HxByteRLE*, implements simple run-length encoding on a per-byte basis. The second one, called *HxZip*, uses a more sophisticated compression technique provided by the external *zlib* library. In any case, the data will be automatically uncompressed when reading an Avizo data file.

It should be pointed out that the Avizo data file format itself merely provides a method for storing arbitrary data organized in single- or multi-dimensional arrays in a file. It does not specify anything about the semantics of the data. Therefore, when reading an Avizo data file, it is not clear what kind of data object should be created from it. To facilitate file I/O of custom data objects, the actual contents of an Avizo data file are indicated by a special parameter called *ContentType*. For each such type, a special read routine is registered. Like an ordinary read routine, an Avizo data reader is a global function or a static member method of a C++ class. It has the following signature:

```
int readMyAvizoFile(AmiraMesh* m, const char* filename);
```

This method is called whenever the *ContentType* parameter matches the one for which the read method is registered. The reader should create an Avizo data object from the contents of the AmiraMesh class. The filename can be used to define the name of the resulting data object. In order to register an Avizo read routine, a statement similar to the following one must be put into the package resource file:

```
AvizoFormat -ContentType "MyType" \
    -load "readMyAvizoFile" \
    -package "mypackage"
```

13.3.4.2 Writing an Avizo Data File

As a concrete example, in this section, we want to show how a colormap is written in Avizo data format. In particular, we consider colormaps of type `HxColormap256`, consisting of N discrete RGBA tuples. Like most other write methods, the Avizo data writer is a global C++ function. Let us first look at the code before discussing the details.

```
HXCOLOR_API
int write(HxColormap256* map, const char* filename)
{
    float minmax[2];
    minmax[0] = map->minCoord();
    minmax[1] = map->maxCoord();
    int size = map->getSize();

    AmiraMesh m;
    m.parameters = map->parameters;
    m.parameters.set("MinMax", 2, minmax);
    m.parameters.set("ContentType", "Colormap");

    AmiraMesh::Location* loc =
        new AmiraMesh::Location("Lattice", 1, &size);
    m.insert(loc);

    AmiraMesh::Data* data = new AmiraMesh::Data("Data", loc,
        McPrimType::mc_float, 4, (void*) map->getDataPtr());
    m.insert(data);

    if ( !m.write(filename,1) ) {
        theMsg->ioError(filename);
        return 0;
    }

    setLoadCmd(filename);
    return 1;
}
```

In the first part of the routine, a variable `m` of type `AmiraMesh` is defined. The parameters of the colormap are copied into `m`. In addition, two more parameters are set. The first one, called *MinMax*, describes the coordinate range of the colormap. The second one indicates the content type of the Avizo data file. This parameter ensures that the colormap can be read back again by a matching Avizo data read routine (see subsection 13.3.4.3).

Before the RGBA data values can be stored, a `Location` of the right size must be created and inserted into the `AmiraMesh` class. Afterwards, an instance of a `Data` class is created and inserted. The constructor of the `Data` class takes a pointer to the `Location` as an argument. Moreover, a pointer to the RGBA data values is specified. Each RGBA tuple consists of four numbers of type float.

13.3.4.3 Reading an Avizo Data File

In the previous section, we presented a simple Avizo data write routine for colormaps. We now want to read back such files again. For this reason, we define a static Avizo data read function in class

HxColormap256. Of course, a global C++ function could be used as well. The read function is
registered in the package resource file hxcolor.rc in the following way:

```
AvizoFormat -ContentType "Colormap" \
    -load "HxColormap256::readAmiraMesh" \
    -package "hxcolor"
```

This statement indicates that the static member method readAmiraMesh of the class
HxColormap256 defined in package hxcolor should be called if the Avizo data file contains a
parameter *ContentType* equal to *Colormap*. The source code of the read routine looks as follows:

```
int
HxColormap256::readAmiraMesh(AmiraMesh* m, const char* filename)
{
    int count = 0;

    for (int i = 0; i < m->dataList.size(); i++)
    {
        AmiraMesh::Data* data = m->dataList[i];

        if (data->getLocation()->getNumberOfDimensions() != 1)
            continue;

        if (data->getDimension() < 3 || data->getDimension() > 4)
            continue;

        int dim = data->getDimension();
        int size = data->getLocation()->getDimensions()[0];

        HxColormap256* colormap =
            (HxColormap256*)HxResource::createObject("HxColormap256");

        if (!colormap)
            return 0;

        colormap->resize(size);
        colormap->parameters = m->parameters;
        colormap->historyLog = m->historyLog;

        switch (data->getPrimitiveType().getType())
        {
            case McPrimType::MC_UINT8:
            {
                unsigned char* src = (unsigned char*)data->getDataPtr();
                for (int k = 0; k < size; k++, src += dim)
                {
                    float a = (dim > 3) ? (src[3]) / 255.0 : 1;
                    colormap->setRGBA(k,
                                        src[0] / 255.,
                                        src[1] / 255.,
                                        src[2] / 255.,
                                        a);
                }
```

```
    }
    break;

    case McPrimType::MC_INT16:
    {
        short* src = (short*)data->getDataPtr();
        for (int k = 0; k < size; k++, src += dim)
        {
            float a = (dim > 3) ? (src[3]) / 255.0 : 1;
            colormap->setRGBA(k,
                              src[0] / 255.,
                              src[1] / 255.,
                              src[2] / 255.,
                              a);
        }
    }
    break;
    case McPrimType::MC_UINT16:
    {
        unsigned short* src = (unsigned short*)data->getDataPtr();
        for (int k = 0; k < size; k++, src += dim)
        {
            float a = (dim > 3) ? (src[3]) / 255.0 : 1;
            colormap->setRGBA(k,
                              src[0] / 255.,
                              src[1] / 255.,
                              src[2] / 255.,
                              a);
        }
    }
    break;
    case McPrimType::MC_INT32:
    {
        int* src = (int*)data->getDataPtr();
        for (int k = 0; k < size; k++, src += dim)
        {
            float a = (dim > 3) ? (src[3]) / 255.0 : 1;
            colormap->setRGBA(k,
                              src[0] / 255.,
                              src[1] / 255.,
                              src[2] / 255.,
                              a);
        }
    }
    break;

    case McPrimType::MC_FLOAT:
    {
        float* src = (float*)data->getDataPtr();
        for (int k = 0; k < size; k++, src += dim)
        {
            float a = (dim > 3) ? src[3] : 1;
            colormap->setRGBA(k, src[0], src[1], src[2], a);
```

```
            }
        }
        break;

        case McPrimType::MC_DOUBLE:
        {
            double* src = (double*)data->getDataPtr();
            for (int k = 0; k < size; k++, src += dim)
            {
                float a = (dim > 3) ? src[3] : 1;
                colormap->setRGBA(k, src[0], src[1], src[2], a);
            }
        }
        break;
    }

    float minmax[2] = { 0, 1 };
    m->parameters.findReal("MinMax", 2, minmax);
    colormap->HxColormap::setMinMax(minmax[0], minmax[1]);

    {
        int localInterpolate;
        if (m->parameters.findNum("Interpolate", localInterpolate))
        {
            colormap->setInterpolate(localInterpolate ? true : false);
        }

        QString outOfBoundsBehavior;
        if (m->parameters.findString("OutOfBoundsBehavior", outOfBoundsBehavior))
        {
            // i.e = 0
            OutOfBoundsBehavior behavior = HxColormap256::DEFAULT_CLAMP;
            if (outOfBoundsBehavior.contains("CycleLeft"))
                *(int*)&behavior |= CYCLE_BEFORE_MIN;
            if (outOfBoundsBehavior.contains("CycleRight"))
                *(int*)&behavior |= CYCLE_AFTER_MAX;
            colormap->setOutOfBoundsBehavior(behavior);
        }

        int isLabelField;
        if (m->parameters.findNum("LabelField", isLabelField))
        {
            colormap->setLabelField(isLabelField);
            colormap->editMinMax.setMinMaxEnabled(!isLabelField);
        }
    }

    // Set the default alpha curve if the file does not provide an alpha channel.
    if (dim != 4)
    {
        colormap->setAlphaCurve(AC_SOFT_RAMP);
    }
```

```
        registerData(colormap, toQString(filename));
        count++;
    }

    return count;
}
```

Compared to the write routine, the read routine is a little bit more complex since some consistency checks are performed. First, the member `dataList` of the `AmiraMesh` structure is searched for a one-dimensional array containing vectors of three or four elements of type byte or float. This array should contain the RGB or RGBA values of the colormap. If a matching `Data` structure is found, a new instance of type `HxColormap256` is created. The parameters are copied from the `AmiraMesh` class into the new colormap. Afterwards, the actual color values are copied. Although the write routine only exports RGBA tuples of type float, the read routine also supports byte data. For this reason, two different cases are distinguished in a switch statement. If the file only contains 3-component data, the opacity value of each colormap entry is set to 1. Finally, the coordinate range of the colormap is set by evaluating the 2-component parameter *MinMax*, and the new colormap is added to the Project View by calling `HxData::registerData`.

13.4   Writing Modules

Besides the data classes, modules are the core of Avizo. They contain the actual algorithms for visualization and data processing. Modules are instances of C++ classes derived from the common base class `HxModule`.

There are two major groups of modules: *compute modules* and *display modules*. The first group usually performs some sort of operation on input data, creates some resulting data object, and deposits the latter in the Project View. In contrast, display modules usually directly visualize their input data. In this section, both types of modules will be covered in separate sections. For each case, a concrete example will be presented and discussed in detail.

In addition, we also discuss the Avizo*Plot API* in this section. This API makes it possible to create simple line plots or bar charts within a module.

Moreover, Avizo allows you to create *compute module on GPU* in CUDA. For each case a concrete example will be presented and discussed in detail.

13.4.1   A Compute Module

As already mentioned, *compute modules* usually take one or more input data objects and calculate a new resulting data object from these. The resulting data object is deposited in the Project View. Compute modules are represented by red icons in the Project View. They are derived from the base class `HxCompModule`.

In order to learn how to implement a new compute module, we will take a look at a concrete example. In particular, we want to write a compute module which performs a threshold operation on a 3D image, i.e., on an input object of type *HxUniformScalarField3*. The module produces another 3D image as output. In the resulting image, all voxels with a value below a user-specified minimum value or above a maximum value should be set to zero.

For easier understanding, we start with a very simple and limited version of the module. Then we iteratively improve the code. In particular, we proceed in three steps:

- *Version 1*: merely scans the input image, does not yet produce a result
- *Version 2*: creates an output object as result, uses the progress bar
- *Version 3*: adds an *Apply* button, overwrites the existing result if possible

You can find the source code of all three versions in the example package provided with Avizo XPand Extension, i.e., under `src/mypackage` in the local Avizo directory. For each version, there are two files: a header file called `MyComputeThresholdN.h` and a source code file called `MyComputeThresholdN.cpp` (where `N` is either 1, 2, or 3). Since the names are different, you can compile and execute all three versions in parallel.

In order to create a new local Avizo directory, please follow the instructions given in subsection 13.2.2. In order to compile the example package, please refer to subsection 13.1.5 (Compiling and Debugging).

13.4.1.1 Version 1: Skeleton of a Compute Module

The first version of our module does not yet produce any output. It simply scans the input image and prints the number of voxels above and below the threshold.

Like most other modules, our compute module consists of a header file containing the class declaration as well as a source file containing the actual code (or the class definition). Let us look at the header file `MyComputeThreshold1.h` first:

```
///////////////////////////////////////////////////////////
//
// Example of a compute module (version 1)
//
///////////////////////////////////////////////////////////
#ifndef MY_COMPUTE_THRESHOLD1_H
#define MY_COMPUTE_THRESHOLD1_H

#include "api.h"                      // storage-class specification
#include <hxcore/HxCompModule.h>      // include declaration of base class
#include <hxcore/HxPortFloatTextN.h>  // provides float text input

class MYPACKAGE_API MyComputeThreshold1 : public HxCompModule
{
    HX_HEADER(MyComputeThreshold1); // required for all base classes

public:
    // This virtual method will be called when the port changes.
    virtual void compute();

    // A port providing float text input fields.
    HxPortFloatTextN portRange;
};

#endif
```

As usual in C++ code, the file starts with a define statement that prevents the contents of the file from being included multiple times. Then three header files are included.

The package header file `api.h` is included. This file provides import and export storage-class specifiers for Windows systems. These are encoded in the macro `MYPACKAGE_API`. A class declared without this macro will not be accessible from outside the DLL it is defined in.

Following `api.h`, `HxCompModule.h` contains the definition of the base class of our compute tool. The last file, `HxPortFloatTextN.h`, contains the definition of a *port* we want to use in our class. A port represents an input parameter of a tool. In our case, we use a port of type `HxPortFloatTextN`. This port provides one or more text fields where the user can enter floating point numbers. The required text fields and labels are created automatically within the port constructor. As a programmer, you simply put some ports into your tool, specifying their types and labels, and do not have to bother creating a user interface for it.

In the rest of the header file, nothing more is done than deriving a new class from `HxCompModule`. The macro `HX_HEADER` is mandatory. This macro defines especially the default constructor and destructor of the class. A member function is defined, namely an overloaded virtual method called `compute`. The `compute` method is called when the module has been created and whenever a change of state occurs on one of the module's input data objects or ports. In fact, a connection to an input data object is also established by a port, as we shall see later on. In this example, we just declare one port in our class, specifically an instance of type `HxPortFloatTextN`.

The corresponding source file looks like this:

```
////////////////////////////////////////////////////////////////////
//
// Example of a compute module (version 1)
//
////////////////////////////////////////////////////////////////////

#include <QApplication>

#include "MyComputeThreshold1.h"        // declaration of this class
#include <hxcore/HxMessage.h>           // for output in console
#include <hxfield/HxUniformScalarField3.h> // class representing 3D images

HX_INIT_CLASS(MyComputeThreshold1, HxCompModule) // required macro

MyComputeThreshold1::MyComputeThreshold1()
    : HxCompModule(HxUniformScalarField3::getClassTypeId())
    , portRange(this,
                "range",
                QApplication::translate("MyComputeThreshold1", "Range"),
                2) // we want to have two float fields
{
}

MyComputeThreshold1::~MyComputeThreshold1()
{
}

void
```

```
MyComputeThreshold1::compute()
{
    // Access the input data object. The member portData (which is of
    // type HxConnection) is inherited from the base class HxModule.
    HxUniformScalarField3* field = (HxUniformScalarField3*)portData.getSource();

    // Check whether the input port is connected
    if (!field)
        return;

    // Get the input parameters from the user interface:
    float minValue = portRange.getValue(0);
    float maxValue = portRange.getValue(1);

    // Access size of data volume:
    const McDim3l& dims = field->lattice().getDims();

    // Now loop through the whole field and count the pixels.
    int belowCnt = 0, aboveCnt = 0;
    for (int k = 0; k < dims[2]; k++)
    {
        for (int j = 0; j < dims[1]; j++)
        {
            for (int i = 0; i < dims[0]; i++)
            {
                // This function returns the value at this specfic grid node.
                // It implicitely casts the result to float if necessary.
                float value = field->evalReg(i, j, k);
                if (value < minValue)
                    belowCnt++;
                else if (value > maxValue)
                    aboveCnt++;
            }
        }
    }

    // Finally print the result.
    theMsg->printf("%d voxels < %g, %d voxels > %g\n",
                   belowCnt,
                   minValue,
                   aboveCnt,
                   maxValue);
}
```

Following the include statements and the obligatory HX_INIT_CLASS macro, the constructor is defined. The usual C++ syntax must be used in order to call the constructors of the base class and the class members. The constructor of the base class HxCompModule takes the class type of the input data object to which this module can be connected. Avizo uses a special run-time type information system that is independent of the rtti feature provided by ANSI C++ compilers.

The second method to define is the destructor.

The third method we have to implement is the compute method. We first retrieve a pointer to our input data object through a member called portData. This port is inherited from the base class

HxModule, i.e., every module has this member. The port is of type `HxConnection` and it is represented as a blue line in the user interface (if connected). The rest of the compute method is rather straightforward. The way the actual data are accessed and how the computation is performed, of course, is highly specific to the input data class and the task the module performs. In this case, we simply loop over all voxels of the input image and count the number of voxels below the minimum value and above the maximum value. In order to access a voxel's value, we use the *evalReg* method. This method is provided by any scalar field with regular coordinates, i.e., by any instance of class `HxRegScalarField3`. Regardless of the primitive data type of the field, the result will always be cast to float.

Compile the `mypackage` example package and restart Avizo. Instructions for compiling local packages are provided in subsection 13.1.5 (Compiling and Debugging). Load the file `chocolate-bar.am` from Avizo 's `data/tutorials` directory. Open the Create Object popup menu and click on `ComputeThreshold1` inside the *Local* category to attach the new module. Type in different threshold values in the *range* port of `ComputeThreshold1`, and look at the different results that appear in the *Console* window while you change *range* values:

```
0 voxels < 0, 8882137 voxels > 0
0 voxels < 0, 8845776 voxels > 1
0 voxels < 0, 8809669 voxels > 2
```

13.4.1.2 Version 2: Creating a Result Object

Now that we have a first working version of the module, we can add more functionality. First, we want to create a real output data object. Then we further want to improve the module by using Avizo's progress bar and by providing better default values for the range port. The header file of our module will not be affected by all these changes. We merely need to add some code in the source file `MyComputeThreshold2.cpp`.

Let us start with the output data object. In the compute method, just before the for-loop, we insert the following statements:

```
// Create output with the same primitive data type as input:
HxUniformScalarField3* output =
    new HxUniformScalarField3(dims, field->primType());

// Output shall have same bounding box as input:
output->coords()->setBoundingBox(field->getBoundingBox());
```

This creates a new instance of type `HxUniformScalarField3` with the same dimensions and the same primitive data type as the input data object. Since the output has the same bounding box, i.e., the same voxel size as the input, we copy the bounding box. Note that this approach will only work for fields with uniform coordinates. For other regular coordinate types, such as stacked or curvilinear coordinates, we refer to subsection 13.5.2.

After the output object has been created, its voxel values are not yet initialized. This is done in the inner part of the nested for-loops. The method `set`, used for this purpose, automatically performs a cast from float to the primitive data type of the output field. In summary, the inner part of the for-loop now looks as follows:

```
float value = field->evalReg(i,j,k);
float newValue = 0;

if (value<minValue)
    belowCnt++;
else if (value>maxValue)
    aboveCnt++;
else newValue = value;

output->set(i,j,k,newValue);
```

Creating a new data object using the `new` operator will not automatically make it appear in the Project View. Instead, we must explicitly register it. In a compute module, this can be done by calling the method `setResult`:

```
setResult(output); // register result
```

This method adds a data object to the Project View, if it is not already present there. In addition, it connects the object's *master* port to the compute module itself. Like any other connection, this link will be represented by a black line in the Project View. The master port of a data object may be connected to a compute module or to an editor. Such a master connection indicates that the data object is controlled by an 'upstream' component, i.e., that its contents may be overridden by the object to which it is connected.

Now that we have created an output object, let us address the progress bar. Although, for the test data set `chocolate-bar.am`, our threshold operation does not take very long, it is good practice to indicate that the application is busy when computations are performed that could take a long time on large input data. Even better is to show a progress bar, which is not difficult. Before the time-consuming part of the compute routine, i.e., before the nested for-loops, we add the following line:

```
// Turn the application into busy state,
// don't activate Stop button.
theProgress->startWorkingNoStop(QApplication::translate("MyComputeThreshold2",
"Computing threshold"));
```

We use the global instance `theProgress` of class `HxApplication` here. The corresponding header file must be included at the beginning of the source file. The method turns the application into the 'busy' state and displays a working message in the status line. As opposed to the method `startWorking`, this variant does not activate the stop button. See subsection 13.7.2 for details. When the computation is done, we must call

```
theProgress->stopWorking(); // stop progress bar
```

in order to switch off the 'busy' state again. Inside the nested for-loops, we update the progress bar just before a new 2D slice is processed. This is done by the following line of code:

```
// Set progress bar, the argument ranges between 0 and 1.
theProgress->setProgressValue((float)(k+1)/dims[2]);
```

The value of `(float)(k+1)/dims[2]` progressively increases from zero to one during computation. Note that you should not call `setProgressValue` in the inner of the three loops. Each call involves an update of the graphical user interface and therefore is relatively expensive. It is perfectly okay to update the progress bar several hundred times during a computation, but not several hundred thousand times.

Another slight improvement we have incorporated into the second version of our compute module concerns the `range` port. In the constructor, we have set new initial values for the minimum and maximum fields. While both values are 0 by default, we now set them to 410 and 950, respectively:

```
// Set default value for the range port:
portRange.setValue(0,410);  // min value is 410
portRange.setValue(1,950);  // max value is 950
```

You may now test this second version of the compute module by loading the test data set `chocolate-bar.am` from Avizo's `data/tutorials` directory. Attach the `ComputeThreshold2` module from the *Local* category inside the data popup menu. Open the *Console* window to watch results as you change the port *range* values. You can also see new output objects created in the *Project View* on each change. To better appreciate the progress bar, try to resample the input data, for example to 512x512x100, and connect the compute module to the resampled data set. However, be sure that you have enough main memory installed on your system.

13.4.1.3   Version 3: Reusing the Result Object

Testing the first two versions of our module, we saw that the module's compute method is triggered automatically when the module is created and whenever the range port is changed. Each time a new result output data object is created. This quickly fills up the computer's main memory as well as Avizo's graphical user interface. Therefore, we now change this behavior: A new result object is to be created only the first time. Whenever the range port is changed afterwards, the existing result object should be overridden. In order to achieve this, we modify the middle part of the compute method in the following way:

```
// Access size of data volume:
const McDim3l& dims = field->lattice().getDims();

// Check if there is a result which we can reuse.
HxUniformScalarField3* output = (HxUniformScalarField3*) getResult();

// Check for proper type.
if (output && !output->isOfType(HxUniformScalarField3::getClassTypeId()))
    output = 0;

// Check if size and primType still match the current input:
if (output) {
    const McDim3l& outdims = output->lattice().getDims();
    if ( dims !=outdims ||
        field->primType() != output->primType())
        output=0;
}
```

```
    // If necessary, create a new result data set.
    if (!output) {
        output = new HxUniformScalarField3(dims, field->primType());
        output->composeLabel(toQString(field->getName()),"masked");
    }
```

The `getResult` method checks whether there is a data set whose master port is connected to the compute module. This typically is the object set by a previous call to `setResult`. However, it also may be any other object. Therefore, a run-time type check must be performed by calling the `isOfType` member method of the output object. If the output object is not of type `HxUniformScalarField3`, the variable `output` will be set to null. Then a check is made whether the output object has the same dimensions and the same primitive data type as the input object. If this test fails, `output` will also be set to null. At the end, a new result object will only be created if no result exists already or if the existing result does not match the input. It is possible to interactively try different range values without creating a bunch of new results.

However, when one of the numbers of the range port is changed, computation starts immediately. Sometimes this may be desired, but in this case, we prefer to add an *Apply* button as present in many other compute modules. The user must explicitly push this button in order to start computation. In order to use the *Apply* button, the following line of code must be added in the public section of the module's header file:

```
    // Start computation when this button is clicked.
    HxPortDoIt portDoIt;
```

Of course, the corresponding include file `hxcore/HxPortDoIt.h` must be included as well. As for the other port, we must initialize `portDoIt` in the constructor of our module in the source file:

```
MyComputeThreshold3::MyComputeThreshold3() :
    HxCompModule(HxUniformScalarField3::getClassTypeId()),
    portRange(this,"range",QApplication::translate("MyComputeThreshold3", "Range"),2),
    // we want to have two float fields
    portDoIt(this,"action",QApplication::translate("MyComputeThreshold3", "Action"))
{
    ...

    // Set text of doIt button
    portDoIt.setLabel(0,QApplication::translate("MyComputeThreshold3", "DoIt"));
}
```

To achieve the desired behavior, we finally change our compute method so that it immediately returns unless the *Apply* button was pressed. This can be done by adding the following piece of code at the beginning of the compute method:

```
    // Check whether doIt button was hit
    if (!portDoIt.wasHit()) return;
```

With these changes, the module is already quite usable. Load the test data set `chocolate-bar.am` from Avizo's `data/tutorials` directory. Attach the `ComputeThreshold3` module from the

Local category inside the data popup menu. Press the *Apply* button, change the range and press *Apply* again. Open the *Console* window to view results. You may have notice, that only one output object has been created in the *Project View*. Attach an *Ortho Slice* module to the result while experimenting with the range (use the histogram mapping in the *Ortho Slice* in order to see small changes). Try to detach the connection between the result and the module and press *Apply* again: a new output is created.

Note: By default, the `HxPortDoIt` port is not visible in the control panel of its associated module. Rather, the fact that a module has an `HxPortDoIt` activates (makes green) the *Apply* button at the bottom of the Properties Area. To request display of the DoIt port in the module control panel, check the *Show DoIt buttons* box in the *Layout* tab of the *Edit/Preferences* dialog.

Finally, some remarks on performance. Although it is probably not critical in this simple example, performance typically becomes an issue in real-world applications. In the inner-most loop, calling the methods `field->evalReg` and `output->set` is convenient but rather expensive. For example, if the input consists of 16-bit signed integers like in `chocolate-bar.am`, these methods involve a cast from `16-bit signed` to `float` and back to `16-bit signed`.

The performance can be improved by writing code which explicitly handles a particular primitive data type. A pointer to the actual data values of a *HxUniformScalarField3* can be obtained by calling `field->lattice().dataPtr()`. The value returned by this method is of type `void*`. It must be explicitly cast to the data type to which the field actually belongs. The voxel values itself are arranged without any padding. This means that the index of voxel (i, j, k) is given by `(k*dims[1]+j)*dims[0]+i`, where `dims[0]` and `dims[1]` denote the number of voxels in the x and y directions, respectively.

13.4.2 A Display Module

Our next example is a module which displays some geometry in Avizo's 3D viewer. The module takes a surface model as input and draws a little cube at every vertex that belongs to *n* triangles, where *n* is a user-adjustable parameter.

From the previous section, we already know the basic idea: we derive a new class from the base class `HxModule`. Since this time our module does not produce a new data set, we directly use `HxModule` as base class instead of `HxCompModule`. As input, the module should accept data of class `HxSurface`. We need one additional port allowing the user to specify the parameter *n*. As in the previous section, we develop different versions of our module, thereby introducing new concepts step by step:

- *Version 1*:
  creates an Open Inventor scene graph and displays it in the viewer.

- *Version 2*:
  adds a colormap port, provides a parse method for Tcl commands.

- *Version 3*:
  implements a new display mode, dynamically shows or hides a port.

You can find the source code of all three versions of the module in the example package provided with Avizo XPand Extension, i.e., under `src/mypackage` in the local Avizo directory. For each version there are two files, a header file called `MyDisplayVerticesN.h` and a source code file

called `MyDisplayVerticesN.cpp` (where `N` is either 1, 2, or 3). Since the names are different, you can compile and execute all three version in parallel.

In order to create a new local Avizo directory, please follow the instructions given in subsection 13.2.2. In order to compile the example package, please refer to subsection 13.1.5 (Compiling and Debugging).

13.4.2.1 Version 1: Displaying Geometry

The first version of our module, called *MyDisplayVertices1*, merely detects the vertices of interest and displays them using little cubes. In order to understand the code, we first need to look more closely at the class `HxSurface`. As we can see in the reference documentation, a surface essentially contains an array of 3D points and an array of triangles. Each triangle has three indices pointing into the list of points. In order to count the triangles per vertex, we simply walk through the list of triangles and increment a counter for each vertex.

Once we have detected all interesting vertices, we are going to display them using small cubes. This is done by creating an Open Inventor scene graph. If you want to learn more about Open Inventor, you probably should look at *The Inventor Mentor*, an excellent book about Open Inventor published by Addison-Wesley. In brief, an Open Inventor scene graph is a tree-like structure of C++ objects which describes a 3D scene. Our scene is quite simple. It consists of one *separator node* containing several cubes, i.e., instances of class `SoCube`. Since an `SoCube` is always located at the origin, we put an additional node of type `SoTranslation` right before each `SoCube`. We adjust the size of the cubes so that each side is 0.01 times the length of the diagonal of the bounding box of the input surface.

After this short overview, we now look at the header file of the module. It is called `MyDisplayVertices1.h`:

```
#ifndef MY_DISPLAY_VERTICES1_H
#define MY_DISPLAY_VERTICES1_H

#include "api.h"                   // storage-class specification
#include <hxcore/HxModule.h>       // include declaration of base class
#include <hxcore/HxPortIntSlider.h> // provides integer slider
#include <mclib/McHandle.h>        // smart pointer template class

#include <Inventor/nodes/SoSeparator.h>

class MYPACKAGE_API MyDisplayVertices1 : public HxModule
{
    HX_HEADER(MyDisplayVertices1);

public:
    // Input parameter.
    HxPortIntSlider portNumTriangles;

    // This is called when an input port changes.
    virtual void compute();

protected:
    McHandle<SoSeparator> scene;
};
```

```
#endif
```

The header file can be understood quite easily. First, some other header files are included. Then the new module is declared as a child class of `HxModule`. As usual, the macros `MYPACKAGE_API` and `HX_HEADER` are mandatory, the last one including the definition of the default constructor and destructor. Our module implements a compute method. In addition, it has a port of type `HxPortIntSlider` which allows the user to specify the number of triangles of the vertices to be displayed.

A pointer to the actual Open Inventor scene is stored in the member variable `scene` of type `McHandle<SoSeparator>`. A `McHandle` is a so-called *smart pointer*. It can be used like an ordinary C pointer. However, each time a value is assigned to it, the reference counter of the referenced object is automatically increased or decreased. This is done by calling the `ref` or `unref` method of the object. If the reference counter becomes zero or less, the object is deleted automatically. We recommend using smart pointers instead of C pointers because they are safer.

The actual implementation of the module is contained in the file `MyDisplayVertices1.cpp`. This file looks as follows:

```
//////////////////////////////////////////////////////////////////////
//
// Example of a compute module (version 1)
//
//////////////////////////////////////////////////////////////////////

#include <QApplication>

#include "MyDisplayVertices1.h"  // header of this class
#include <hxcore/HxMessage.h>     // for output in console
#include <hxsurface/HxSurface.h> // class representing a surface

#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoTranslation.h>

HX_INIT_CLASS(MyDisplayVertices1, HxModule)

MyDisplayVertices1::MyDisplayVertices1()
    : HxModule(HxSurface::getClassTypeId())
    , portNumTriangles(this,
                       "numTriangles",
                       QApplication::translate("MyDisplayVertices1", "Num Triangles"))
{
    portNumTriangles.setMinMax(1, 12);
    portNumTriangles.setValue(6);
    scene = new SoSeparator;
}

MyDisplayVertices1::~MyDisplayVertices1()
{
    hideGeom(scene);
}

void
```

```
MyDisplayVertices1::compute()
{
    int i;

    // Access input object (portData is inherited from HxModule):
    HxSurface* surface = (HxSurface*)portData.getSource();

    if (!surface)
    { // Check if input object is available
        hideGeom(scene);
        return;
    }

    // Get value from input port, query size of surface:
    int numTriPerVertex = portNumTriangles.getValue();
    int nVertices = surface->points().size();
    int nTriangles = surface->triangles().size();

    // We need a triangle counter for every vertex:
    McDArray<unsigned short> triCount(nVertices);
    triCount.fill(0);

    // Loop through all triangles and increase counter of the vertices:
    for (i = 0; i < nTriangles; i++)
        for (int j = 0; j < 3; j++)
            triCount[surface->triangles()[i].points[j]]++;

    // Now create the scene graph. First remove all previous childs:
    scene->removeAllChildren();

    // Cube size should be 1% of the diagonal of the bounding box.
    float size = surface->getBoundingBox().getSize().length() * 0.01;

    // Pointer to surface coordinates casted from McVec3f to SbVec3f.
    SbVec3f* p = (SbVec3f*)surface->points().dataPtr();

    SbVec3f q(0, 0, 0); // position of last point
    int count = 0;      // vertex counter

    for (i = 0; i < nVertices; i++)
    {
        if (triCount[i] == numTriPerVertex)
        {
            SoTranslation* trans = new SoTranslation;
            trans->translation.setValue(p[i] - q);

            SoCube* cube = new SoCube;
            cube->width = cube->height = cube->depth = size;

            scene->addChild(trans);
            scene->addChild(cube);

            count++;
```

Writing Modules   **790**

```
            q = p[i];
        }
    }

    theMsg->printf("Found %d vertices belonging to %d triangles",
                   count,
                   numTriPerVertex);

    showGeom(scene); // finally show scene in viewer
}
```

A lot of things are happening here. Let us point out some of these in more detail now. The constructor initializes the base class with the type returned by `HxSurface::getClassTypeId`. This ensures that the module can only be attached to data objects of type `HxSurface`. The constructor also initializes the member variable `portNumTriangles`. The range of the slider is set from 1 to 12. The initial value is set to 6. Finally, a new Open Inventor separator node is created and stored in `scene`.

The destructor contains only one call, `hideGeom(scene)`. This causes the Open Inventor scene to be removed from all viewers (provided it is visible). The scene itself is deleted automatically when the destructor of `McHandle` is called.

The actual computation is performed in the `compute` method. The method returns immediately if no input surface is present. If an input surface exists, the numbers of triangles per point are counted. For this purpose, a dynamic array `triCount` is defined. The array provides a counter for each vertex. Initially it is filled with zeros. The counters are increased in a loop over the vertices of all triangles.

In the second part of the compute method, the Open Inventor scene graph is created. First, all previous children of `scene` are removed. Then the length of the diagonal of the input surface is determined. The size of the cubes will be set proportional to this length. For convenience, the pointer to the coordinates of the surface is stored in a local variable `p`. Actually, the coordinates are of type `McVec3<float>`. However, this class is fully compatible with the Open Inventor vector class `SbVec3f`. Therefore, the pointer to the coordinates can be cast as shown in the code.

After everything has been set up, every element of the array `triCount` is checked in a for-loop. If the value of an element matches the selected number of triangles per vertex, two new Inventor nodes of type `SoTranslation` and `SoCube` are created, initialized, and inserted into `scene`. Since the `SoTranslation` also affects all subsequent translation nodes, we must remember the position of the last point in `q` and subtract this position from the one of the current point. Alternatively, we could have encapsulated the `SoTranslation` and the `SoCube` in an additional `SoSeparator` node. However, this would have resulted in a more complex scene graph. At the very end of the compute method, the new scene graph is made visible in the viewer by calling `showGeom`. This method automatically checks if a node has already been visible. Therefore, it may be called multiple times with the same argument.

The module is registered in the usual way in the package resource file, i.e., in `mypackage/share/resource/mypackage.rc`. Once you have compiled the example package, you may test the module by loading the surface `mypackage/data/test.surf` located in the local Avizo directory. Attach the module *DisplayVertices1* from the *Local* category inside the popup menu of the data.

13.4.2.2 Version 2: Adding Color and a Parse Method

In this section, we want to add two more features to our module. First, we want to use a colormap port which allows us to specify the color of the cubes. Second, we want to add a parse method which allows us to specify additional Tcl commands for the module.

A colormap port is used to establish a connection to a colormap, i.e., to a class of type `HxColormap`. It is derived from `HxConnection` but, in contrast to the base class, it provides a graphical user interface showing the contents of the colormap and letting the user change its coordinate range. If no colormap is connected to the port, a default color is displayed. The default color can be edited by the user by double-clicking the color bar.

In order to provide our module with a colormap port, we must insert the following line into the module's header file:

```
HxPortColormap portColormap;
```

Of course, we must also include the header file of the class `HxPortColormap`. This file is located in package `hxcolor`. Note that the order in which ports are displayed on the screen depends on the order in which the ports are declared in the header file. If we declare `portColormap` before `portNumTriangles`, the colormap port will be displayed before the integer slider.

In the compute method of our module, we add the following piece of code just after the previous children of the scene graph have been removed:

```
SoMaterial* material = new SoMaterial;
material->diffuseColor =
    portColormap.getColor(numTriPerVertex);
scene->addChild(material);
```

With these lines, we insert a material node right before all the translation and cube nodes into the separator. The material node causes the cubes to be displayed in a certain color. We call the `getColor` method of the colormap port in order to determine this color. If the port is not connected to a colormap, this method simply returns the default color. However, if it is connected, the color is taken from the colormap. As an argument, we specify `numTriPerVertex`, the number of triangles of the selected vertices. Depending on the value of `portNumTriangles`, the cubes, therefore, will be displayed in different colors. Of course, this requires that the range of the colormap extend from something like 1 to 10 or 12.

Besides the colormap port, we also want to add a Tcl command interface to our module. This is done by overloading the virtual method `parse` of `HxModule`. We, therefore, insert the following line into the module's class declaration:

```
virtual int parse(Tcl_Interp* t, int argc, char **argv);
```

In a parse method, special commands can be defined which allow us to control the module in a more sophisticated way. A typical application is to set special parameters which should not be represented by a separate port in the user interface. As an example, we want to provide a method which allows us to change the size of the cubes. In the initial version of the module, the cubes were adjusted so

that each side was 0.01 times the length of the diagonal of the bounding box of the input surface. The value of the scale factor shall now be stored in the member variable *scale*. In order to set and get this variable, two Tcl commands `setScale` and `getScale` shall be provided. The implementation of the parse method looks as follows:

```
int
MyDisplayVertices2::parse(Tcl_Interp* t, int argc, char** argv)
{
    if (argc < 2)
        return TCL_OK;
    QString cmd = QString::fromUtf8(argv[1]);

    if (cmdCheck(cmd, "setScale"))
    {
        ASSERTARG(3);
        scale = atof(argv[2]);
        fire();
    }
    else if (cmdCheck(cmd, "getScale"))
    {
        Tcl_VaSetResult(t, "%g", scale);
    }
    else
        return HxModule::parse(t, argc, argv);

    return TCL_OK;
}
```

Commands are defined in a sequence of if-else statements. For each command, the method `cmdCheck` should be used. At the end of the if-else sequence the parse method of the base class should be called. Note that after a command is issued, the compute method of the module will not be called automatically by default. This is in contrast to interactive changes of ports. However, we may explicitly call `fire` in a command like shown above. In this case, the size of the cubes then will be adjusted immediately. You may test the parse method by loading the file `mypackage/data/test.surf`, attaching `DisplayVertices2` to it, and then typing something like `DisplayVertices2 setScale 0.03` into the Avizo console window.

13.4.2.3 Version 3: Adding an Update Method

Besides a compute method, modules may also define an *update* method. This method is called just before the compute method and also whenever a module is selected. In the update method, the user-interface of the module can be configured, i.e., ports can be shown or hidden dynamically if this is required, the sensitivity of ports can be adjusted, or the number of entries of an option menu can be modified dynamically.

In order to illustrate how an update method might work, we implement an alternate display mode in our module. In this mode, all vertices of a surface should be displayed, not only the ones with a certain number of neighboring triangles. In this second mode, the slider `portNumTriangles` is not meaningful anymore. We, therefore, hide it by defining an appropriate update method. The following lines are added in the header file `MyDisplayVertices3.h`:

```
    // Mode: 0=selected vertices, 1=all vertices
    HxPortRadioBox portMode;

    // Shows or hides required ports.
    virtual void update();
```

The new radio box port lets the user switch between the two display modes. Like the compute method, the update method takes no argument and also has no return value.

If you look into the source code file `MyDisplayVertices3.cpp`, you will notice that the radio box port is initialized in the constructor of the module and that the text labels are set properly. The update method itself is quite simple:

```
void
MyDisplayVertices3::update()
{
    if (portMode.getValue() == 0)
        portNumTriangles.show();
    else
        portNumTriangles.hide();
}
```

The slider `portNumTriangles` is shown or hidden depending on the value of the radio box port. Note that before the update method is called, all ports are marked to be shown. Therefore, you must hide them every time `update` is called. For example, the `show` and `hide` calls should not be encapsulated by an if statement which checks if some input port is new.

In order to support the new all-vertices display style, we slightly modify the way the Open Inventor scene graph is created. Instead of a single `SoMaterial` node, we insert a new one whenever the color of a cube needs to be changed, i.e., whenever the number of triangles of a vertex differs from the previous one. The new for-loop looks as follows:

```
    int lastNumTriPerVertex = -1;
    int allVertices = portMode.getValue();

    for (i = 0; i < nVertices; i++)
    {
        if (allVertices || triCount[i] == numTriPerVertex)
        {
            if (triCount[i] != lastNumTriPerVertex)
            {
                SoMaterial* material = new SoMaterial;
                material->diffuseColor = portColormap.getColor(triCount[i]);
                scene->addChild(material);
                lastNumTriPerVertex = triCount[i];
            }

            SoTranslation* trans = new SoTranslation;
            trans->translation.setValue(p[i] - q);

            SoCube* cube = new SoCube;
            cube->width = cube->height = cube->depth = size;
```

```
            scene->addChild(trans);
            scene->addChild(cube);

            count++;
            q = p[i];
        }
    }
```

Again, you can test the module by loading the file `mypackage/data/test.surf` and attaching `DisplayVertices3` to it. If you connect the *physics.icol* colormap to the colormap port, adjust the colormap range to 1...9, and select the all-vertices display style, you should get an image similar to the one shown in Figure 13.10.
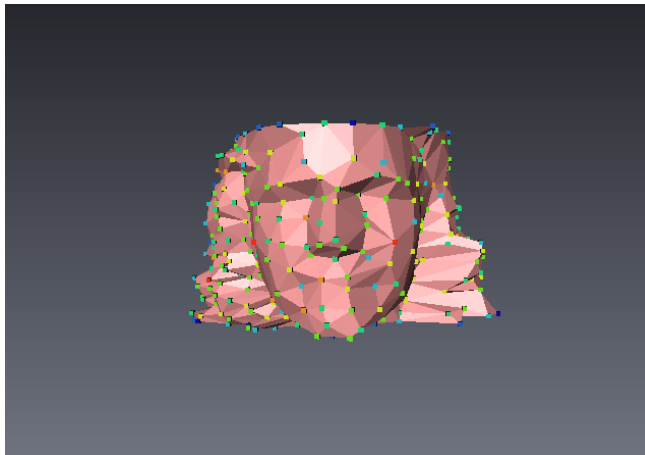
Figure 13.10: The example module *DisplayVertices3* displays little cubes at the vertices of a surface. The cubes are colored according to the number of neighboring triangles.

13.4.3   A Module With Plot Output

In some cases, you may want to show a simple 2D plot in an Avizo module, for example a histogram or some bar chart. To facilitate this task Avizo provides a special-purpose *Plot API* which can be used in any Avizo object, regardless of whether it is a compute module or a display module.

The class `PzEasyPlot` provides the necessary methods to open a plot window and to draw in that window. In the following, we illustrate how to use this class, again by means of an *example*. In particular, we are going to write a module which plots the number of voxels per slice for all materials defined in a label image. A label image usually represents the results of an image segmentation operation. For each voxel, there is a label indicating to which material the voxel belongs. Further features of the *further features of the Plot API* will be described in a separate section.

13.4.3.1 A Simple Plot Example

In this section, we show how to plot some simple curves using the class `PzEasyPlot`. As mentioned above, the curves represent the number of voxels per slice for the materials of a label image. For this purpose, we define a new module called `MyPlotAreaPerSlice`.

Like the other examples, this module is contained in the Avizo example package. In order to check out the example package, you must create a local Avizo directory as described in subsection 13.2.2. In order to compile the example package, please refer to subsection 13.1.5 (Compiling and Debugging).

Let us first look at the header file `MyPlotAreaPerSlice.h`:

```
/////////////////////////////////////////////////////////////////////
//
// Example of a plot module (header file)
//
/////////////////////////////////////////////////////////////////////
#ifndef MY_PLOT_AREA_PER_SLICE_H
#define MY_PLOT_AREA_PER_SLICE_H

#include "api.h"                       // storage-class specification
#include <hxcore/HxModule.h>          // include declaration of base class
#include <hxcore/HxPortButtonList.h> // provides a push button
#include <hxplot/PzEasyPlot.h>        // simple plot window

class MYPACKAGE_API MyPlotAreaPerSlice : public HxModule
{
    HX_HEADER(MyPlotAreaPerSlice);

public:
    // Shows the plot window.
    HxPortButtonList portAction;

    // Performs the actual computation.
    virtual void compute();

protected:
    McHandle<PzEasyPlot> plot;
};

#endif
```

The class declaration is very simple. The module is derived directly from `HxModule`. It provides a compute method and a port of type `HxPortButtonList`. In fact, we will only use a single push button in order to let the user pop up the plot window. The plot window class `PzEasyPlot` itself is referenced by a smart pointer, i.e., by a variable of type `McHandle<PzEasyPlot>`. We have already used smart pointers in subsection 13.4.2.1, for details see there.

Now let us take a look at the source file `MyPlotAreaPerSlice.cpp`:

```
/////////////////////////////////////////////////////////////////////
//
// Example of a plot module (source code)
//
```

```
/////////////////////////////////////////////////////////////////

#include "MyPlotAreaPerSlice.h" // declaration of this module
#include <QApplication>
#include <hxcore/HxApplication.h>
#include <hxcore/HxProgressInterface.h>
#include <hxfield/HxLabelLattice3.h> // represents segmentation results

HX_INIT_CLASS(MyPlotAreaPerSlice, HxModule)

MyPlotAreaPerSlice::MyPlotAreaPerSlice()
    : HxModule(HxLabelLattice3::getClassTypeId())
    , portAction(this,
                 "action",
                 QApplication::translate("MyPlotAreaPerSlice", "Action"),
                 1)
{
    portAction.setLabel(0,
                        QApplication::translate("MyPlotAreaPerSlice", "Show Plot"));
    plot = new PzEasyPlot("Area per slice");
    plot->autoUpdate(0);
}

MyPlotAreaPerSlice::~MyPlotAreaPerSlice()
{
}

void
MyPlotAreaPerSlice::compute()
{
    HxLabelLattice3* lattice =
        (HxLabelLattice3*)portData.getSource(HxLabelLattice3::getClassTypeId());

    // Check if valid input is available.
    if (!lattice)
    {
        plot->hide();
        return;
    }

    // Return if plot window is invisible and show button wasn't hit
    if (!plot->isVisible() && !portAction.isNew())
        return;

    theProgress->busy(); // activate busy cursor

    int i, k, n;
    const McDim3l& dims = lattice->getDims();
    unsigned char* data = lattice->getLabels<unsigned char>();
    int nMaterials = lattice->materials()->getNumberOfBundles();

    // One counter per material and slice
    McDArray<McDArray<float> > count(nMaterials);
```

Writing Modules   **797**

```
    for (n = 0; n < nMaterials; n++)
    {
        count[n].resize(dims[2]);
        count[n].fill(0);
    }

    // Count number of voxels per material and slice
    for (k = 0; k < dims[2]; k++)
    {
        for (i = 0; i < dims[1] * dims[0]; i++)
        {
            int label = data[k * dims[0] * dims[1] + i];
            if (label < nMaterials)
                count[label][k]++;
        }
    }

    plot->remData(); // remove old curves

    for (n = 0; n < nMaterials; n++) // add new curves
        plot->putData(
            lattice->materials()->getBundle(n)->getName().toLatin1().constData(),
            dims[2],
            count[n].dataPtr());

    plot->update(); // refresh display
    plot->show();   // show or raise plot window

    theProgress->notBusy(); // deactivate busy cursor
}
```

In the constructor, the base class `HxModule` is initialized with the class type ID of the class `HxLabelLattice3`. This class is not a data class derived from `HxData` but a so-called *interface*. Interfaces are used to provide a common API for objects not directly related by inheritance. In our case, `MyPlotAreaPerSlice` can be connected to any data object providing a `HxLabelLattice3` interface. This might be a `HxUniformLabelField3` but also a `HxStackedLabelField3` or something else.

Also in the constructor, a new plot window of type `PzEasyPlot` is created and stored in `plot`. Then the method `plot->autoUpdate(0)` is called. This means that we must explicitly call the `update` method of `PzEasyPlot` after the contents of the plot window are changed. Auto-update should be disabled when more than one curve is being changed at once.

As usual, the actual work is performed by the compute method. First, we retrieve a pointer to the label lattice. Since we want to use an interface instead of a data object itself, we must specify the class type ID of the interface as an argument of the `source` method of `portData`. Otherwise, we would get a pointer to the object providing the interface, but we can't be sure about the type of this object.

The method returns if no label lattice is present or if the plot window is not visible and the show button has not been pressed. Otherwise, the contents of the plot window are recomputed from scratch. For this purpose, a dynamic array of arrays called `count` is defined. The array provides a counter for each

material and for each slice of the label lattice. Initially, all counters are set to zero. Afterwards, they are incremented while the voxels are traversed in a nested for-loop.

The actual initialization of the plot window happens subsequently. First, old curves are removed by calling `plot->remData`. Then, for each material, a new curve is added by calling `plot->putData`. Afterwards, `plot->update` is called. If we had not disabled 'auto update' in the constructor, the plot window would have been updated automatically in each call of `putData`. The `putData` method creates a curve with the given name and sets the values. If a curve of the given name exists, the old values are overridden. The method returns a pointer to the curve which in turn can be used to set attributes for the curve individually (see below). Finally, the plot window is popped up and the 'busy' cursor we have activated before is switched off again.

To test the module, first compile the example package. For instructions, see subsection 13.1.5 (Compiling and Debugging). Then load the file `data/tutorials/chocolate-bar-labels.am` from the Avizo root directory. Attach `PlotAreaPerSlice` to it and press the show button. You then should get a result like that shown in Figure 13.11.
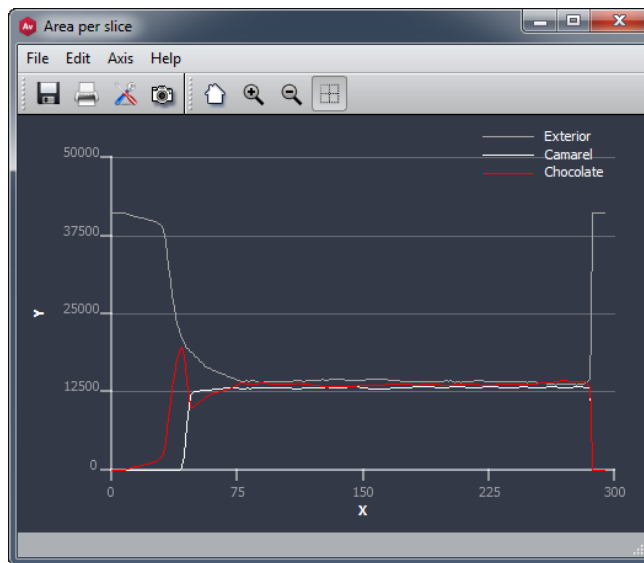
Figure 13.11: Plot produced by sample module *PlotAreaPerSlice*.

13.4.3.2 Additional Features of the Plot API

The 'pointer to curve' objects returned by the `putData` call can be used to access the curve directly, i.e., to manipulate its attributes. The most important attributes of curve objects are:

- Color, represented by a RGB values between 0 and 1. Can be set by calling:
  ```
  curve->setAttr("color", r, g, b);
  ```

- Line width, represented by an integer number. Can be set by calling:
  ```
  curve->setAttr("linewidth", linewidth);
  ```
- Line type, represented by an integer number. Available line types are 0=no line, 1=line, 2=dashed, 3=dash-dotted, and 4=dotted. Can be set by calling:
  ```
  curve->setAttr("linetype", type);
  ```
- Curve type, represented by an integer number. Available curve types are 0=line curve, 1=histogram, 2=marked line, 3=marker. Can be set by calling:
  ```
  curve->setAttr("curvetype", type);
  ```

For each attribute corresponding `getAttr` methods are available. In order to access the axis of the 'easy plot' window, you must call

```
PzAxis* axis = plot->getTheAxis();
```

Don't forget to include the corresponding header file `PzAxis.h`.

The color, line width, and line type attributes of the curves apply to axes as well. Besides this, there are some more methods to change the appearance of axes:

```
// Set the range of the axes
float xmin = 0.0, xmax = 1.0;
float ymin = 0.0, ymax = 1.0;
axis->setMinMax(xmin, xmax, ymin, ymax);

// Set the label of an axis
axis->setAxisLabel(0, "X Axis");
axis->setAxisLabel(1, "Y Axis");
```

If you are not satisfied with the size of the plot window and you don't want to change it using the mouse every time, just call `setSize` right after creating the plot window:

```
plot->setSize(width, height);
```

As you would expect, the methods `getMinMax`, `getAxisLabel` and `getSize` are also available with the same parameter list as their `set` counterparts.

Finally, it is also possible to have a legend or a grid in the plot. In this case, more arguments must be specified in the constructor of `PzEasyPlot`:

```
int withLegend = 1;
int withGrid = 0;
plot = new PzEasyPlot("Area per slice",
    withLegend, withGrid);
```

Like the axis, the legend and the grid are internally represented by separate objects of type `PzLegend` and `PzGrid`. You can access these objects by calling the methods `getTheLegend` and `getTheGrid`. Details about the member methods of these objects are listed in the class reference documentation.

13.4.4   A Compute Module on GPU

Avizo contains everything necessary for execution of GPU programming in CUDA. Unfortunately, a specialized toolkit needs to be installed to develop new modules with GPU compute capabilities. To follow the next steps of this guide, installing CUDA toolkit is necessary.

The latest CUDA Toolkit can be downloaded from this URL: CUDA Toolkit Download page . However, it is more reliable to download the versions used by Avizo. These versions can be found:

- CUDA 5.5 Toolkit on Linux and Mac OSX
- CUDA 6.5 on Windows

Once downloaded, run the installation instructions of the toolkit. The installers usually provide CUDA Toolkit, CUDA Samples and the minimal driver version for CUDA. Please carefully follow the instructions given with the installers and check that CUDA works on the system.

Once the installation is successfully finished, an environment variable must be set so that Avizo knows where to find the CUDA toolkit. This environment variable must be called CUDA_PATH and contain the path to the CUDA Toolkit (Location validated during the installation).

On Mac OSX, an additional installation might be required for CUDA to work (see *system requirements for Mac OSX* for more details). For CUDA programming, you must have a CUDA compatible GPU with an up-to-date driver.

Yet, GPU programming is complex and could be tricky. This tutorial does not intend to be a GPU programming tutorial. This tutorial purpose is to demonstrate how to interface a GPU program within Avizo and requires minimal knowledges in the domain of GPU programming.

In order to learn how to implement a module using GPU programming, we will take a look at a concrete example. In particular, we want to write a module which applies a 2D Gaussian filter on a 3D image, i.e., on an input object of type *HxUniformScalarField3*. The module produces another 3D image as output which is placed in the Project View.

We present you one implementation:

- *Version 1*: a version using CUDA C

You can find the source code of both versions in the example package provided with Avizo XPand Extension, i.e., under src/gaussianfiltercudac in the local Avizo directory. For each version, there are three files: a header file, a source code CPU file and a source code GPU file. Since the names are different, you can compile and execute both versions in parallel.

These modules present how to integrate GPU filters in Avizo. They are as simple as possible to be didactic, so they are not able to treat large amount of data.

Once you have compiled the example module, you can load the file motor.am from Avizo's data/tutorials directory and attach the module to it. These modules can be found in the *Local* category of the data popup menu. Instructions for compiling local packages are provided in subsection 13.1.5 (Compiling and Debugging). If you encounter any error while attaching one of the module or executing it (ex: a dialog complaining about CUDA version, or unknown symbols that prevent a lib to be loaded...), update your graphics driver.

For each module, you can evaluate performance with the TCL command *time*. For the GaussianFilterCudaC module, check the *auto-refresh* box and execute the following command in Avizo console :

```
time {GaussianFilterCudaC fire} 5
```

It will apply 5 times the filter on your data and print the average amount of time required per iteration, in microseconds.

In order to create a new local Avizo directory, please follow the instructions given in subsection 13.2.2. In order to compile the example package, please refer to subsection 13.1.5 (Compiling and Debugging). When a compute module using CUDA API calls is added to an existing package, the LIBS entry of the Package file must be completed depending on the used API. The keyword to add in the list of libraries to link with is:

- CUDA C API: cudac;

For example, if the targeted API is CUDA C, the Package file should contain something like:

```
set LIBS {
    hxplot hxtime hxsurface hxcolor hxfield
    hxcore amiramesh mclib oiv tcl qt cudac
}
```

13.4.4.1 Version 1: Gaussian filter in CUDA C

The main interface currently supported to write CUDA programs is CUDA C. CUDA driver API is not exposed in Avizo.

CUDA C exposes the CUDA programming model as a minimal set of extensions to the C language. These extensions allow programmers to define a kernel as a C function and use some new syntax to specify the grid and block dimension each time the function is called. The first filter is implemented at this level.

The source is divided in two separated parts:

- *a CPU part* with a header file and a C++ file: GaussianFilterCudaC.h and GaussianFilterCudaC.cpp
- *a GPU part* with a CUDA file: Convolve2DCudaC.cu

In order to be didactic, this version is not the optimal implementation for this filter. The GaussianFilterCudaCOptim module is another CUDA C implementation of this Gaussian filter which gives better performance. This implementation is not detailed in this document but it is implemented in the three files called GaussianFilterCudaCOptimized.h, GaussianFilterCudaCOptimized.cpp and Convolve2DCudaCOptimized.cu.

13.4.4.1.1 CPU part
Like most other modules, our compute module consists of a header file containing the class declaration as well as a source file containing the actual code (or the class definition). Let us look at the header file GaussianFilterCudaC.h first:

```
////////////////////////////////////////////////////////////////
//
//  Example of a convolution filter in CUDA C
//
////////////////////////////////////////////////////////////////
#ifndef GAUSSIANFILTERCUDAC_H
#define GAUSSIANFILTERCUDAC_H

#include <hxcore/HxCompModule.h>
```

```
#include <hxcore/HxPortDoIt.h>

#include "api.h"

// Defined in Convolve2DCudaC.cu
extern "C" cudaError
applyFilter(const unsigned char* h_src,
            unsigned char* h_dst,
            const float* h_filter,
            const int* h_dims,
            const int sizeFilter);

class GAUSSIANFILTERCUDAC_API GaussianFilterCudaC : public HxCompModule
{
    // This macro is required for all modules and data objects
    HX_HEADER(GaussianFilterCudaC);

public:
    // To have a Apply button
    virtual void compute();

    // This virtual method will be called when the port changes
    HxPortDoIt portAction;
};

#endif // GAUSSIANFILTERCUDAC_H
```

As usual in C++ code, the file starts with a define statement that prevents the contents of the file from being included multiple times. Then two header files are included. `HxCompModule.h` contains the definition of the base class of our compute module. The other file, `HxPortDoIt.h`, allows the use of the *Apply* button.

The package header file `api.h` is included. This file provides import and export storage-class specifiers for Windows systems. These are encoded in the macro `GAUSSIANFILTERCUDAC_API`. A class declared without this macro will not be accessible from outside the DLL in which it is defined. On Unix systems, the macro is empty and can be omitted.

The `applyFilter` function is declared with the `extern "C"` qualifier. This function is defined in the `Convolve2DCudaC.cu` file and will be explained in the *GPU section*.

In the rest of the header file, the only tasks that remain are to derive a new class from `HxCompModule` and define one member function, namely the overloaded virtual method called `compute`. The `compute` method is called when the module has been created and whenever a change of state occurs on one of the module's input data objects or ports.

The corresponding source file looks like this:

```
///////////////////////////////////////////////////////////
//
//  Example of a convolution filter in CUDA C
//
///////////////////////////////////////////////////////////

#include <QApplication>
```

```
#include <hxcore/HxApplication.h>
#include <hxcore/HxMessage.h>
#include <hxcore/HxProgressInterface.h>
#include <hxfield/HxUniformScalarField3.h>
#include <mclib/McPrimType.h>

#include "CudaCUtils.h"
#include "GaussianFilterCudaC.h"

HX_INIT_CLASS(GaussianFilterCudaC, HxCompModule)

GaussianFilterCudaC::GaussianFilterCudaC()
    : HxCompModule(HxUniformScalarField3::getClassTypeId())
    , portAction(this,
                "action",
                QApplication::translate("GaussianFilterCudaC", "Action"))
{
    portAction.setLabel(0, QApplication::translate("GaussianFilterCudaC", "DoIt"));
}

GaussianFilterCudaC::~GaussianFilterCudaC()
{
}

void
GaussianFilterCudaC::compute()
{
    // Check whether Apply button was hit
    if (!portAction.wasHit())
        return;

    // Access the input data object
    HxUniformScalarField3* input = (HxUniformScalarField3*)portData.getSource();

    // Check whether the input port is connected
    if (!input)
        return;

    // Check data type
    if (input->primType() != McPrimType::MC_UINT8)
        return;

    // Turn into busy state, don't activate the Stop button
    theProgress->startWorkingNoStop(QApplication::translate("GaussianFilterCudaC",
                                                            "Filtering"));

    // Access size of data volume
    const McDim3l& dims = input->lattice().getDims();

    // Check if there is a result which we can reuse
    HxUniformScalarField3* output = (HxUniformScalarField3*)getResult();
```

```
    // Check for proper type
    if (output && !output->isOfType(HxUniformScalarField3::getClassTypeId()))
        output = 0;

    // Check if size and primType still match the current input
    if (output)
    {
        const McDim3l& outdims = output->lattice().getDims();
        if (dims != outdims ||
            input->primType() != output->primType())
            output = 0;
    }

    // If necessary, create a new result data set
    if (!output)
    {
        output = new HxUniformScalarField3(dims, input->primType());
        output->composeLabel(input->getLabel(), "filtered");
    }

    // Output shall have same bounding box as input
    output->coords()->setBoundingBox(input->getBoundingBox());

    // Define Gaussian filter
    int sizeFilter = 3;
    float gaussianFilter[9];
    gaussianFilter[0] = 1 / 16.;
    gaussianFilter[1] = 2 / 16.;
    gaussianFilter[2] = 1 / 16.;
    gaussianFilter[3] = 2 / 16.;
    gaussianFilter[4] = 4 / 16.;
    gaussianFilter[5] = 2 / 16.;
    gaussianFilter[6] = 1 / 16.;
    gaussianFilter[7] = 2 / 16.;
    gaussianFilter[8] = 1 / 16.;

    // Compute filtered image
    cudaError err = applyFilter((unsigned char*)input->lattice().dataPtr(),
                                (unsigned char*)output->lattice().dataPtr(),
                                gaussianFilter,
                                McDim3i(dims),
                                sizeFilter);
    CudaCUtils::cleanupNoFailure(err);

    // Stop progress bar
    theProgress->stopWorking();

    // Register result
    setResult(output);
}
```

Following the include statements, there is the required HX_INIT_CLASS macro for class initialization. Next is the constructor definition which calls the constructor of the base class. There are no special

macros for this, so the usual C++ syntax is used to call the base class constructor and to initialize the class members. The constructor of the base class `HxCompModule` takes the class type of the input data object to which this module can be connected.

The second method we have to implement is the `compute` method. We first retrieve a pointer to our input data object through a member called `portData`. This port is inherited from the base class `HxModule`, i.e., every module has this member. The port is of type `HxConnection` and it is represented as a blue line in the user interface (if connected). This filter only accepts `unsigned char` images, so we must check the data type.

A new result object is to be created. Whenever the range port is changed afterwards, the existing result object should be overridden. The `output` file is of type `HxUniformScalarField3`. The `getResult` method checks whether there is a data set whose master port is connected to the compute module. However, it also may be any other object. Therefore, a run-time type check must be performed by calling the `isOfType` member method of the output object. If the output object is not of type `HxUniformScalarField3`, the variable output will be set to null. Then a check is made whether the output object has the same dimensions and the same primitive data type as the input object. If this test fails, output will also be set to null. At the end, a new result object will only be created if no result exists already or if the existing result does not match the input.

Then the Gaussian filter is defined and the `applyFilter` function is called. The calls to `input->lattice.dataPtr()` and `output->lattice.dataPtr()` allow us to get a pointer to the data values of the `input` and `output` objects. The returned value of this `dataPtr` method is of type `void*`. It must be explicitly cast to the data type to which the field actually belongs. The voxel values itself are arranged without any padding. This means that the index of voxel (`i`, `j`, `k`) is given by (`k * dims[1] + j ) * dims[0] + i`, where `dims[0]` and `dims[1]` denote the number of voxels in the `x` and `y` directions, respectively.

After the device computation, the `cleanupNoFailure` function is called in order to correctly clean up device memory, if there is a CUDA error.

Finally, creating a new data object using the new operator will not automatically make it appear in the Project View. Instead, we must explicitly register it. In a compute module, this can be done by calling the method `setResult`. This method adds a data object to the Project View if it is not already present there. In addition, it connects the object's master port to the compute module itself. Like any other connection, this link will be represented by a blue line in the Project View. The master port of a data object may be connected to a compute module or to an editor. Such a master connection indicates that the data object is controlled by an 'upstream' component, i.e., that its contents may be overridden by the object to which it is connected.

13.4.4.1.2 GPU part In GPU programming, it is important to know where a variable is located in memory (specifically, which section of memory). To make it easier to keep track of this, variable names are prefixed in the CUDA file with an identifier indicating their memory location:

- `h_` if the variable is in host memory
- `d_` if the variable is in device global memory
- `c_` if the variable is in device constant memory

- s_ if the variable is in device shared memory

The CUDA file `Convolve2DCudaC.cu` is the following:

```
///////////////////////////////////////////////////////////////
//
//  Example of a convolution filter in CUDA
//
///////////////////////////////////////////////////////////////
#define BLOCK_SIZE 16
#define BORDER_SIZE 1


// In constant memory
__constant__ float c_filter[9]; // Convolution filter
__constant__ int c_dims[3]; // Size of data volume


// Compute 2D convolution product with a 3*3 filter
// __device__: callable form the device, executed on the device
__device__ unsigned char
convolve2DDrv( int i,
               int j,
               unsigned char s_imageBlock[BLOCK_SIZE + 2 * BORDER_SIZE]
                                         [BLOCK_SIZE + 2 * BORDER_SIZE],
               int imageBlockSize )
{
    int idxFilter = 0;
    float convolutionProduct = 0;
    for ( int ii = - BORDER_SIZE; ii <= BORDER_SIZE; ii++)
    {
        for ( int jj = - BORDER_SIZE; jj <= BORDER_SIZE; jj++)
        {
            convolutionProduct += s_imageBlock[i + ii][j + jj] * c_filter[idxFilter];
            idxFilter++;
        }
    }
    return convolutionProduct;
}


// Return the voxel's value (i, j) of the buffer d_slice
// __device__: callable form the device, executed on the device
int
__device__ getValueDrv( int i, int j, const unsigned char* d_slice )
{
    unsigned char value;
    if ( (i >= 0) && (i < c_dims[0]) && (j >= 0) && (j < c_dims[1]) )
        value = d_slice[j * c_dims[0] + i];
    else
        value = 0;
    return value;
}


// Fill in a variable in shared memory and call convolve2D
// __global__: callable form the host, executed on the device
```

```
__global__ void
applyFilterKernel( unsigned char* d_src, unsigned char* d_dst )
{
    // The threadIdx variable indicates the thread position in the block.
    // The blockIdx variable indicates the block position in the grid.
    // The blockDim variable indicates the dimension of thread block.
    //(i, j, k) repairs the top left corner of the slice in d_src
    int k = blockIdx.y * blockDim.y / c_dims[1];
    int i = blockIdx.x * blockDim.x;
    int j = ( blockIdx.y - k * c_dims[1] / blockDim.y ) * blockDim.y;

    // In shared memory
    __shared__ unsigned char s_imageBlock[BLOCK_SIZE + 2][BLOCK_SIZE + 2];
    int imageBlockSize = BLOCK_SIZE + 2 * BORDER_SIZE;

    // Pointer to the top left corner of the current slice
    const unsigned char* d_slice = d_src + k * c_dims[1] * c_dims[0];

    int iBlock = threadIdx.x + BORDER_SIZE;
    int jBlock = threadIdx.y + BORDER_SIZE;

    // Fill in imageBlock
    // Main data
    s_imageBlock[iBlock][jBlock] =
        getValueDrv(i + threadIdx.x, j + threadIdx.y, d_slice );
    if ( iBlock == 1 )
    {
        // Left border
        s_imageBlock[0][jBlock] =
            getValueDrv( blockIdx.x * blockDim.x - 1, j + threadIdx.y, d_slice );
        s_imageBlock[0][jBlock - 1] =
            getValueDrv( blockIdx.x * blockDim.x - 1, j + threadIdx.y - 1, d_slice );
    }
    else if ( iBlock == BLOCK_SIZE )
    {
        // Right border
        s_imageBlock[imageBlockSize - 1][jBlock] =
            getValueDrv( blockIdx.x * blockDim.x + imageBlockSize - 1,
                         j + threadIdx.y,
                         d_slice );
        s_imageBlock[imageBlockSize - 1][jBlock + 1] =
            getValueDrv( blockIdx.x * blockDim.x + imageBlockSize - 1,
                         j + threadIdx.y + 1,
                         d_slice );
    }

    if ( jBlock == 1 )
    {
        // Top border
        s_imageBlock[iBlock][0]  =
            getValueDrv( i + threadIdx.x, j - 1, d_slice );
        s_imageBlock[iBlock + 1][0] =
            getValueDrv( i + threadIdx.x + 1, j - 1, d_slice );
```

Writing Modules   **809**

```
    }
    else if ( jBlock == BLOCK_SIZE )
    {
        // Botttom border
        s_imageBlock[iBlock][imageBlockSize – 1] =
            getValueDrv( i + threadIdx.x, j + imageBlockSize – 1, d_slice );
        s_imageBlock[iBlock – 1][imageBlockSize – 1] =
            getValueDrv( i + threadIdx.x – 1, j + imageBlockSize – 1, d_slice );
    }
    // All threads should have finished to fill in s_imageBock
    // before begin computation of convolution product
    __syncthreads();

    // Compute convolution product
    int idx = k * c_dims[1] * c_dims[0];
    idx += ( j + threadIdx.y ) * c_dims[0];
    idx += i + threadIdx.x;
    d_dst[idx] = convolve2DDrv( iBlock, jBlock, s_imageBlock, imageBlockSize );
}


// Call the kernel
extern "C" cudaError
applyFilter( const unsigned char* h_src,
             unsigned char* h_dst,
             const float* h_filter,
             const int* h_dims,
             const int sizeFilter )
{
    int dimsTotal = h_dims[0] * h_dims[1] * h_dims[2];

    // Clear error state
    cudaGetLastError();

    // Allocate device memory
    void* d_src = NULL;
    void* d_dst = NULL;
    cudaMalloc( &d_src, dimsTotal * sizeof( unsigned char ) );
    cudaMalloc( &d_dst, dimsTotal * sizeof( unsigned char ) );

    // Copy host memory to device
    cudaMemcpy( d_src, h_src, dimsTotal * sizeof( unsigned char ),
        cudaMemcpyHostToDevice );
    cudaMemcpyToSymbol( c_filter, h_filter, 9 * sizeof( float ) );
    cudaMemcpyToSymbol( c_dims, h_dims, 3 * sizeof( int ) );

    // Execute GPU kernel
    dim3 nThreadsPerBlock( BLOCK_SIZE, BLOCK_SIZE );
    dim3 nBlocks( h_dims[0] / BLOCK_SIZE, h_dims[1] * h_dims[2] / BLOCK_SIZE );
    applyFilterKernel<<<nBlocks, nThreadsPerBlock>>>(
                        static_cast<unsigned char*>(d_src),
                        static_cast<unsigned char*>(d_dst) );
```

```
    // Copy results from device to host
    cudaMemcpy( h_dst, d_dst, dimsTotal * sizeof( unsigned char ),
        cudaMemcpyDeviceToHost );

    // Cleanup devive memory
    cudaFree( d_src );
    cudaFree( d_dst );

    // Cleanup all runtime-related resources
    cudaThreadExit();

    // Return last CUDA error
    return cudaGetLastError();
}
```

This file begins with a definition of two global variables: BLOCK_SIZE will be used to size the grid and blocks and BORDER_SIZE. Then, two variables are declared with the __constant__ qualifier, and they will be placed in constant memory. A variable in constant memory must be initialized by the CPU and it is visible from all threads in read-only. The c_filter will contain the convolution filter and the c_dims variable the size of data volume.

Following these declarations, the convolution product is defined in the convolve2D function. This is a __device__ function, i.e., it must be called from GPU and it is executed on the device. This function computes the 2D convolution product at the point (i, j) with a $3 * 3$ filter.

The getValue function returns the voxel's value (i, j) of a buffer.

The applyFilterKernel function is called for each voxel. It is defined using the __global__ qualifier. This means that the function is called from the host and executed on the device. It is a *kernel* function, so it must have the void return type.

The initial 3D image is divided in several blocks of size BLOCK_SIZE * BLOCK_SIZE. Three indexes i, j and k are declared to repair the top left corner of the current slice in d_src and to define d_slice.

The s_imageBlock buffer is created and placed in shared memory. Then this buffer is filled in and represents a part of the initial image. The size of this buffer is (BLOCK_SIZE + 2) * (BLOCK_SIZE + 2) in order to correctly treat the borders (1 size border around the image because the convolution filter is a $3 * 3$ filter).

The two indexes iBlock and jBlock allow filling in the s_imageBlock buffer.

Before using s_imageBlock for calculation, we must be sure that this buffer is full, so we use the __syncthreads() function. This function acts as a barrier at which all threads in the block must wait before any is allowed to proceed. After this, each thread computes one convolution product using imageBlock.

The last function in this file is the applyFilter function which handles the interaction between the CPU and GPU.

In order to manage CUDA errors, we first call the cudaGetLastError function which returns the last error that has been produced by any of the runtime calls in the same host thread and resets it to CUDA_SUCCESS. Several pointers are declared in order to be allocated in GPU memory. They are allocated in global memory using the cudaMalloc function. The cudaMemcpy function is used to initialized the d_src buffer from host memory. The direction of the transfer is indicated by the

`cudaMemcpyHostToDevice` flag.

The `d_filter`, which was declared at the beginning of this file, is in constant memory, so it is initialized with the `cudaMemcpyToSymbol` function.

The host initiates the execution of the kernel function, `applyFilterKernel`, on the CUDA device. A CUDA device contains individual processing elements, each of which can execute a thread. A number of the processing elements are grouped together to form a block, and a group of blocks constitute a grid. The dimensions of grids and blocks are defined in variables `nThreadsPerBlock` and `nBlocks`. The number of blocks and the number of threads in each block are indicated between `<<<...>>>` following the kernel name. This information is picked up by the Nvidia compiler, *nvcc*, and is used when generating the instructions that start the kernel on the CUDA device. Following that, a list of arguments is passed to `kernel` function.

The `applyFilterKernel` function fills in the `d_dst` buffer. After calculation, we must copy this buffer to host memory. This is done using the `cudaMemcpy` function with the `cudaMemcpyDeviceToHost` flag.

Finally, the global memory is freed using the `cudaFree` function and all runtime-related resources associated with the calling host thread are cleaned up by calling the `cudaThreadExit` function. The last CUDA error is returned using the `cudaGetLastError` function.

13.5   Data Classes

This section provides an overview of the structure of Avizo data classes. Important classes are discussed in more detail. In particular, the following topics will be covered:

- *an introduction to data classes*, including the hierarchy of data classes
- *data on regular grids*, e.g., 3D images with uniform or stacked coords
- *tetrahedral grids*, including data fields defined on such grids
- *hexahedral grids*, including data fields defined on such grids
- *unstructured mixed models*, including data fields defined on such models
- *other issues related to data classes*, including transparent data access

13.5.1   Introduction

A profound knowledge of the Avizo data objects is essential to developers. Data objects occur as input of write routines and almost all modules, and as output of read routines and compute modules. In the previous sections, we already encountered several examples of Avizo data objects such as 3D image data (represented by the class `HxUniformScalarField3`), triangular surfaces (represented by the class `HxSurface`), or colormaps (represented by the class `HxColormap`). Like modules, data objects are instances of C++ classes. All data objects are derived from the common base class `HxData`. Data objects are represented by green icons in Avizo's Project View.

In the following, let us first present an overview of the *hierarchy of data classes*. Afterwards, we will discuss some of the *general concepts* behind it.

13.5.1.1 The Hierarchy of Data Classes

The hierarchy of Avizo data classes roughly looks as follows (derived classes are indented, auxiliary base classes are ignored):

`HxData` *base class of all data objects*
   `HxSpreadSheet` *spreadsheet containing an arbitrary number of rows and columns*
   `HxColormap` *base class of colormaps*
     `HxColormap256` *colormap consisting of discrete RGBA tuples*
   `HxSpatialData` *data objects embedded in 3D space*
     `HxField3` *base class representing fields in 3D space*
       `HxScalarField3` *scalar field (1 component)*
         `HxRegScalarField3` *scalar field with regular coordinates*
           `HxUniformScalarField3` *scalar field with uniform coordinates*
             `HxUniformLabelField3` *material labels with uniform coordinates*
         `HxAnnaScalarField3` *scalar field defined by an analytic expression*
         `HxTetraScalarField3` *scalar field defined on a tetrahedral grid*
         `HxHexaScalarField3` *scalar field defined on a hexahedral grid*
       `HxVectorField3` *vector field (3 components)*
         `HxRegVectorField3` *vector field with regular coordinates*
           `HxUniformVectorField3` *vector field with uniform coordinates*
         `HxAnnaVectorField3` *vector field defined by an analytic expression*
         `HxTetraVectorField3` *vector field defined on a tetrahedral grid*
         `HxHexaVectorField3` *vector field defined on a hexahedral grid*
       `HxColorField3` *color field consisting of RGBA-tuples*
         `HxRegColorField3` *color field with regular coordinates*
           `HxUniformColorField3` *color field with uniform coordinates*
       `HxRegField3` *other n-component field with regular coordinates*
       `HxTetraField3` *other n-component field defined on a tetrahedral grid*
       `HxHexaField3` *other n-component field defined on a hexahedral grid*
     `HxVertexSet` *data objects providing a set of discrete vertices*
       `HxSurface` *represents a triangular surface*
       `HxTetraGrid` *represents a tetrahedral grid*
       `HxHexaGrid` *represents a hexahedral grid*
     `HxSurfaceField` *base class for fields defined on triangular surfaces*
       `HxSurfaceScalarField` *scalar field defined on a surface (1 component)*
       `HxSurfaceVectorField` *vector field defined on a surface (3 components)*
       `HxSurfaceField` *other n-component field defined on a surface*

Note that you can find an in-depth description of every class in the online reference documentation: `share/devrefAvizo/Avizo.chm` or `share/devrefAvizo/index.html` in the Avizo root directory. The reference documentation not only covers data objects, but all classes provided with Avizo XPand Extension. As you already know, these classes are arranged in packages. For example, all data classes derived from `HxField3` are located in package `hxfield`, and all classes

related to triangular surfaces are located in package `hxsurface`.

13.5.1.2 Remarks About the Class Hierarchy

All data classes are derived from the base class `HxData`. This class in turn is derived from `HxObject`, the base class of all objects that can be put into the Avizo Project View. The class `HxData` adds support for reading and writing data objects, and it provides the variable `parameters` of type `HxParamBundle`. This variable can be used to annotate a data object by an arbitrary number of nested parameters. The parameters of any data object can be edited interactively using the parameter editor described in the User's Guide.

We observe that the majority of data classes are derived from `HxSpatialData`. This is the base class of all data objects which are embedded in 3D space as opposed for example to colormaps. `HxSpatialData` adds support for user-defined affine transformations, i.e., translations, rotations, and scaling. For details refer to subsection 13.5.6.2. It also provides the virtual method `getBoundingBox` which is redefined by all derived classes. Two important child classes of `HxSpatialData` are `HxField3` and `HxVertexSet`.

`HxVertexSet` is the base class of all data objects that are defined on an unstructured set of vertices in 3D space, like surfaces or tetrahedral grids. The class provides methods to apply a user-defined affine transformation to all vertices of the object, or modify the point coordinates in some other way.

`HxField3` is the base class of data fields defined on a 3D-domain, like 3D scalar fields or 3D vector fields. `HxField3` defines an efficient procedural interface to evaluate the field at an arbitrary 3D point within the domain, independent of whether the latter is a regular grid, a tetrahedral grid, or something else. The procedural interface is described in more detail in subsection 13.5.6.1.

Looking at the inheritance hierarchy again, we observe that a high level distinction is made between fields returning a different number of data values. For example, all 3D scalar fields are derived from a common base class `HxScalarField3`, and all 3D vector fields are derived from a common base class `HxVectorField3`. The reason for this structure is that many modules depend on the data dimensionality of a field only, not on the internal representation of the data. For example, a module for visualizing a flow field by means of particle tracing can be written to accept any object of type `HxVectorField3` as input. It then automatically operates on all derived vector fields, regardless of the type of grid they are defined on.

On the other hand, it is often useful to treat the number of data variables of a field as a dynamic quantity and to distinguish between the type of grid a field is defined on. For example, we may wish to have a common base class of fields defined on a regular grid and derived classes for regular scalar or vector fields. Since this structure and the one sketched above are very hard to incorporate into a common class graph, even if multiple inheritance were used, another concept has been chosen in Avizo, namely *interfaces*. Interfaces were first introduced by the Java programming language. They allow the programmer to take advantage of common properties of classes that are not related by inheritance.

In Avizo, interfaces can be implemented as class members, or as additional base classes. In the first case, a data class *contains* an interface class, while in the second case, it is *derived* from `HxInterface`. Important interface classes are `HxLattice3`, `HxTetraData`, and `HxHexaData`, which are members of fields defined on regular, tetrahedral, and hexahedral grids, respectively. Another example is `HxLabelLattice3`, which is a member of

`HxUniformLabelField3`, as well as `HxStackedLabelField3`. In subsection 13.4.3.1, we have already presented an example of how to use this interface in order to write a module which operates on any label image, regardless of the actual coordinate type.

13.5.2 Data on Regular Grids

Fields defined on a regular grid occur in many different applications. For example, 3D image volumes fall into this category. The term 'regular' means that the nodes of the grid are arranged as a regular 3D array, i.e., every node can be addressed by an index triple (i,j,k). A regular field can be characterized by three major properties: the coordinate type, the number of data components, and the primitive component data type (for example `short` or `float`).

In the *class hierarchy*, a major distinction is made between the number of data components of a field. For example, there is a class `HxRegScalarField3` representing (one-component) scalar fields defined on a regular grid. This class is derived from the general base class `HxScalarField3`. Similar classes exist for (three-component) vector fields, complex scalar field, and complex vector fields defined on regular grids. Fields not falling into one of these categories, i.e., fields defined on regular grids with a different number of data components, are represented by the class `HxRegField3` which is directly derived from `HxField3`. Moreover, there are separate subclasses for the most relevant combinations of the number of data components and the coordinates type, like `HxStackedScalarField3` or `HxUniformVectorField3`. All regular data classes provide a member variable `lattice` of type `HxLattice3`. This variable is an *interface*. It can be used to access data fields with a different number of components in a transparent way.

The *lattice interface* is discussed in more detail below. We then present an overview of all supported *coordinate types*. Afterwards, two more types of data fields defined on regular coordinates are discussed, namely *label images* and *color fields*.

Note that all these fields can be evaluated without regard to the actual coordinate type or the primitive data type by means of Avizo's procedural interface for 3D fields (see subsection 13.5.6.1).

13.5.2.1 The Lattice Interface

The actual data of any regular 3D field is stored in a member variable `lattice` of type `HxLattice3`. This variable essentially represents a dynamic 3D array of n-component vectors. The number of vector components as well as the primitive data type are subject to change, i.e., a data object of type `HxLattice3` can be re-initialized to hold a different number of components of different primitive data type. However, a lattice contained in an object of type `HxRegScalarField3` always consists of 1-component vectors, while a lattice contained in an object of type `HxRegVectorField3` always consists of 3-component vectors. In addition, the coordinates of the field are stored in a separate coordinate object that is also referenced by the lattice.

Accessing the Data   To learn what kind of methods are provided by the lattice class, please refer to the online reference documentation or directly inspect the header file `HxLattice3.h` located in package `hxfield`. At this point, we just present a short example which shows how the dimensionality of the lattice, the number of data components, and the primitive data type can be queried. The primitive

data type is encoded by the class `McPrimType` defined in package `mclib`. In particular, here are some of the following data types supported by Avizo:

- `McPrimType::mc_uint8` (8-bit unsigned bytes)
- `McPrimType::mc_int16` (16-bit signed shorts)
- `McPrimType::mc_uint16` (16-bit unsigned shorts)
- `McPrimType::mc_int32` (32-bit signed integers)
- `McPrimType::mc_float` (32-bit floats)
- `McPrimType::mc_double` (64-bit doubles)
- ...

Regardless of the actual type of the lattice data values, the pointer to the data array is returned as `void*`. The return value must be explicitly cast to a pointer of the correct type. This is illustrated in the following example where we compute the maximum value of all data components of a lattice. Note that the data values are stored one after another without any padding. The first index runs fastest.

```
HxLattice3& lattice = field->lattice();
const int* dims = lattice.dims();
int nDataVar = lattice.nDataVar();

switch (lattice.primType()) {
case McPrimType::mc_uint8: {
    unsigned char* data = (unsigned char*) lattice.dataPtr();
    unsigned char max = data[0];
    for (int k=0; k<dims[2]; k++)
        for (int j=0; j<dims[1]; j++)
            for (int i=0; i<dims[0]; i++)
                for (int n=0; n<nDataVar; n++) {
                    int idx =
                        nDataVar*((k*dims[1]+j)*dims[0]+i)+n;
                    if (data[idx]>max)
                        max = data[idx];
                }
    theMsg->printf("Max value is %d", max);
    } break;

case McPrimType::mc_int16: {
    short* data = (short*) lattice.dataPtr();
    short max = data[0];
    for (int k=0; k<dims[2]; k++)
        for (int j=0; j<dims[1]; j++)
            for (int i=0; i<dims[0]; i++)
                for (int n=0; n<nDataVar; n++) {
                    int idx =
                        nDataVar*((k*dims[1]+j)*dims[0]+i)+n;
                    if (data[idx]>max)
                        max = data[idx];
                }
    theMsg->printf("Max value is %d", max);
    } break;
```

```
    ...

    }
```

As a tip, note that the processing of different primitive data types can often be simplified by defining appropriate template functions locally. In the case of our example, such a template function may look like this:

```
template<class T>
void getmax(T* data, const int* dims, int nDataVar)
{
    T max = data[0];
    for (int k=0; k<dims[2]; k++)
        for (int j=0; j<dims[1]; j++)
            for (int i=0; i<dims[0]; i++)
                for (int n=0; n<nDataVar; n++) {
                    int idx =
                        nDataVar*((k*dims[1]+j)*dims[0]+i)+n;
                    if (data[idx]>max)
                        max = data[idx];
                }
    theMsg->printf("Max value is %d", max);
}
```

Using this template function, the above switch statement looks as follows:

```
    switch (lattice.primType()) {
    case McPrimType::mc_uint8:
        getmax((unsigned char*)lattice.dataPtr(),dims,nDataVar);
        break;
    case McPrimType::mc_int16:
        getmax((short*)lattice.dataPtr(),dims,nDataVar);
        break;

    ...

    }
```

Though less efficient, another possibility for handling different primitive data types is to use one of the methods `eval`, `set`, `getData`, or `putData`. These methods always involve a cast to `float`, if the primitive data type of the field requires it.

Accessing the Lattice Interface   Imagine you want to write a module which operates on any kind of regular field, i.e., on objects of type `HxRegScalarField3`, `HxRegVectorField3`, and so on. One way to achieve this would be to configure the input port of the module so that it can be connected to all possible regular field input objects. This can be done by calling the method `portData.addType()` in the module's constructor multiple times with the required class type IDs. In addition, all input types must be listed in the package resource file. This can be done by specifying a blank-separated list of types as the argument of the `-primary` option of the `module`

command. In the compute method of the module, the actual type of the input must be queried, then the input pointer must be cast to the required type before a pointer to the lattice member of the object can be stored.

Of course, this approach is very tedious. A much simpler approach is to make use of the fact that the lattice member of a regular field is an interface. Instead of the name of a real data class, the class type ID of `HxLattice3` may be used to specify to what kind of input object a module may be connected to. In fact, if this is done, any data object providing the lattice interface will be considered as a valid input. In order to access the lattice interface of the input object, the following statement must be used in the module's compute method (also check subsection 13.4.3.1 for an example of how to deal with interfaces):

```
HxLattice3* lattice = (HxLattice3*)
    portData.source(HxLattice3::getClassTypeId());
```

Creating a Field From an Existing Lattice   When working with lattices, we may want to deposit a new lattice in the Project View, for example, as the result of a compute module. However, since `HxLattice3` is not an Avizo data class, this is not possible. Instead we must create a suitable field object of which the lattice is a member. For this purpose, the class `HxLattice3` provides a static method `create` which creates a regular field and puts an existing lattice into it. If the lattice contains one data component, a scalar field will be created; if it contains three components, a vector field will be created, and so on. The resulting field may then be used as the result of a compute module. Note that the lattice must not be deleted once it has been put into a field object. The concept is illustrated by the following example:

```
HxLattice3* lattice = new HxLattice3(dims, nDataVar,
    primType, otherLattice->coords()->duplicate());

...

HxField3* field = HxLattice3::create(lattice);
theObjectPool->addObject(field);
```

13.5.2.2   Regular Coordinate Types

Currently four different coordinate types are supported for regular fields, namely uniform coordinates, stacked coordinates, rectilinear coordinates, and curvilinear coordinates. The coordinate types are distinguished by way of the enumeration data type `HxCoordType`. The coordinates themselves are stored in a separate utility class of type `HxCoord3` which is referenced by the lattice member of a regular field. For each coordinate type, there is a corresponding subclass of `HxCoord3`.

As already mentioned in the introduction, for some important cases, there are special subclasses of a regular field dedicated to a particular coordinate type. Examples are `HxStackedScalarField3` (derived from `HxRegScalarField3`) or `HxUniformVectorField3` (derived from `HxRegVectorField3`). If such special classes do not exist, the regular base class should be used instead. In this case, the coordinate type must be checked dynamically and the pointer to the coordinate object has to be down-cast explicitly before it can be used. This is illustrated in the following example:

```
HxCoord3* coord = field->lattice.coords();

if (coord->coordType() == c_rectilinear) {
    HxRectilinearCoord3* rectcoord =
        (HxRectilinearCoord3*) coord;
    ...
}
```

Uniform Coordinates   Uniform coordinates are the simplest form of regular coordinates. All grid cells are axis-aligned and of equal size. In order to compute the position of a particular grid node, it is sufficient to know the number of cells in each direction as well as the bounding box of the grid. Uniform coordinates are represented by the class `HxUniformCoord3`. This class provides a method `bbox` which returns a pointer to an array of six floats describing the bounding box of the grid. The six numbers represent the minimum x-value, the maximum x-value, the minimum y-value, the maximum y-value, the minimum z-value, and the maximum z-value in that order. Note that the values refer to grid nodes, i.e., to the corner of a grid cell or to the center of a voxel. In order to compute the width of a voxel, you should use code like this:

```
const int* dims = uniformcoords->dims();
const float* bbox = uniformcoords->bbox();
float width = (dims[0]>1) ? (bbox[1]-bbox[0])/(dims[0]-1):0;
```

Stacked Coordinates   Stacked coordinates are used to describe a stack of uniform 2D slices with variable slice distance. They are represented by the class `HxStackedCoord3`. This class provides a method `bboxXY` which returns a pointer to an array of four floats describing the bounding box of a 2D slice. In addition, the method `coordZ` returns a pointer to an array containing the z-coordinate of each 2D slice.

Rectilinear Coordinates   Same as for uniform or stacked coordinates, in the case of rectilinear coordinates the grid cells are aligned to the axes, but the grid spacing may vary from cell to cell in each direction. Rectilinear coordinates are represented by the class `HxRectilinearCoord3`. This class provides three methods, `coordX`, `coordY`, and `coordZ`, returning pointers to the arrays of x-, y-, and z-coordinates, respectively.

Curvilinear Coordinates   In the case of curvilinear coordinates, the position of each grid node is stored explicitly as a 3D vector of floats. A single grid cell need not to be axis-aligned anymore. An example of a 2D curvilinear grid is shown in Figure 13.12. Curvilinear coordinates are represented by the class `HxCurvilinearCoord3`. This class provides a method `pos` which can be used to query the position of a grid node indicated by an index triple (i,j,k). Alternatively, a pointer to the coordinate values may be obtained by calling the method `coords`. The coordinate vectors are stored one after another without padding and with index i running fastest. Here is an example:

```
const int* dims = curvilinearcoords->dims();
```

```
const float* coords = curvilinearcoords->coords();

// Position of grid node (i,j,k)
float x = coords[3*((k*dims[1]+j)*dims[0]+i)];
float y = coords[3*((k*dims[1]+j)*dims[0]+i)+1];
float z = coords[3*((k*dims[1]+j)*dims[0]+i)+2];
```
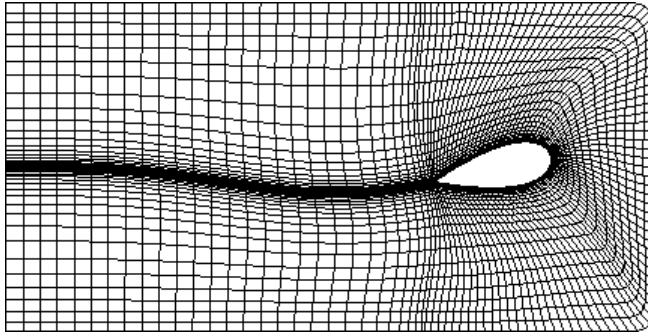

Figure 13.12: Example of a 2D grid with curvilinear coordinates.

13.5.2.3 Label Images and the Label Lattice Interface

Label fields are used to store the results of an image segmentation process. Essentially, at each voxel, a number is stored indicating to which material the voxel belongs. Consequently, label images can be considered scalar fields. In fact, currently there are two different types of label images, one for uniform coordinates (represented by class `HxUniformLabelField3` derived from `HxUniformScalarField3`) and one for stacked coordinates (represented by class `HxStackedLabelField3` derived from class `HxStackedScalarField3`). Since the two types are not derived from a common base class, a special-purpose interface called `HxLabelLattice3` is provided. In fact, this interface is in turn derived from `HxLattice3`. It replaces the standard lattice variable of ordinary regular fields (see subsection 13.5.2.1).

The primitive data type of a label image can be `McPrimType::mc_uint8`, `McPrimType::mc_uint16` or `McPrimType::mc_int32`. In addition to the standard lattice interface, the label lattice interface also provides access to the label field's materials. Materials are stored in a special parameter subdirectory of the underlying data object. While discussing the plot API, we already encountered an example of how to interpret the materials of a label image (see subsection 13.4.3.1). Note that whenever a new label is introduced, a new entry should also be put into the material list. Existing materials are marked so that they can not be removed from the material list (this would corrupt the labeling). In order to remove obsolete materials, call the method `removeEmptyMaterials` of `HxLabelLattice3`.

In addition to the labels, special weights can be stored in a label lattice. These weights are used to achieve sub-voxel accuracy when reconstructing 3D surfaces from the segmentation results. A pointer

to the weights can be obtained by calling `getWeights` or `getWeights2` of the label lattice. For more details about `HxLabelLattice3`, please refer to the online class documentation.

13.5.2.4 Color Fields

Color fields are yet another type of regular fields. They consist of 4-component RGBA-byte-tuples and are represented by the class `HxRegColorField3` derived from `HxColorField3`. The latter class is closely related to `HxScalarField3` or `HxVectorField3`, see the overview on data class inheritance presented in subsection 13.5.1.1. For color fields with uniform coordinates, there is a special subclass `HxUniformColorField3`. Like any other regular fields, color fields provide a member `lattice` which can be used to access the data in a transparent way.

13.5.3 Unstructured Tetrahedral Data

Another important type of data refers to fields defined on unstructured tetrahedral grids. Such grids are often used in finite element simulations (FEM). In Avizo, tetrahedral grids and data fields defined on such grids are implemented by two different classes or groups of classes and are also distinguished in the user interface by different icons. The reason is that by separating grid and data there is no need for replicating the grid in case many fields are defined on the same grid, a case that occurs frequently in practice.

In the following two sections, we introduce the *grid class* `HxTetraGrid` before discussing the corresponding *field classes* and the interface `HxTetraData`.

13.5.3.1 Tetrahedral Grids

Tetrahedral grids in Avizo are implemented by the class `HxTetraGrid` and its base class *Tetra Grid*. Looking at the reference documentation of *Tetra Grid*, we observe that a tetrahedral grid essentially consists of a number of dynamic arrays such as `points`, `tetras`, or `materialIds`.

- The `points` array is a list of all 3D points contained in the grid. A single point is stored as an element of type `McVec3<float>`. This class has the same layout as the Open Inventor class `SbVec3f`. Thus, a pointer to `McVec3<float>` can be cast to a pointer to `SbVec3f` and vice versa.

- The `tetras` array describes the actual tetrahedra. For each tetrahedron, the indices of the four points and the indices of the four triangles it consists of are stored. The numbering of the points and triangles is shown in Figure 13.13. In particular, the fourth point is located above of the triangle defined by the first three points. Triangle number *i* is located opposite to point number *i*.

- The `materialIds` array contains 8-bit labels that assign a 'material' identifier to every tetrahedron. For example, this is used in tetrahedral grids generated from segmented image data to distinguish between different image segments corresponding to different material components of physical objects represented by the (3D) image data Like in the case of label images or surfaces, the set of possible material values is stored in the parameter list of the grid data object.
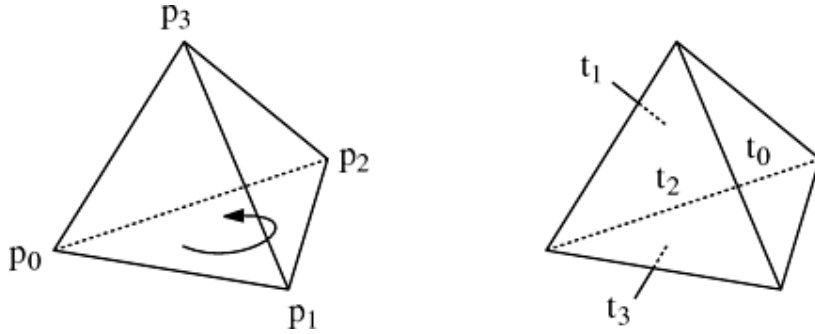
Figure 13.13: Numbering of points in a tetrahedron with positive volume (left). Numbering of the corresponding triangles (right).

The three arrays, `points`, `tetras`, and `materialIds`, must be provided by the 'user'. The triangles of the grid are stored in an additional array called `triangles`. This array can be constructed automatically by calling the member method `createTriangles2`. This method computes the triangles from scratch and sets the triangle indices of all tetrahedra defined in `tetras`.

The `triangles` array also provides a way for accessing neighboring tetrahedra. Among other information (see reference documentation) stored for each triangle, the indices of the two tetrahedra it belongs to are available. In the case of boundary triangles, one of these indices is -1. Therefore, in order to get the index of a neighboring tetrahedron you can use the following code:

```
// Find tetra adjacent to tetra n at face 0:
int triangle = grid->tetras[n].triangles[0];
otherTetra = grid->triangles[triangle].tetras[0];
if (otherTetra == n)
    otherTetra = grid->triangles[triangle].tetras[1];
if (otherTetra == -1) {
    // No neighboring tetra, boundary face
    ...
}
```

Note that it is possible to define a grid with duplicated vertices, i.e., with vertices having exactly the same coordinates. This is useful to represent discontinuous data fields. The method *createTriangles2* checks for such duplicated nodes and correctly creates a single triangle between two geometrically adjacent tetrahedra, even if these tetrahedra refer to duplicated points.

Optionally, the edges of a grid can be computed in addition to its points triangles, and tetrahedra by calling *createEdges*. The edges are stored in an array called `edges` and another array `edgesPerTetra` is used in order to store the indices of the six edges of a tetrahedron.

Moreover, the class *Tetra Grid* provides additional optional arrays, for example to store a dynamic list of the indices of all tetrahedra adjacent to a particular point (`tetrasPerPoint`). This and other information is primarily used for internal purposes, for example to facilitate editing and smoothing of tetrahedral grids.

13.5.3.2 Data Defined on Tetrahedral Grids

In most applications, you will not only have to deal with a single tetrahedral grid, but also with data fields defined on it, for example scalar fields (e.g., temperature) or vector fields (e.g., flow velocity). Avizo provides special classes for these data modalities, namely `HxTetraScalarField3`, `HxTetraVectorField3`, `HxTetraComplexScalarField3`, `HxTetraComplexVectorField3`, and `HxTetraField3` (see class hierarchy in subsection 13.5.1.1).

Like in the case of regular data fields, the actual information is stored a private member and accessible with the *tetraData()* method, which is of type `HxTetraData`. Like the corresponding member type `HxLattice3` for regular data, `HxTetraData` is an `interface`, i.e., derived from `HxInterface`. The *tetraData()* method provides transparent access to data fields defined on tetrahedral grids regardless of the actual number of data components of the field. In order to access that interface without knowing the actual type of input object within a module, you may use the following statement:

```
HxTetraData* data = (HxTetraData*)
    portData.source(HxTetraData::getClassTypeId());
if (!data) return;
```

Data on tetrahedral grids must always be of type `float`. The data values may be stored in three different ways, indicated by the encoding type as defined in `HxTetraData`:

- PER_TETRA: One data vector is stored for each tetrahedron. The data are assumed to be constant inside the tetrahedron.
- PER_VERTEX: One data vector is stored for each vertex of the grid. The data are interpolated linearly inside a tetrahedron.
- PER_TETRA_VERTEX: Four separate data vectors are stored for each tetrahedron. The data are also interpolated linearly.
- PER_VERTEX_AND_EDGE: One data vector is stored for each vertex and edge of the grid.

This last encoding scheme is useful for modeling discontinuous fields. In order to evaluate a field at an arbitrary location in a transparent way, Avizo's procedural data interface should be used. This interface is described in subsection 13.5.6.1.

Like `HxLattice3`, the class `HxTetraData` provides a static method `create` which can be used to create a matching data field, e.g., an object of type `HxTetraScalarField3`, from an existing instance of `HxTetraData`. The `HxTetraData` object will not be copied but will be directly put into the field object. Therefore, it may not be deleted afterwards. Also see subsection 13.5.2.1.

13.5.4 Unstructured Hexahedral Data

In an unstructured hexahedral grid, the grid cells are defined explicitly by specifying all the points in the cell. This is in contrast to regular hexahedral grids where the grid cells are arranged in a regular 3D array and thus are defined implicitly. The implementation of hexahedral grids is very similar to tetrahedral grids as described in the previous section. There are separate classes for the grid itself and for data fields defined on a hexahedral grid.

In the following two sections, we introduce the *grid class* `HxHexaGrid` before discussing the corresponding *field classes* and the interface `HxHexaData`.

13.5.4.1 Hexahedral Grids

Hexahedral grids in Avizo are implemented by the class `HxHexaGrid` and its base class *Hexa Grid*. Looking at the reference documentation of *Hexa Grid*, we observe that a hexahedral grid essentially consists of a number of dynamic arrays such as `points`, `hexas`, or `materialIds`.

- The `points` array is a list of all 3D points contained in the grid. A single point is stored as an element of type `McVec3<float>`. This class has the same layout as the Open Inventor class `SbVec3f`. Thus, a pointer to `McVec3<float>` can be cast to a pointer to `SbVec3f` and vice versa.

- The `hexas` array describes the actual hexahedra. For each hexahedron, the indices of the eight points and the indices of the six faces it consists of are stored. The numbering of the points is shown in Figure 13.14. Degenerate cells such as prisms or tetrahedra may be defined by choosing the same index for neighboring points.

- The `materialIds` array contains 8-bit labels, which assign a material identifier to every hexahedron. Like the case of label images or surfaces, the set of possible material identifiers is stored in the parameter list of the grid data object.
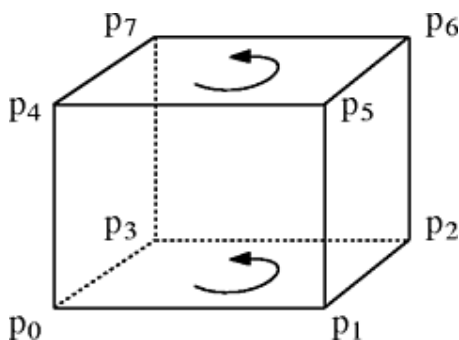
Figure 13.14: Numbering of points in a hexadron with positive volume.

The three arrays, `points`, `hexas`, and `materialIds`, must be provided by the user. The faces of the grid are stored in an additional array called `faces`. This array can be constructed automatically by calling the member method `createFaces`. This method computes the faces from scratch and sets the face indices of all hexahedra defined in `hexas`.

Note that, in contrast to tetrahedral grids, in a hexahedral grid degenerate cells are allowed, i.e., cells where neighboring corners in a cell coincide. In this way, grids with mixed cell types can be defined. The faces of a hexahedron are stored in a small dynamic array called `faces`. For a degenerate cell, this array contains less than six faces.

Also note that, although non-conformal grids are allowed, i.e., grids with hanging nodes on edges and faces, currently the method *createFaces* does not detect the connectivity between neighboring hexahedra sharing less than four points. Thus, faces between such cells are considered to be external cells.

13.5.4.2 Data Defined on Hexahedral Grids

In most applications, you will not only have to deal with a single hexahedral grid, but also with data fields defined on it, for example scalar fields (e.g., temperature) or vector fields (e.g., flow velocity). Avizo provides special classes for these data modalities, namely HxHexaScalarField3, HxHexaVectorField3, HxHexaComplexScalarField3, HxHexaComplexVectorField3, and HxHexaField3 (see class hierarchy in subsection 13.5.1.1).

As for fields defined on tetrahedral grids, the actual information is stored in a private member and accessible with the *hexaData()* method, which is of type HxHexaData. HxHexaData is a so-called interface, i.e., derived from HxInterface. The *hexaData()* method provides transparent access to data fields defined on hexahedral grids regardless of the actual number of data components the field has. In order to access the interface without knowing the actual type of input object within a module, you may use the following statement:

```
HxHexaData* data = (HxHexaData*)
    portData.source(HxHexaData::getClassTypeId());
if (!data) return;
```

Data on hexahedral grids must always be of type float. The data values may be stored in three different ways, indicated by the encoding type defined in HxHexaData:

- PER_HEXA: One data vector is stored for each hexahedron. The data are assumed to be constant inside the hexahedron.
- PER_VERTEX: One data vector is stored for each vertex of the grid. The data are interpolated trilinearly inside a hexahedron.
- PER_HEXA_VERTEX: Eight separate data vectors are stored for each hexahedron. The data are also interpolated trilinearly.

The last encoding scheme is useful for modeling discontinuous fields. In order to evaluate a field at an arbitrary location in a transparent way, Avizo's procedural data interface should be used. This interface is described in subsection 13.5.6.1.

Like HxLattice3, the class HxHexaData provides a static method create which can be used to create a matching data field, e.g., an object of type HxHexaScalarField3, from an existing instance of HxHexaData. The HxHexaData object will not be copied but will be directly put into the field object. Therefore, it may not be deleted afterwards. Also see subsection 13.5.2.1.

13.5.5 Unstructured Mixed Models

Most CAE and CFD simulations are done on unstructured grids. Avizo XWind Extension is the software suite including the Avizo Lite Edition feature-set and all its extensions for analyzing, visualizing,

and presenting numerical solutions from CAE and CFD simulations. Avizo XWind Extension has its own support for unstructured mixed meshes made up of tetrahedra, hexahedra, pyramids and wedges. In the following two sections, we introduce the *model class* `HxUnstructuredModel` before discussing the corresponding *field classes* `HxUnstructuredDataSet`.

13.5.5.1 Unstructured Models

Unstructured models are only available in Avizo XWind Extension and are implemented by the class `HxUnstructuredModel` and its base class *Unstructured Model*.
A model contains:

- the mesh of the domain under study, with 2D or 3D cells depending on the dimension of the model,
- the boundaries of the domain,
- the different regions the domain might be composed of,
- the different materials the domain might be made of.

Depending whether the dimension of the model is 2D or 3D, the `Type` of the model will be `VOLUME` or `SURFACE`. As a consequence, the mesh associated with the model should be a `HxUnstructuredVolumeMesh` or a `HxUnstructuredSurfaceMesh`. The mesh has to be associated with the model through the `setMesh` method.
The unstructured mesh is characterized by its geometry and its topology. The geometry (`HxUnstructuredGeometry`) contains the coordinates of the mesh nodes, stored as elements of type `MbVec3d`. The topology (`HxUnstructuredVolumeTopology` or `HxUnstructuredSurfaceTopology`) contains the mesh cells, defined by the list of their nodes index (as stored in the geometry). In case the model is 3D, the cells are elements of type `HxVolumeCell`. Supported cell types are tetrahedron (`HxTetrahedronCell`), pyramid (`HxPyramidCell`), wedge (`HxWedgeCell`) and hexahedron (`HxHexahedronCell`). In case the model is 2D, the cells are elements of type `HxSurfaceCell`. Supported cell types are triangle (`HxTriangleCell`) and quadrangle (`HxQuadrangleCell`).
In the 3D case, the mesh might contain 2D cells. This set of cells is called *boundaries*, referring to the CFD field but can as well define shell or membrane elements in the CAE field. `HxUnstructuredSurfaceBoundary` inherits from `HxUnstructuredSurfaceMesh` and so is as well defined by its geometry and topology.
If the model is composed of several regions, those regions can be identified by a name, an index and a color (see `HxModelParts`) and assigned to each cell of the mesh through the `assignCellParts` method. The same is true for the materials (`HxCellMaterials`, `assignCellMaterials`).

13.5.5.2 Data Defined on Unstructured Models

Data defined on unstructured models are implemented by the class `HxUnstructuredModelDataSet`. The fields can be of type `SCALARSET`, `VECTORSET`, `TENSORSET` or `SYMTENSORSET` (symmetric tensor set). Depending on the type, data are stored as `HxUnstructuredScalarSet`, `HxUnstructuredVectorSet`, `HxUnstructuredTensorSet` or `HxUnstructuredSymTensorSet`, all inheriting from

`HxUnstructuredDataSet`. The data has to be associated with the model data set through the `setDataSet` method.

Two data bindings are supported: PER_NODE (the data field values are stored at the mesh nodes) and PER_CELL (the data field values are stored at the mesh cells). Set the binding of the `HxUnstructuredDataSet` with the `setBinding` method.

In the 3D case, if boundaries exist, data can be stored in the same way and associated with the model data set using the `getBoundaries` method. The two bindings are available on boundaries.

Finally, the model data set is connected to the unstructured model it is defined on.

```
HxUnstructuredModel* model = new HxUnstructuredModel;
// Define the unstructured model
...
HxUnstructuredModelDataSet* modelDataSet = new HxUnstructuredModelDataSet;
HxUnstructuredScalarSet* scalarSet = new HxUnstructuredScalarSet;
// Assign the scalar set
...
scalarSet->setBinding(HxUnstructuredDataSet::PER_NODE);
modelDataSet->setDataSet(scalarSet);
...
modelDataSet->portGrid.connect(model);
...
```

13.5.6 Other Issues Related to Data Classes

The following topics will be covered in this section:

- *Avizo's procedural interface for evaluating 3D fields*
- *coordinate systems and transformations of spatial data objects*
- *defining parameters and materials in data objects*

13.5.6.1 Procedural Interface for 3D Fields

The internal representation of a data field depends largely on whether the field is defined on a regular, tetrahedral, or hexahedral grid. There are even data types such as `HxAnnaScalarField3` or `HxAnnaVectorField3` for fields that are defined by an analytical mathematical expression. To allow for writing a module which operates on any scalar field without having to bother about the particular data representation, a transparent interface is needed. One could think of a function like

```
float value = field->evaluate(x,y,z);
```

For the sake of efficiency, a slightly different interface is used in Avizo. Evaluating a field defined on tetrahedral grid at an arbitrary location usually involves a global search to detect the tetrahedron, which contains that point. The situation is similar for other grid types. In most algorithms, however, the field is typically evaluated at points not far from each other, e.g., when integrating a field line. To take advantage of this fact, the concept of an abstract `Location` class has been introduced. A `Location` describes a point in 3D space. Depending on the underlying grid, `Location` may keep track of additional information such as the current grid cell number. The `Location` class provides

two different search strategies, a global one and a local one. In this way performance can be improved significantly. Here is an example of how to use a `Location` class:

```
float pos[3];
float value;
...
HxLocation3* location = field->createLocation();
if (location->set(pos))
    field->eval(location, &value);
...
if (location->move(pos))
    field->eval(location, &value);
...
delete location;
```

First a location is created by calling the virtual method `createLocation` of the field to be evaluated. The two methods, `location->set(pos)` and `location->move(pos)`, both take an array of three floats as argument, which describe a point in 3D space. The `set` method always performs a global search in order to locate the point. In contrast, `move` first tries to locate the new point using a local search strategy starting from the previous position. You should call `move` when the new position differs only slightly from the previous one. Both `set` and `move` may return 0 in order to indicate that the requested point could not be located, i.e., that it is not contained in any grid cell.

In order to locate the field at a particular location, `field->eval( location, &value)` is called. The result is written to the variable pointed to by the second argument. Internally the `eval` method does two things. First it interpolates the field values, for example, using the values at the corners of the cell in which the current point is contained. Secondly, it converts the result to a float value, if the field is represented internally by a different primitive data type.

13.5.6.2 Transformations of Spatial Data Objects

In Avizo, all data objects which are embedded in 3D space are derived from the class `HxSpatialData` defined in the subdirectory `hxcore` (see class hierarchy in subsection 13.5.1.1). On the one hand, this class provides a virtual method `getBoundingBox` which derived classes should redefine. On the other hand, it allows the user to transform the data object using an arbitrary geometric transformation. The transformation is stored in an Open Inventor *SoTransform* node. This node is applied automatically to any display module attached to a transformed data object.

In total there are three different coordinate systems:

- The *world coordinate system* is the system where camera of the 3D viewer is defined in.
- The *table coordinate system* is usually the same as the world coordinate system. However, it might be different if special modules displaying, for example, the geometry of a radiotherapy device is used. These modules should call the method `HxBase::useWorldCoords` with a non-zero argument in their constructor. Later they may then call the method `HxController::setWorldToTableTransform` of the global object `theController`. In this way they can cause all other objects to be transformed simultaneously.

- Finally, the *local coordinate* system is defined by the transformation node stored for objects of type `HxSpatialData`. This transformation can be modified interactively using the transformation editor. Transformations can be shared between multiple data objects using the method `HxBase::setControllingData`. Typically, all display modules attached to a data object will share its transformation matrix, so that the geometry generated by these modules is transformed automatically when the data itself is transformed.

The transformation node of a spatial data object may be accessed using the `SoTransform*` `HxSpatialData::getTransform()` method, which may return a NULL pointer when the data object is not transformed.

Often it is easier to use `HxSpatialData::getTransform(SbMatrix& matrix)` instead, which returns the current transformation matrix or the identity matrix when there is no transformation. This matrix is to be applied by multiplying it to a vector from the right-hand side. It transforms vectors from the local coordinate system to the table or world coordinate system.

If you want to transform table or world coordinates to local coordinates, use `HxSpatialData::getInverseTransform( SbMatrix& matrix)`. For example, consider the following code which transforms the lower left front corner of object A into the local coordinate system of a second object B:

```
float bbox[6];
SbVec3f originWorld,originB;
SbMatrix matrixA, inverseMatrixB;

// Get origin in local coordinates of A
fieldA->getBoundingBox(bbox);
SbVec3f origin(bbox[0],bbox[1],bbox[2]);

// Transform origin to world coordinates:
fieldA->getTransform(matrixA);
matrixA.multVecMatrix(origin,originWorld);

// Transform origin from world coords to local coords of B
fieldB->getInverseTransform(inverseMatrixB);
inverseMatrixB.multVecMatrix(originWorld,originB);
```

Instead of this two-step approach, the two matrices could also be combined:

```
SbMatrix allInOne = matrixA;
allInOne.multRight(inverseMatrixB);

allInOne.multVecMatrix(origin,originB);
```

Note that the same result is obtained in the following way:

```
SbMatrix allInOne = inverseMatrixB;
allInOne.multLeft(matrixA);

allInOne.multVecMatrix(origin,originB);
```

Since the transformation could contain a translational part, special attention should be paid when directional vectors are transformed. In this case the method `HxSpatialData::getTransformNoTranslation( SbMatrix& matrix)` should be used.

13.5.6.3 Data Parameters and Materials

An arbitrary number of attributes or parameters can be defined for every data object. The parameters are stored in a member variable `parameters` of type `HxParamBundle`. The header file of the class `HxParamBundle` is located in the subdirectory `include/amiramesh`.

`HxParamBundle` is derived from the base class `HxParamBase`. Another class derived from `HxParamBase` is `HxParameter`. This class is used to actually store a parameter value. A parameter value may be a string or an n-component vector of any primitive data type supported in Avizo (byte, short, int, float, or double). The bundle class `HxParamBundle` may hold an arbitrary number of `HxParamBase` objects, i.e., parameters or other bundles. In this way, parameters may be ordered hierarchically.

Many data objects such as label images, surfaces, or unstructured finite element grids make use of the concept of a `material` list. Material parameters are stored in a special sub-bundle of the object's parameter bundle called *Materials*. In order to access all material parameters of such an object, the following code may be used:

```
HxParamBundle* materials = field->parameters.materials();
int nMaterials = materials->nBundles();

for (int i=0; i<nMaterials; i++) {
    HxParamBundle* material = materials->bundle(i);
    const char* name = material->name();
    theMsg->printf("Material[%d] = %s\n", name);
}
```

The class `HxParamBundle` provides several methods for looking up parameter values. All these *find*-methods return 0 if the requested parameter could not be found. For example, in order to retrieve the value of a one-component floating point parameter called *Transparency*, the following code may be used:

```
float transparency = 0;
if (!material->findReal("Transparency",transparency))
    theMsg->printf("Transparency not defined, using default");
```

In order to add a new parameter or to overwrite the value of an existing one, you may use one of several different *set*-methods, for example:

```
material->set("Transparency",transparency);
```

Many modules check whether a color is associated with a particular 'material' in the material list of a data object. If this is not the case, the color or some other value is looked up in the global material database Avizo provides. This database is represented by the class `HxMatDatabase` defined in

hxcore. It can be accessed via the global pointer `theDatabase`. Like an ordinary data object, the database has a member variable `parameters` of type `HxParamBundle` in order to store parameters and materials. In addition, it provides some convenience methods, for example `getColor(const char* name)`, which returns the color of a material, defining a new one if the material is not yet contained in the database.

13.6 Documentation of Modules in Avizo XPand Extension

Avizo XPand Extension allows the user to write the documentation for his own modules and integrate it into the user's guide.

The documentation must be written in Avizo's native documentation style. The syntax is borrowed from the *Latex* text processing language. Documentation files can easily be created by the `createDocFile` command. To create a documentation template for `MyModule`, type

```
MyModule createDocFile
```

in the Avizo console. This will generate a template for the documentation file as well as snapshots of all ports in the directory `AVIZO_LOCAL/src/mypackage/doc`.

The file `MyModule.doc` already provides the skeleton for the module description and includes the port snapshots.

The command `createPortSnaps` only creates the snapshots of the module ports. This is useful when the ports have changed and their snapshots must be updated in the user's guide.

13.6.1 The documentation file

Here, the basic elements of a documentation file are presented.

```
\begin{hxmodule}{MyModule}
This command indicates the begin of a description file. MyModule
is the module name.

\begin{hxdescription}
 This block contains a general module description.

All beginning blocks must have an end.
\end{hxdescription}

\begin{hxconnections}
\hxlabel{MyModule_data}
This command sets a label such that this
connection can be referenced in the documentation.
\hxport{Data}{\tt [required]}\\
 Here the required master connection is described.

\end{hxconnections}

\begin{hxports}
The module ports are listed here.
\end{hxports}
```

```
Anywhere in the documentation a label can be referenced:
\link{MyModule_data}{Text for reference}


\end{hxmodule}
```

This file describes the documentation of a module and starts with

```
\begin{hxmodule}{name}
```

This is a hxmodule description. Others are also available:

```
\begin{hxmodule2}{name}{short description to appear in the table of modules}
\begin{fileformat}{name}
\begin{fileformat2}{name}{short description to appear
in the table of file formats}
\begin{data}{name}
\begin{data2}{name}{short description to appear
in the table of data types}
```

The file always must be closed by the corresponding end command.
More commands Other formats allow formating and structure the documentation:

- ```
  \begin{itemize}
  \item This is an enumeration,
  and each item starts with the key word item
  \end{itemize}
  ```
- `{\bf This will be set in bold face}`
- `{\it This will be set in italics}`
- `{\tt This will be set in Courier}`

Formulas can be included by means of the text processor *LaTeX*. They must be written in the *Latex* syntax. This requires that *LaTeX* and *Ghostscript* be installed on the user's system. The following environment variables need to be set:

- DOC2HTML_LATEX points to the *LaTeX* executable
- DOC2HTML_DVIPS points to the dvi to PostScript converter (*dvips*)
- DOC2HTML_GS points to *Ghostscript*

13.6.2 Generating the documentation

All documentation files must be converted to HTML files and copied into the user's guide. For this purpose, the program doc2html is provided. Run this program from a command shell with the following option:
```
doc2html -A -skin AvizoSkin -d AVIZO_ROOT/share/doc/html -local
AVIZO_LOCAL
```
This converts the documentation and copies the HTML files to the appropriate places in the AVIZO_ROOT/share/doc/html directory. Call "doc2html -help" to get a complete list of options.

13.7 Miscellaneous

This section covers a number of additional issues of interest for the Avizo developer. In particular, the following topics are included:

- *Import of time-dependent data*, including the use of `HxPortTime`
- *Important global objects*, such as `theMsg` and `theProgress`
- *Save-project issues*, making save Avizo project work for custom modules
- *Troubleshooting*, providing a list of common errors and solutions

13.7.1 Time-Dependent Data And Animations

This section covers some more advanced topics of Avizo XPand Extension, namely the handling of dynamic data sets and the implementation of animated compute tasks. Before reading the section, you should at least know how to write ordinary IO routines and modules.

13.7.1.1 Time Series Control Modules

In general, the processing of time-dependent data sets is a challenging task in 3D visualization. Usually not all time steps of a dynamic data series can be loaded at once because of insufficient main memory. Even if all time steps would fit into memory, it is usually not a good idea to load every time step as a separate object in Avizo. This would result in a large number of icons in the Project View. The selection between different time steps would become difficult.

A better solution comprise special-purpose control modules. An example is the *Time Series Control* module described in the user's guide. This module is created if a time series of data objects each stored in a separate file is imported via the *Load time series...* option of the main window's file menu. Instead of loading all time steps together, the control module loads only one time step at a time. The current time step can be adjusted via a time slider. When a new time step is selected, the data objects associated with the previous one are replaced.

If you want to support a file format where multiple time steps are stored in a common file, you can write a special time series control module for that format. For each format, a special control module is needed because seeking for a particular time step inside the file, of course, is different for each format. For convenience, you may derive a control module for a new format from the class *HxDynamicDataControl* contained in the package *hxtime*. This base class provides a time slider and a virtual method `newTimeStep(int k)` which is called whenever a new new time step is to be loaded. In contrast to the standard time series control module, in most other control modules, data objects should be created only once. If a new time step is selected, existing objects should be updated and reused instead of replacing them by new objects. In this way, the burden of disconnecting and reconnecting down-stream objects is avoided.

13.7.1.2 The Class HxPortTime

In principal, an ordinary float slider (*HxPortFloatSlider*) can be used to adjust the time of a time series control module or of some other time-dependent data object. However, in many cases, the special-purpose class *HxPortTime* defined in the package *hxtime* is more appropriate. This class can be used

like an ordinary float slider but it provides many additional features. The most prominent one is the possibility to auto-animate the slider. In addition, *HxPortTime* can be connected to a global time object of type *Time*. In this way, multiple time-dependent modules can be synchronized. In order to create a global *Time* object, select *Animations And Scripts / Time* from the main window's *Project >Create Object...* menu.

Another feature of *HxPortTime* is that the class is also an interface, i.e., it is derived from *HxInterface* (compare subsection 13.5.1.2). In this way, it is possible to write modules which can be connected to any object containing an instance of *HxPortTime*. An example is the *Display Time* module. In order to access the time port of a source object, the following C++ dynamic cast construct should be used:

```
HxPortTime* time = dynamic_cast<HxPortTime*>(
    portData.source(HxPortTime::getClassTypeid()));
```

In the previous section, we discussed how time-dependent data could be imported using special-purpose control modules. Another alternative is to derive a time-dependent data object from an existing static one. An example of this is the class *MyDynamicColormap* contained in the example package of Avizo XPand Extension. Looking at the header file `src/mypackage/MyDynamicColormap.h` in the local Avizo directory, you notice that this class is essentially an ordinary colormap with an additional time port. Here is the class declaration:

```
class MYPACKAGE_API MyDynamicColormap : public HxColormap
{
    HX_HEADER(MyDynamicColormap);

  public:
    // Constructor.
    MyDynamicColormap();

    // This will be called when an input port changes.
    virtual void compute();

    // The time slider
    HxPortTime portTime;

    // Implements the colormap
    virtual void getRGBA1(float u, float rgba[4]) const;
};
```

The implementation of the dynamic colormap is very simple too (see the file `MyDynamicColormap.cpp`). First, in the constructor the time slider is initialized:

```
portTime.setMinMax(0,1);
portTime.setIncrement(0.1);
portTime.setDiscrete(0);
portTime.setRealTimeFactor(0.5*0.001);
```

The first line indicates that the slider should go from 0 to 1. The increment set in the next line defines by what amount the time value should be changed if the backward or the forward button of the slider is pressed. The next line unsets the discrete flag. If this flag is on, the slider value always would be

an integer multiple of the increment. Finally, the so-called real-time factor is set. Setting this factor to a non-zero value implies that the slider is associated with physical time in animation mode. More precisely, the number of microseconds elapsed since the last animation update is multiplied with the real-time factor. Then the result is added to the current time value.

In order to see the module in action compile, the example package, start Avizo (use the −debug option or the debug executable, if you compiled in debug mode), and choose *Other / DynamicColormap* from the main window's *Project >Create Object...* menu. Attach a *Colormap Legend* module to the colormap and change the value of the colormap's time slider. Animate the slider. The speed of the animation can be adjusted by resetting the value of the real-time factor using the Tcl command `DynamicColormap time setRealTimeFactor`.

13.7.1.3 Animation Via Time-Out Methods

In some cases, you might want certain methods to be called in regular intervals without using a time port. There are several ways to do this. First, you could use the Open Inventor class `SbTimerSensor` or related classes. Another possibility would be to use the Qt class `QTimer`. However, both methods have the disadvantage that the application can get stuck if too many timer events are emitted at once. In same cases, it could even be impossible to press the stop button or some other button for turning off user-defined animation. For this reason, Avizo provides its own way off registering time-out methods. The relevant methods are implemented by the class `HxController`. Suppose you have written a module with a member method called `timeOut`. If you want this method to be called automatically once in a second, you can use the following statement:

```
    theController->addTimeOutMethod(
this,(HxTimeOutMethod)timeOut,1000);
```

In order to stop the animation again, use

```
    theController->removeTimeOutMethod(
this,(HxTimeOutMethod)timeOut);
```

Instead of using a member method of an Avizo object class, you can also register an arbitrary static function using the method `addTimeOutFunction` of class `HxController`. The corresponding remove method is called `removeTimeOutFunction`. For more information, see the reference documentation of `HxController`.

The Avizo XPand Extension example package contains the module `MyAnimateColormap` which makes use of the above time-out mechanism. The source code of the module again is quite easy to understand. After compiling the example package, you can attach to module under the name *DoAnimate* to an existing colormap. The colormap then is modified and copied. After pressing the animate toggle of the module, the output colormap is shifted automatically at regular intervals. Note that in this example the `fire` method of the module is used as time-out method. `fire` invokes the modules `compute` method and also updates all down-stream objects.

13.7.2 Important Global Objects

Beside the base classes of modules and data objects, there are some more classes in the Avizo kernel that are important to the developer. Many of these classes have exactly one global in-

stance. A short summary of these global objects is presented here. For details, please refer to the online reference documentation by looking at the file `share/devrefAvizo/index.html` or `share/devrefAvizo/Avizo.chm` in the Avizo root directory.

HxMessage: This class corresponds to the Avizo console window in the lower right part of the screen. There is only one global instance of this class, which can be accessed by `theMsg`. All text output should go to this object. Text can be printed using the function `theMsg->printf("...",...)`, which supports common C-style `printf` syntax. `HxMessage` also provides static methods for popping up error and warning messages or simple question dialogs.

HxObjectPool: This class maintains the list of all currently existing data objects and modules. In the graphical user interface, this area is called the Project View and contains the modules and data objects icons. There is only one global instance of this class, which can be accessed by the pointer `theObjectPool`.

HxProgressInterface: This class displays the ports of selected objects in the Properties Area and provides the progress bar and busy-state functionality. Important functions are `startWorking`, `stopWorking`, `wasInterrupted` as well as `busy` and `notBusy`. There is only one global instance of this class, which can be accessed by the pointer `theProgress`.

HxFileDialog: This class represents the file browser used for loading and saving data. Normally the developer does not need to use this class since the standard I/O mechanism is completely implemented in the Avizo kernel. However, for special purpose modules, a separate file browser might be useful. There is a global instance of this class, which can be accessed by the pointer `theFileDialog`.

HxResource: This class maintains the list of all registered file formats and modules as defined in the package resource files. It also provides information about the Avizo root directory, the local Avizo directory, the version number, and so on. Normally there is no need for the developer to use this class directly. There is no instance of this class, since all its members are static.

HxViewer: This class represents an Avizo 3D viewer. There can be multiple instances which are accessed via the method `viewer` of the global object `theController`. Normally you will not not need to use this class. Instead, you should use the member functions `showGeom` and `hideGeom` which every module and data object provides in order to display geometry.

HxController: This class controls all 3D viewers and Open Inventor geometry. In order to access a viewer, you may use the following statement:

```
HxViewer* v0 = theController->viewer(0,0);
```

The first argument indicates the ID number of the viewer to be retrieved. In total, there may be up to 16 different viewers. The second argument specifies whether the viewer should be created or not if it does not already exist.

HxColorEditor: Avizo's color editor. Used, for example, to define the background color of the viewer. In a standard module, you should use a port such as `HxPortColorList` or `HxPortColorMap` instead of directly accessing the color editor. There is a global instance of this class, which can be accessed by the pointer `theColorEditor`.

HxHTML: A window used to display HTML files. This class is used for Avizo's online help. The global instance used for displaying the online user's guide and the online programmer's guide can be accessed by the pointer `theBrowser`.

HxMatDatabase: This class represents Avizo's global data parameter and material database. The

database can be accessed by the global pointer `theDatabase`. Details about the material database are discussed in subsection 13.5.6.3.

13.7.3 Save-Project Issues

This section describes the mechanism used in Avizo to save projects. For most modules, this is done transparently for the developer.

The menu command "Save Project" dumps a Tcl script that should reconstruct the current Avizo project. Essentially this is done by writing a `load ...` command for each data object, a `create ...` command for each module and `setValue ...` commands for each port of a module.

This suffices to reconstruct the Avizo project correctly if all information about a module's state is kept in the module's ports only. If this is not the case, e.g., if the developer uses extra member variables that are important for the modules current state, those values are not restored automatically. If you cannot avoid this, you must extend the "Save Project" functionality of your module. In order to do so, you can override the virtual function `savePorts` so that it writes additional Tcl commands. For example, let us take a look at the `HxArbitraryCut` class, which is the base class e.g., for the `Slice` module and which has to save its current slice orientation:

```
void HxArbitraryCut::savePorts(FILE* fp)
{
    HxModule::savePorts(fp);
    ...
    fprintf(fp, "%s setPlane %g %g %g %g %g %g %g %g %g\n",
        getName(),
        origin()[0], origin()[1], origin()[2],
        uVec()[0], uVec()[1], uVec()[2],
        vVec()[0], vVec()[1], vVec()[2]);
}
```

Note that this method requires that `HxArbitraryCut` or some of its parent classes implement the Tcl command `setPlane`. Hints about implementing new Tcl commands are given in subsection 13.4.2.2.

Some remarks about how to generate the load command for data objects are given in subsection 13.3.2.
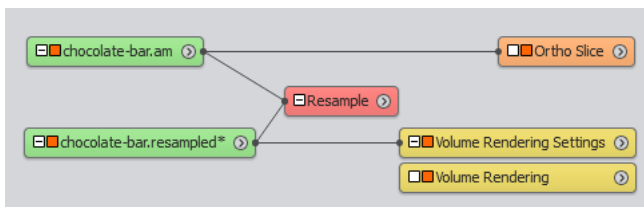
Figure 13.15: When loading this Avizo project, the *Resample* module recreates the *chocolate-bar.Resampled* data object on the fly.

There is a special optimization for data objects created by computational modules. Avizo automatically determines whether data objects which are created by other modules are not yet saved and asks the user

to do so, if necessary. Avizo proposes also two policies for saving projects: "Minimize project size" and "Minimize project computation". If the user selects "Minimize project size" then the data objects will not be saved at project saving and they will be computed at project loading, otherwise they will be saved at project saving and they will not be computed at project loading. As an example, consider the Avizo project in Figure 13.15. In this case, if the user selects "Minimize project size" as policy, the resample module can automatically recompute the `chocolate-bar.Resampled` data object when the Avizo project is loaded. An algorithm must determine whether the compute module can create the data object.

Determining whether a compute module can create a data object may be tricky. Typically, it must be assured that in the time between the actual creation of the data object by the computational module and the execution of the save Avizo project command, neither the parameters nor the input has changed, and that the resulting data object had not been edited.

This behavior is completely handled by `HxCompModule` while the method `HxCompModule::setResult` is used to register the output data object.

13.7.4 Troubleshooting

This section describes some frequently occurring problems and some general problem solving approaches related to Avizo development.

The section is divided into two parts: Problems which may occur when compiling a new package, and problems which may occur when executing it.

13.7.4.1 Compile-Time Problems

Unknown identifier, strange errors: A very common problem occurring in C++ programming is the omission of necessary include statements. In Avizo, most classes have their own header file (.h file) containing the class declaration. You must include the class declaration for each class that you are using in your code. When you get strange error messages that you do not understand, check whether all classes used in the neighborhood of the line that the compiler complains about have their corresponding include statement.

Unresolved symbols: If the linker complains about unresolved symbols, you probably are missing a library on your link line. The Avizo development wizard makes sure that the Avizo kernel library and important system libraries are linked. If you are using Avizo data classes, you will need to link with the corresponding package library `hxfield`, `hxcolor`, `hxsurface`, and so on. To add libraries, edit the `Package` file, find the line starting with `LIBS`, append the name of the package you want to add and regenerate the build system files with the Development Wizard of Avizo. Details are given in subsection 13.2.10 and in subsection 13.2.9.

13.7.4.2 Run-Time Problems

The module does not show up in the popup menu: If your module did compile, but is not visible in the popup menu of a corresponding data object, there is probably a problem with the resource file. The resource file will be copied from your package's `share/resources` directory to the directory `share/resources` in your local Avizo directory during compilation. Launch a new build each time the resource file is modified.

If the resource file is present, the next step is to check whether it is really parsed. Add a line `echo "hello mypackage"` to the resource file. Verify that the message appears in the Avizo console when Avizo starts. If not, probably the Local Avizo directory is not set correctly. Reset it with the Development Wizard in Avizo. Details are given in subsection 13.2.2

If the file is parsed, but the module still does not show up, the syntax of the rc file entry might be wrong or you specified a wrong primary data type, so that the module will appear in the menu of a different data class.

There is an entry in the popup menu, but the module is not created: Probably something is wrong with the shared library (the .so or .dll file). In the Avizo console, type `dso verbose 1` and try to create the module again. You will see some error messages, indicating that either the dll is not found, or that it cannot be loaded (and why) or that some symbol is missing. Check whether your building mode (debug/optimize) and execution mode are the same. In particular, if you have compiled debug code you must start Avizo using the `-debug` command line option or the debug executable (see subsection 13.1.5).

A read or write routine does not work: The procedure for such problems is the same. First check whether the load function is registered. Then verify that your save-file format shows up in the file format list when saving the corresponding data object. For a load method, right-click on a filename in the load-file dialog. Choose format and check whether your format appears in the list. If that is the case, you probably have a dll problem. Follow the steps above. If the library can be loaded, but the symbol cannot be found, your method may have either a wrong signature (wrong argument types) or on Windows you might have forgotten the `<PACKAGE>_API` macro. This macro indicates that the routine should be exported by the DLL.

In general, if you have problems with **unresolved** and/or **missing symbols**, you should take a look at the symbols in your library. On Unix, type `nm lib/arch-*-Debug/libmypackage.so`. On Windows, type in Visual Studio command prompt: `dumpbin /exports bin/arch-*-Debug/mypackage.dll`.

13.7.4.3 Debugging Problems

Setting breakpoints does not work: Since Avizo uses shared libraries, the code of an individual package is not loaded even after the program has started. Therefore, some debuggers refuse to set breakpoints in such packages or disable previously set breakpoints. To overcome this problem, first create your module and then set the breakpoint. If you want to debug a module's constructor or a read or write routine, of course, this does not work. In these cases, load the library by hand, by typing into the Avizo console `dso open libmypackage.so` (if your package is called mypackage). Then set the breakpoint and create your module or load your data file.

Chapter 14

Avizo XMetrology Extension

The Avizo Metrology workroom is only available on a Windows platform, and the workroom allows creating test plans consisting of set of measures on a part. Once a test plan has been created and saved on a reference part, then test plan can be reapplied to any similar part to validate or reject the part. Measures should be created according to specific orientations. The first step is to realign the part according to a specific orientation since a part may be tilted during CT acquisition, for instance, to optimize the beam path. Different tools are provided to define orientation, including directions defined by created geometries on the part. Once an orientation (i.e., Local Coordinate System or LCS) or multiple orientations have been defined and geometries have been created, you can create measurements. A set of LCS, geometries, and measures can be saved as a test plan.

- *Data selection*
- *Data visualization*
- *Geometries fitting*
- *Inherited Geometries*
- *Local Coordinate Systems (LCS) creation*
- *Measurements creation*
- *Test plans usage*

14.1 Data Selection

14.1.1 General

When switching to the *Metrology* workroom, the *Data Selection* panel (see Figure 14.1) will allow you to select the following:

- Data on which measurements should be done
- The type of surface defining the interface between the part and the "outside"
- A mask for additional types of measure
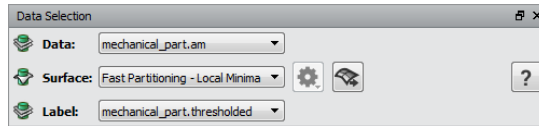
Figure 14.1: The Data Selection Panel

14.1.2 Metrology Surface Determination

There are several methods to automatically determine the surface from selected data. The other surfaces listed in this pulldown menu are surfaces from the *Project View*. This surface determination is important because geometries fitting will rely on it and consequently most of the measures will also rely on it. For all *Fast Partitioning* methods, surface is extracted using an *Isosurface* module. The only difference between the methods is the threshold determination method to compute the surface. The current surface determination method has parameters that you can edit using the *Surface Determination Parameters* button.
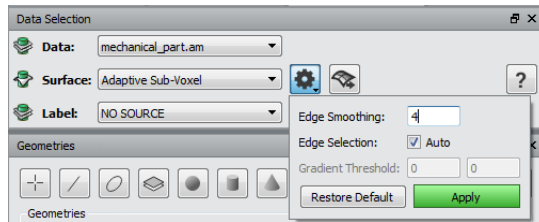
Figure 14.2: Surface Determination Parameters for Adaptive Sub-Voxel method

If parameters are modified, you can recompute the surface or restore default values.

14.1.2.1 Fast Partitioning - Local Minima

The threshold is automatically determined using automatic *Intensity Range Partitioning* on selected data. See the *Intensity Range Partitioning Editor* documentation for more details.

14.1.2.2 Fast Partitioning - ISO 50

The threshold is determined using an ISO 50 algorithm:

1. Set an arbitrary threshold.
2. Using the threshold, build two sets of values:

   - *Background Material values:* Data values that are below the threshold.
   - *Gray Material values:* Data values that are over the threshold.

3. Compute the mean of each value set.

4. Compute a new threshold using the following formula:
$$T_{ISO50} = \frac{MeanGrayMaterial + MeanGrayBackground}{2}$$
5. Repeat steps 2,3, and 4 until threshold converges.

14.1.2.3 Fast Partitioning - Otsu

The threshold is determined using *Otsu* criterion. The value is determined by maximizing the between-class variance:

$$\sigma_B^2[t] = \omega_0[t] \times \omega_1[t] \times (\mu_0[t] - \mu_1[t])^2$$

Where:

- $\omega_0[t]$ and $\omega_1[t]$ are respectively the probabilities of being in classes C_0 and C_1.
- $\mu_0[t]$ and $\mu_1[t]$ are respectively the means of classes C_0 and C_1.

The algorithm is the following:

1. Compute histogram and probabilities $\omega_0[t]$ and $\omega_1[t]$
2. Define initial $\omega_0[0]$ and $\omega_1[0]$
3. Go through all available thresholds t

    - Update ω_i and μ_i
    - Compute $\sigma_B^2[t]$

4. Otsu threshold corresponds to $\sigma_B^2[t]$ maximum

Note: Float data are not supported for this method. Modules *Background Detection Correction* and *Convert Image Type* can be used to convert data type.

14.1.2.4 Adaptive Sub-Voxel

This is an advanced and CPU/RAM-intensive algorithm that provides a more accurate surface computation. The computation may take some extra time the first time this option is selected, but the result is cached so that it will not be recomputed if another surface is selected. Measurements are expected to have a precision smaller than the actual voxel size of the acquired volume (resolution). This surface is computed using *Morphological Laplacian*, *Image Gradient*, *Interactive Thresholding*, or *Auto Thresholding* and an *Isosurface* filtered by label field. Some of the preferences under the *Metrology Preferences* allow specifying *Morphological Laplacian* and thresholding.

Adaptive Sub-Voxel Parameters
The parameters are used to generate *Morphological Laplacian* (i.e., *Edge Smoothing* parameter) and labeling (i.e., *Edge Selection* parameter) data to generate an accurate surface.

- *Edge Smoothing:* Kernel size of *Morphological Laplacian* filter
- *Edge Selection: Interactive* in which the viewer mask needs to be configured, or *Auto Thresholding* mode

Note: Float data are not supported for this method. Modules *Background Detection Correction* and *Convert Image Type* can be used to convert data type.

14.1.2.5 Adjusted Sub-Voxel

This method adjusts a given surface to the sub-voxel accuracy by moving points along the gradient direction and snapping them to the local gradient maxima. The points are moved along a path defined by the gradient vector field. If there are several gradient maxima within the search distance, gradient magnitude and distance from the starting point are used to make a decision.
The input surface is extracted using an *Isosurface* module.

Adjusted Sub-Voxel Parameters

- *Maximum Distance:* Defines the distance within which the initial determined surface can be adjusted. This value can be modified depending on image quality, for instance increased in case of strong beam hardening artefact, or reduced for good quality images in particular with thin wall structures
- *Initial Threshold:* Allows selecting the threshold used by *Isosurface* to extract the surface used as input of Adjusted Sub-Voxel algorithm. The available thresholds are Local Minima, Otsu and ISO 50.

Note: Float data are not supported for Otsu threshold. Modules *Background Detection Correction* and *Convert Image Type* can be used to convert data type.

14.1.2.6 Export Surface in Project View

For each surface computation method, click **Export** to export the computed surface in project view.

14.2   Data Visualization

Some visualization options are available in the viewers toolbar.

14.2.1   Metrology 3D Display

This button displays data volume rendering or surface view (see Figure 14.3). Points used for geometry fitting are picked on the surface.

Figure 14.3: Metrology 3D Display Button

14.2.2   Volume Rendering Options

These options are visible when the volume rendering is activated and the current viewer is set to the 3D viewer (see Figure 14.4).
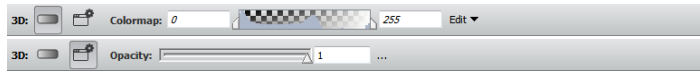
Figure 14.4: Volume Rendering Options

- *Colormap*: To set volume rendering colormap
- *View Parameters*: To set volume rendering opacity

14.2.3   Surface Options

These options are visible when the surface is activated and current viewer is 3D viewer (see Figure 14.5).
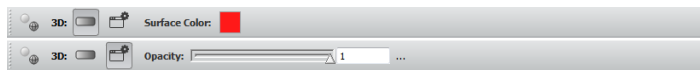
Figure 14.5: Surface Options

- *Surface Color*: To set color for surface view. This option also changes the color for isocontours in 2D viewers.
- *View Parameters*: To set opacity for surface view

14.2.4   Ortho Views Options

These options are visible when current viewer is a 2D viewer (see Figure 14.6).
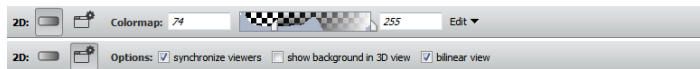
Figure 14.6: Ortho Views Options

- *Colormap*: To set colormap for slices
- *View Parameters*: See *Ortho Views* options.

14.3   Geometries Fitting

14.3.1   General

The *Metrology* workroom includes a dockable *Geometries* panel (see Figure 14.7) from where geometries can be fitted. Once data, surface, and optional mask have been selected through the *Data Selection* panel, geometries can be fitted.
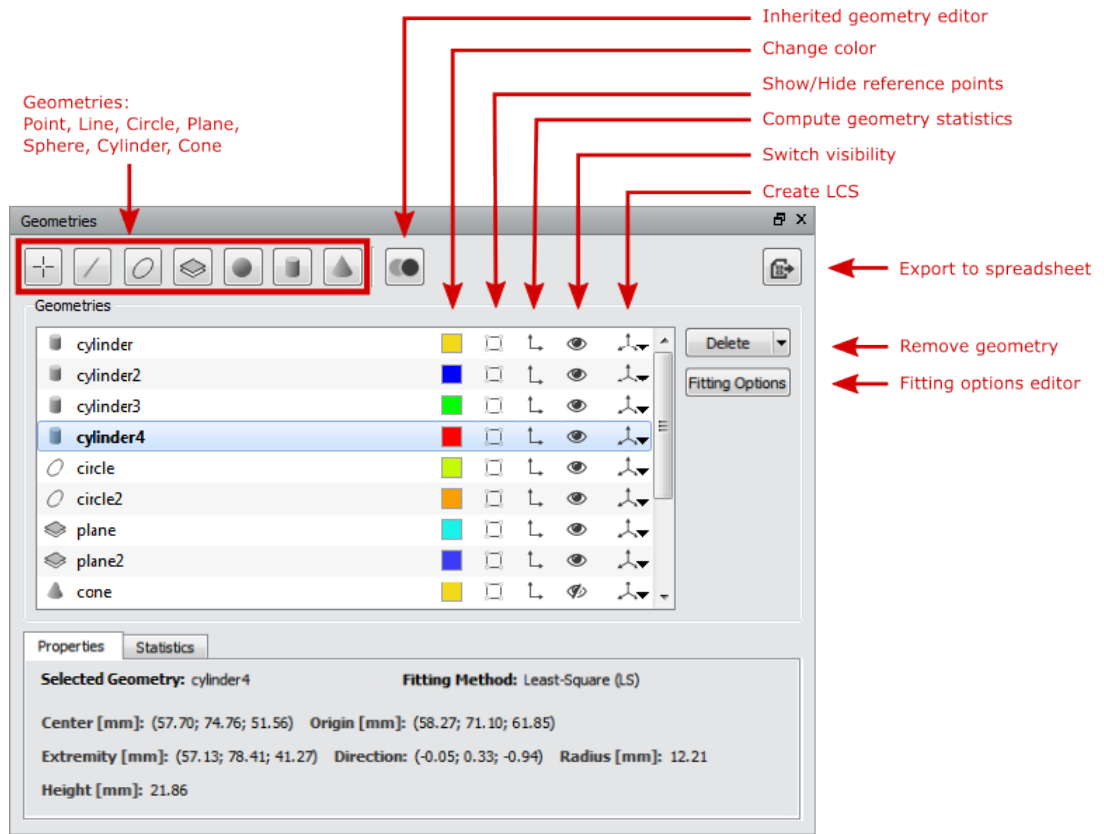
Figure 14.7: The Geometries Panel

Geometry fitting relies on selecting points that support the geometry lying on the interface between the part and the "outside" of the part (e.g., air). Select these points by clicking in one of the four viewers provided in the layout using 3D or 2D view. When a point is selected, it is projected on the surface set in the *Data Selection* panel. Different point selection tools are provided in the toolbar on top of the viewers.

To fit geometry on an area that is not visible or hidden in the volume, activate the clipping by clicking

 in the lower right corner of each 2D viewer (see Figure 14.8). Clipping is done through a plane in which orientation is defined by the view orientation of the 2D viewer in one of the following modes:

 No clipping is applied.

 The clipping direction is either the direction of the normal to the plane or its inverse, depending on which one forms the largest angle with the camera direction.

 The clipping direction is the direction of the normal to the plane.

 The clipping direction is the opposite direction of the normal to the plane.
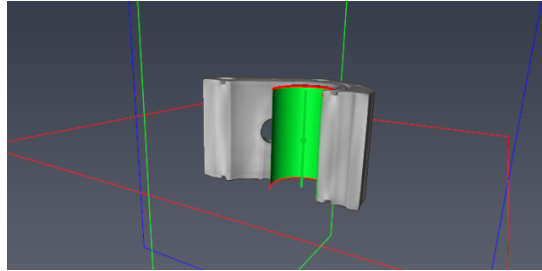
Figure 14.8: Clipping Auto Activated to Fit Inside the Part

14.3.2   Geometry Fitting Toolbar
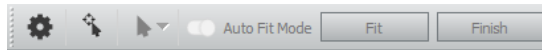
Figure 14.9: Geometry Fitting Toolbar

This toolbar contains all geometry-related tools. The different tools include:

- *Advanced Parameters:* Use *Metrology Preferences* to customize the fitting method.
- *Pick Geometry:* Pick any geometry in 3D viewer to select the corresponding geometry in the geometries panel.
- *Fitting Tools:* This pulldown menu contains all the tools to fit a geometry in the 3D or 2D viewers.
- *Auto Fit Mode:* Enable/Disable *Auto Fit* mode. In *Auto Fit* mode, the geometry fitting is updated every time you select a new set of points.
- *Fit:* If *Auto Fit* mode is disabled, the geometry will be fitted only after you click this button.
- *Finish:* To stop the fitting editing when you finished picking points.

14.3.3   Fitting Tools

The different fitting tools include:

- *Pick Tool*
- *Knn Tool*
- *Radius Tool*
- *Magic Wand Tool*

When sets of points are selected with these tools, the last set is highlighted in red and the others are highlighted in yellow and blue. Sets in yellow and red can be deselected by the Undo action in the order of selection. The red color indicates which set is the next to be deselected by the Undo action. Blue sets cannot be deselected because they correspond to the minimum point set to fit the selected geometry.

14.3.4 Fitting Methods

Several fitting methods can be used in the *Metrology* workroom, with the following configuration options:

- *Least Square Method (LS)*
- *Minimum Zone Method (MZ)*
- *Minimum Zone Inner Method (MZI)*
- *Minimum Zone Outer Method (MZO)*
- *Maximum Inscribed (MI)*
- *Minimum Circumscribed (MC)*
- *Auto-Adjustment Refit*
- *RANSAC Parameters*

In Avizo, the PCL library is used to do the fitting with the *Least Square Method (LS)*. The other fitting methods are based on the *Chebyshev* algorithm and are initialized with the result of the *Least Square Method (LS)*. For more detailed information about the *Chebyshev* algorithm, see the following article *Reference Algorithms for Chebyshev and One-sided Data Fitting for Coordinate Metrology* `http://www.nist.gov/customcf/get_pdf.cfm?pub_id=822103`.

The fitting methods based on the *Chebyshev* algorithm required a minimum number of points per geometry. For the 3D geometries, the points used to fit must represent 3D volume.

Geometry	Mathematical minimum	Recommended
Line	2	5
Plane	3	9
Circle	3	7
Sphere	4	9
Cylinder	5	15
Cone	6	15

The default fitting method can be changed via the *Metrology Preferences.*
The fitting method of a geometry can be changed using the *Fitting Options Editor* (see Figure 14.10).
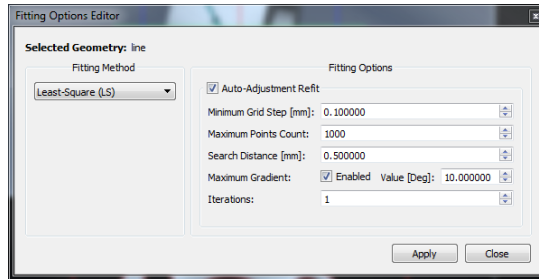
Figure 14.10: Fitting Options Editor

14.3.4.1 Least Square Method (LS)

For now in Avizo, the *Least Square* method can be used to fit the following: lines, circles, planes, spheres, cylinders, and cones. After you clicked points in the viewers, the points are registered on the data surface. The result point cloud is given as an entry of the *Least Square* algorithm, which tries to find the best fitting geometry by minimizing the sum of the squares of the point-shape distances.

$$R^2 \equiv \sum [y_i - f(x_i, a_1, a_2, ..., a_n)]^2$$

The previous formula is the sum of the squares of the vertical deviations R^2 of a set of n data points. R^2 is minimized when the following equation is verified:

$$\frac{\partial (R^2)}{\partial (a_i)} = 0$$

Here is an example of 2D Circle fitting using the *Least Square* algorithm (see Figure 14.11):
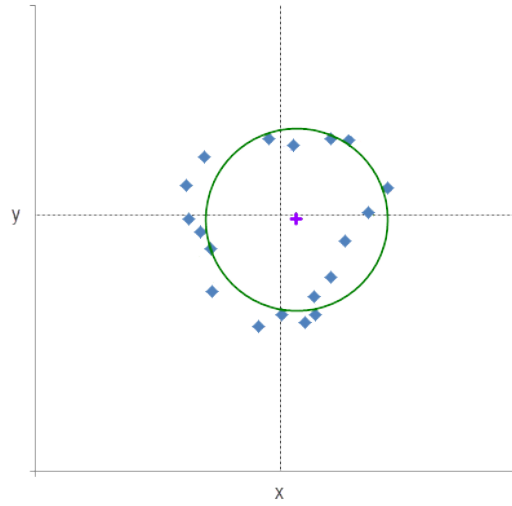
Figure 14.11: Least Square Fitting of a 2D Circle

However, outlying points can have a disproportionated effect on the fit, because squares of the offsets are used. This can be avoided by using *RANSAC Parameters*.

14.3.4.2 Minimum Zone Method (MZ)

The Minimum Zone method is based on the *Chebyshev* algorithm. This method can be used to fit the same geometries as the *Least Square Method (LS)*.

- *Line*: The line that is the axis of the minimum-circumscribed cylinder of the points.
- *Plane*: The median plane, situated halfway between two distinct parallel planes of minimal separation, containing all the points within the space separating them.
- *Circle*: The median circle, situated halfway between two distinct concentric circles of minimal separation, containing all the points within the space separating them.
- *Sphere*: The median sphere, situated halfway between two distinct concentric spheres of minimal separation, containing all the points within the space separating them.
- *Cylinder*: The median cylinder, situated halfway between two distinct coaxial cylinders of minimal separation, containing all the points within the space separating them.
- *Cone*: The median cone, situated halfway between two distinct coaxial cones of minimal separation (i.e., that have the same apex angle), containing all the points within the space separating them.

The following Figure 14.12 illustrates the Minimum Zone criterion for a 2D Circle, with the same point cloud used in the *Least Square Method (LS)*:
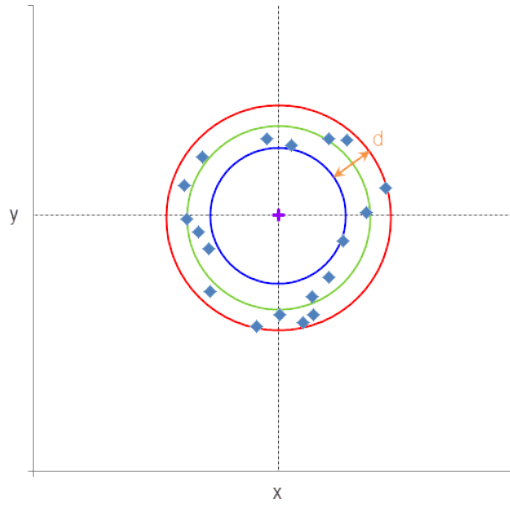
Figure 14.12: Minimum Zone Fitting of a 2D Circle. Note: the fitted circle is the green circle

The goal is to find the two circles that minimize the distance d. The minimum zone circle is the mean of the two circles.

14.3.4.3 Minimum Zone Inner Method (MZI)

The *Minimum Zone Inner* method is based on the *Chebyshev* algorithm. This method can be used to fit the same geometries as the *Least Square Method (LS)*. It follows the same algorithm as the *Minimum Zone Method (MZ)*. The only difference is that instead of taking the mean geometry, we take the minimum one (i.e., the blue circle in Figure 14.12).

14.3.4.4 Minimum Zone Outer Method (MZO)

The *Minimum Zone Outer* method is based on the *Chebyshev* algorithm. This method can be used to fit the same geometries as the *Least Square Method (LS)*. It follows the same algorithm as the *Minimum Zone Method (MZ)*. The only difference is that instead of taking the mean geometry, we take the maximum one (i.e., the red circle in Figure 14.12).

14.3.4.5 Maximum Inscribed (MI)

The *Maximum Inscribed* method calculates a geometry such that all points are outside it, and the geometry is as large as possible. This method can be used to fit the following: circles, cylinders and spheres (see Figure 14.13).
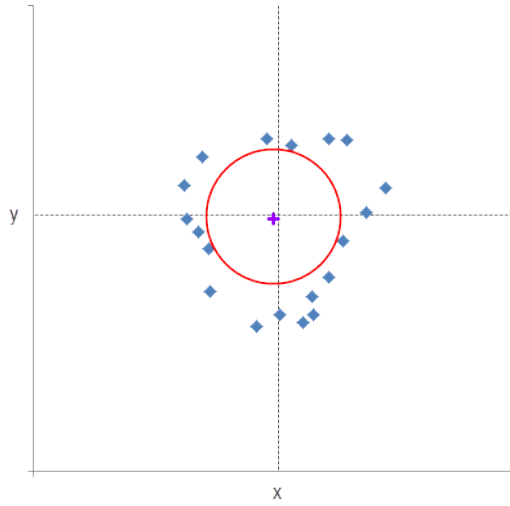
Figure 14.13: Maximum Inscribed Fitting of a 2D Circle

14.3.4.6 Minimum Circumscribed (MC)

The *Minimum Circumscribed* method calculates a geometry such that all points are inside it, and the geometry is as small as possible. This method can be used to fit the following: circles, cylinders and spheres (see Figure 14.14).

Note: Unlike the *Minimum Zone Inner Method (MZI)* and the *Minimum Zone Outer Method (MZO)*, the center of the circumscribed shape is often different from the center of the inscribed shape.
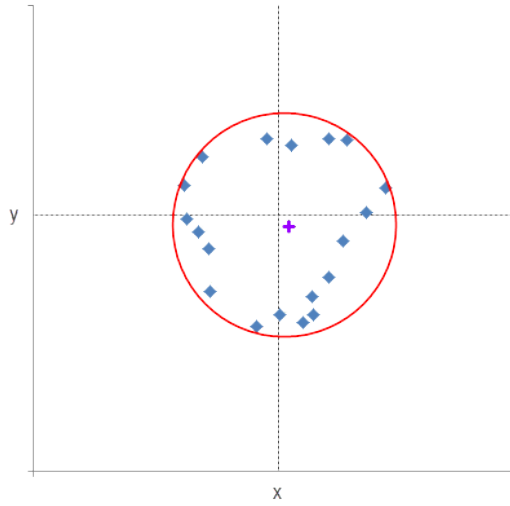
Figure 14.14: Minimum Circumscribed Fitting of a 2D Circle

The following table summarizes the different fitting methods and the geometries you can fit with them:

Geometry	Least Squares	Minimum Zone	Minimum Inscribed	Maximum Circumscribed
Line	X	X		
Plane	X	X		
Circle	X	X	X	X
Sphere	X	X	X	X
Cylinder	X	X	X	X
Cone	X	X		

14.3.4.7   Auto-Adjustment Refit

Auto-Adjustment Refit is a feature that allows you to get better-distributed support points and to optimize fitting accuracy. This feature relies on tesselation algorithm and projects discretized points to closest points onto the surface before *Least Square Method (LS)*.

Auto-Adjustment Refit is enabled by default. This default behaviour can be changed in the *Metrology Preferences*. It can also be changed individually on a geometry using the *Fitting Options Editor*.

You can set the following parameters for *Auto-Adjustment Refit* algorithm:

- *Minimum Grid Step*: Minimum distance between neighboring points
- *Maximum Points Count*: Maximum number of support points
- *Search Distance*: Maximum search distance for tessellation algorithm; points located above this distance will be discarded (use when the surface contains holes)

- *Maximum Gradient*: Maximum angular deviation allowed between the normal of the reference and the surface normal of a fit point. If the angle is greater than the specified value, the fit point is discarded. This can be switched On/Off
- *Iterations*: Number of tessellations to perform on the row

14.3.4.8   RANSAC Parameters

RANSAC (RANdom SAmple Consensus) algorithm can be used for each fitting method as it filters the support point cloud given to fitting methods. It searches and removes outlier points, so they are ignored when doing the fitting (see Figure 14.15).
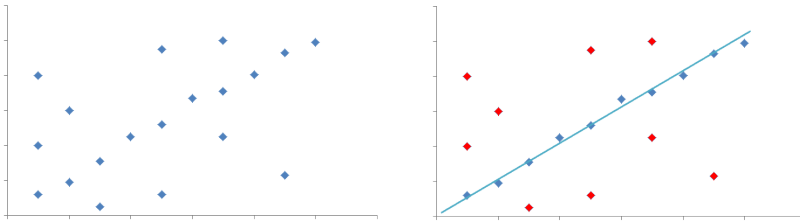
Figure 14.15: **Left:** Unfiltered point cloud on which a line should be fitted. **Right:** Point cloud filtered with RANSAC; red points are ignored for line fitting.

RANSAC can be enabled in the *Metrology Preferences*.
You can set the following parameters for RANSAC algorithm:

- *Distance Threshold*: Maximum distance of the points to the calculated geometry.
- *Direction Weight*: Describes the normal for the support point. The value must be between 0 and 1, in which 0 means normals are ignored and 1 means that only points with normal exactly fitting the geometry's normals are accepted.
- *Iteration Number*: Number of iterations of RANSAC algorithm. A maximum is set to 5,000 iterations. There is a formula to estimate the number of iterations needed to obtain a result:

$$Nb_{iteration} = \frac{\log{(1.0 - P)}}{1.0 - P_i^{Nb_{MinPointToFit}}}$$

- P is the probability to have a good result.
- P_i is the probability that picked point should be an inlier.
- $Nb_{MinPointToFit}$ is the minimal points number needed to fit a selected geometry.

The more P is high and the more P_i is low will imply a great $Nb_{iteration}$. Most complicated geometries (sphere and cone) have $Nb_{MinPointToFit} = 4$. Then, if P is set to 90% and P_i to 20%, $Nb_{iteration} \approx 1,500$.

- *Radius Range*: Radius limit of the geometry

14.4 Inherited Geometries

14.4.1 General

In addition to fitting, the *Metrology* workroom provides another way to create geometries using inheritance. The following operations are available:

- *Intersection*: Create a new geometry by intersecting existing geometries.
- *Composition*: Create a new geometry by composing existing geometries.
- *Projection*: Create a new geometry by projecting existing geometries.

14.4.2 Inherited Geometry Editor

To create an inherited geometry, click *Create Inherited Geometry...* next to the geometry buttons (see Figure 14.7). This opens the *Inherited Geometry Editor*:
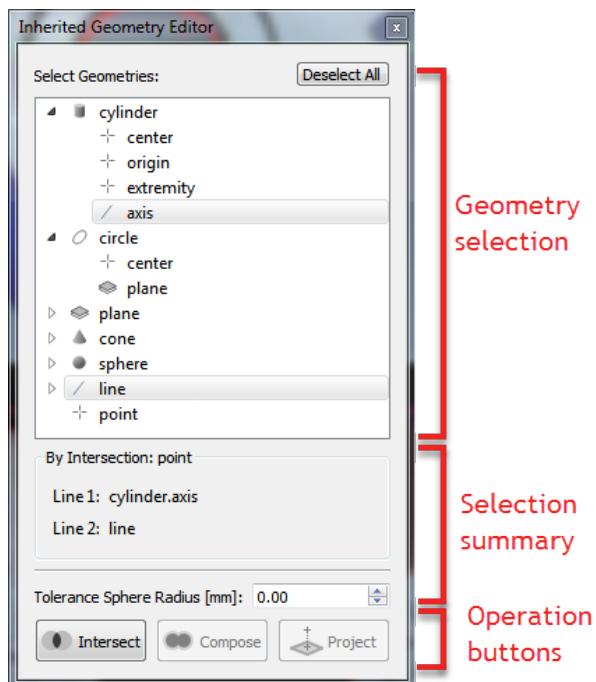
Figure 14.16: Inherited Geometry Editor

The first part lists all available geometries and their parts. Any item can be selected by clicking on it. Once the selection allows creating an inherited geometry, the second part will be updated with the following information:

- Selection summary. There, the selection can be quickly switched if selection order is relevant.
- Relevant parameters.

Then, one or more of the operation buttons of the third part is available. When you click on an operation button, the application tries to perform the corresponding operation. If the operation succeeds, then the resulting geometry is added to the list. Otherwise, an error message will pop up describing why the operation failed. Once an operation has been performed, the selection is cleared and the process can be performed again to create another inherited geometry.

The next section describes the available combinations for each operation.

14.4.3 Intersection

Using the *Intersection* operation, you can create the following geometries:

- **Point**
  - Intersection of 2 lines. In this case , the radius of a *Tolerance Sphere* can be set to consider or not the intersection
  - Intersection of a plane and a line
  - Intersection of 3 planes

- **Line**
  - Intersection of 2 planes

The selection order in *Inherited Geometry Editor* does not matter in each of the previous cases.

14.4.4 Composition

Using the *Composition* operation, you can create the following geometries:

- **Line**
  - Composition of 2 points. The first selected point will be the line origin, the other one will be its extremity.
  - Composition of a point and a direction (vector).

- **Plane**
  - Composition of 3 points. The first point will be the plane origin, the second one will determine the width of the plane and the last one will determine its height.
  - Composition of a point and 2 vectors. The point will be the plane origin. The first vector will be used as the half-width vector of the plane and the second one as its half-height vector.

Note: It is possible to quickly switch the selection in the *selection summary* part of the *Inherited Geometry Editor* (see Figure 14.16) in all cases where selection order matters.

14.4.5 Projection

Using the *Projection* operation, it is possible to create the following geometries:

- **Point**

  - Projection of a point on a line.
  - Projection of a point on a plane.

- **Line**

  - Projection of a line on a plane.

14.4.6 Inherited Geometry Updates

If a geometry is updated (for example a new fit), then any inherited geometry generated using the updated one will be recomputed. In the same way, if a geometry is deleted, any inherited geometry generated using the deleted one will be deleted too.

14.5 Local Coordinate Systems (LCS) Creation

14.5.1 General

As explained in the introduction, a part might need to be realigned according to a specific orientation since it might have tilted during CT acquisition to optimize the beam path and minimize acquisition artifacts. The geometries are used to define the directions of Local Coordinate Systems (LCS). Different tools are provided to create these LCSs and multiple LCSs can be created. Visualization can be switched from one LCS to another to create measures that are aligned with the directions of a particular LCS. As pictured below, LCS can be created from the *Geometries* panel or from the LCS toolbar (see Figure 14.17).
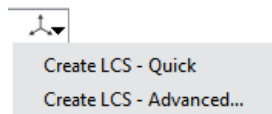
Figure 14.17: Creating LCS from Fitting Panel

Multiple LCSs can be created with the different tools and a user can switch from one to another using the LCS toolbar. This allows aligning measures according to different orientations of the part.

14.5.2 LCS Toolbar

This toolbar contains all Local Coordinate Systems (LCS) related tools (see Figure 14.18).
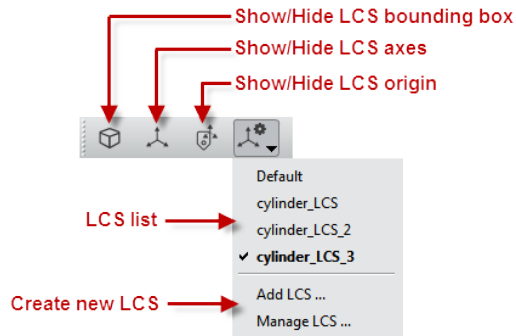
Figure 14.18: The LCS Toolbar

The different tools include:

- *Bounding box*: Shows/hides the LCS bounding box
- *Axes*: Shows/hides the LCS axes
- *Origin*: Shows/hides the LCS origin
- *Options*: Displays the LCS options. This menu contains:

  - The LCS list, which allows you to quickly set the current LCS
  - An access to the LCS Wizard (*Add LCS...*), which allows the creation of LCS
  - An access to the LCS Manager (*Manage LCS...*) which allows all LCS related actions (creation, deletion, renaming, display of the LCS characteristics). See Figure 14.19.
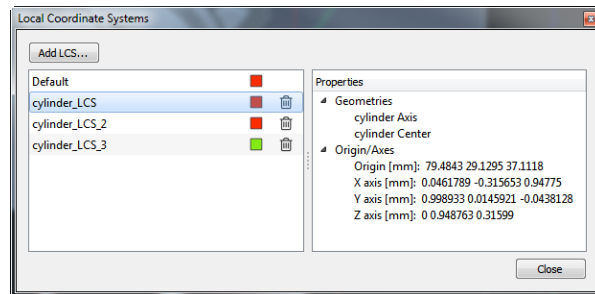
Figure 14.19: The LCS Manager

14.6   Measurements Creation

Avizo allows creating measures. Two main categories of measures are provided. One is based on geometries that have been created on the interface of the part with the "outside" of the part and the other

is based on the optional mask provided in the *Data Selection* panel. The geometry-based measures give a sub-resolution precision measure, while the mask-based ones give measures with a resolution precision. The *Measures* panel (see Figure 14.20) gives access to all of the measurement tools:
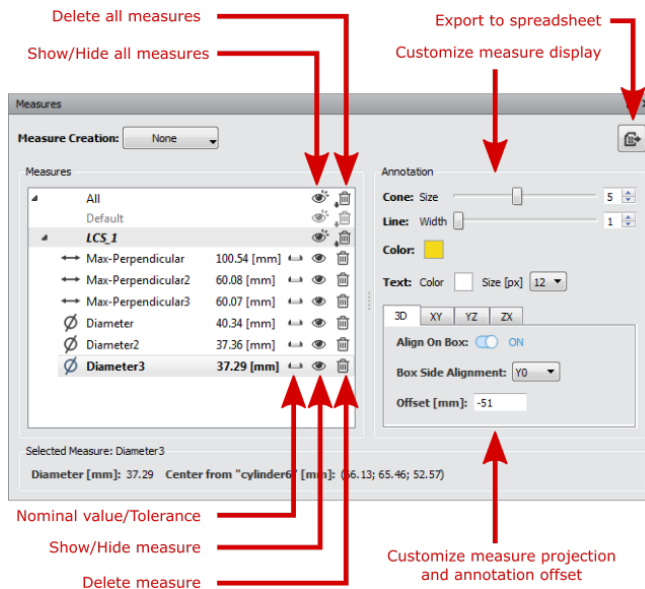
Figure 14.20: The Measures Panel

14.7 Test Plans Usage

Once the geometries have been created, the LCS and measures can be created and they can be saved as a test plan. A collection of test plans can be created and reapplied on the other parts that need to be validated against a reference model. This action is performed through the top *Metrology* menu, where a test plan can be saved or opened.

To become more familiar with the usage of the Metrology workroom, you can follow the step-by-step *Metrology tutorial.*

Chapter 15

Avizo XRecipe Extension

Avizo allows for the creation of user-defined *recipes* to automate the execution of a series of modules. Avizo recipes facilitate the application of high-level workflows such as filtering, segmentation, and the extraction of user-defined statistics from an image. Recipes integrate your knowledge and make it available for other users as *push button* solutions.

A dedicated workroom is available to create, edit, and execute recipes. The workroom can be launched by selecting the *Recipes* icon located on the *Workroom Toolbar*. The workroom consists of three panels: an instance of the common 3D Viewer, a dedicated *Recipes* panel presenting the sequence of steps in a list, and the *Properties* panel showing the module properties of the current step. The main interaction area is the *Recipes* panel, which let's you edit and execute your recipes (see Figure 15.1). Once a recipe has been selected from the dropdown menu in the *Recipes* panel, it can be played back and edited. Avizo will load all recipes found in the standard recipe folder of your home directory. By default, this folder is located at `%APPDATA%/FEI/Recipes`. You can click **Load Recipe** to load additional recipes, and click **Save Recipe** to save the recipes on disk to share them with other users.
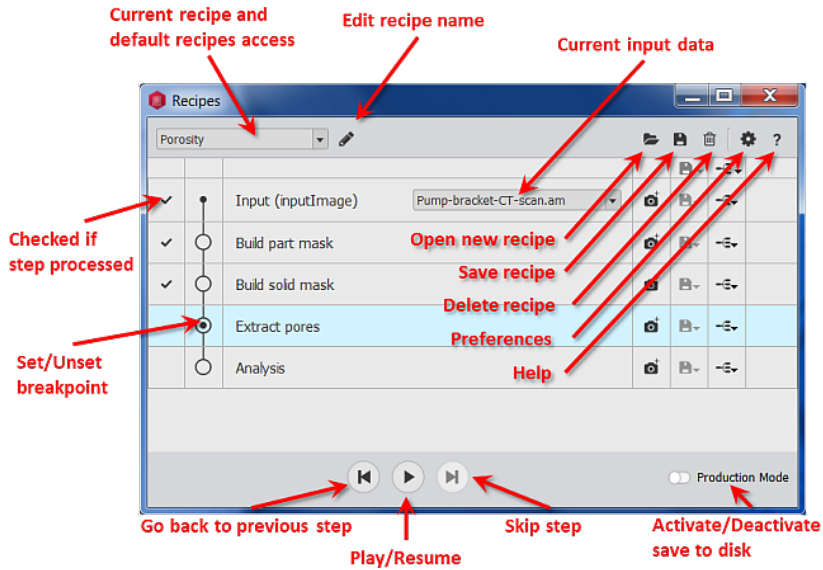
Figure 15.1: The Recipes Panel

Recipes can be executed at once or with *breakpoints* in which case the execution of the recipe will pause at user-defined steps (see Figure 15.2). By default, the recipe configures the modules as you save them. Breakpoints allow you to change the properties of a module used in the recipe during runtime. In this way, a recipe can be executed several times by only changing one parameter to evaluate the sensitivity of that parameter. For example, you can change a threshold to evaluate if porosity is sensitive to the segmentation parameter.

Note: Modules that require user interaction (e.g., paint) will automatically force a breakpoint.

For each step, you can turn on/off saving the results to disk and exporting them in the *Project* workroom. If the export to the *Project* workroom is off, the data will be removed from memory as soon as it is no longer needed. This action allows you to reduce the memory consumption.

Note: Your results will be saved to disk only if *Production Mode* is turned on.
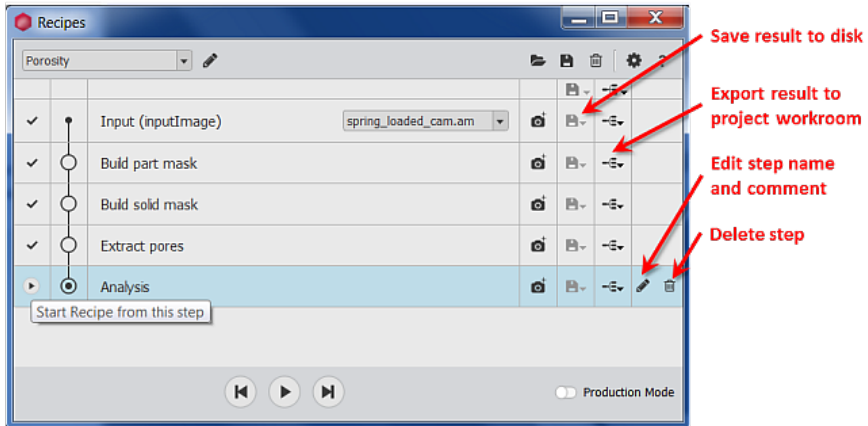
Figure 15.2: Edit Recipe

When a recipe has been played back, hovering with the mouse over a step displays a preview thumbnail of the result image (see Figure 15.3).
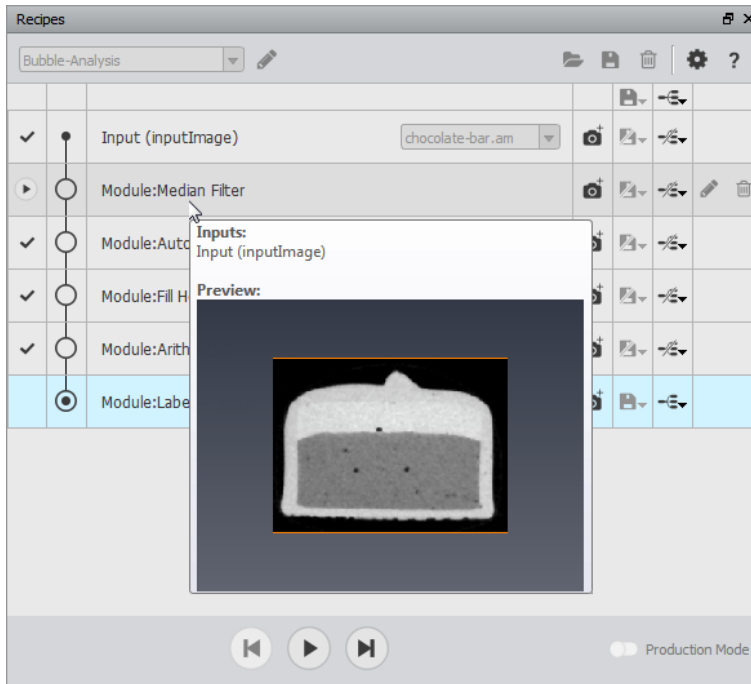
Figure 15.3: Recipe Preview

At each step, it is possible to take a snapshot of its result and save it in the application data directory. By default, the display module defined in *Edit > Preferences > Auto Display* is used. However, you can add an explicit snapshot step with a breakpoint to change it. When executing the recipe, it will pause at this step and allow you to customize the display module in the *Properties* panel (see Figure 15.4).
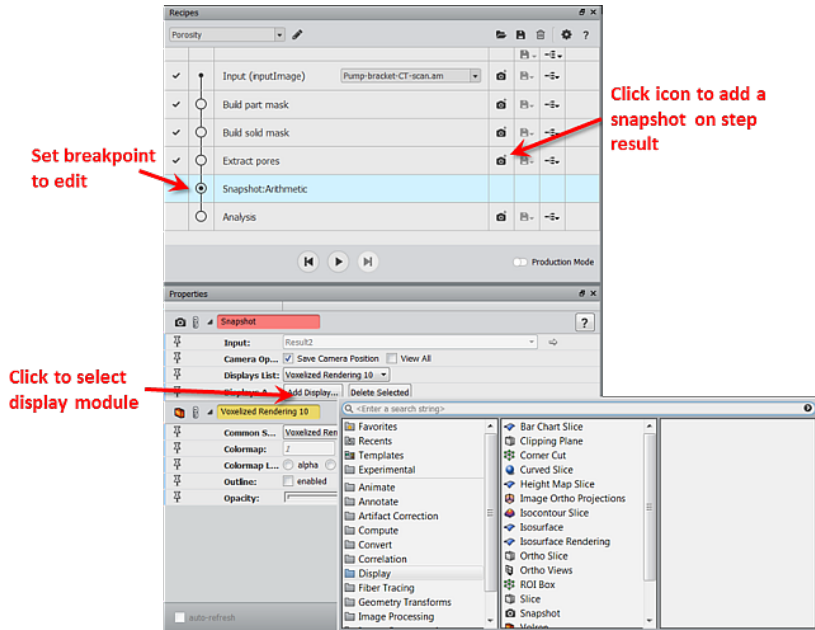
Figure 15.4: Adding a Snapshot

The Avizo XRecipe Extension documentation is organized into the following sections:

- *Creating and editing recipes*
- *Using breakpoints, recomputing another region, and processing full data*
- *Recipes in a TCL command*
- *Recipes Limitations*

15.1 Creating and Editing Recipes

Using a simple example, this section shows you how to create and edit a recipe. In this example, the chocolate bar data set from Avizo's tutorial directory is analyzed with respect to air bubbles inside the candy matrix. We begin by setting up a processing workflow:

- Load AVIZO_ROOT/data/tutorials/chocolate-bar.am
- Attach an *Image Processing > Smoothing And Denoising > Median Filter*. Select *Interpretation 3D*, set *Iterations* to 1 in its *Properties*, and click **Apply**.
- Attach an *Image Segmentation > Binarization > Auto Thresholding* module to *chocolate-bar.filtered* object and click **Apply**.
- Attach an *Image Processing > Separating and Filling > Fill Holes* module to *chocolate-bar.labels* data, select *3D* option, and then click **Apply**.

- Attach a *Compute > Arithmetic* module to *chocolate-bar.filled* data, connect its *InputB* port to *chocolate-bar.labels*. In the *Expression* field, enter **A-B** and then click **Apply**.
- Attach a *Measure And Analyze > Individual Measures > Label Analysis* module to *Result* data (result from previous *Arithmetic* module), connect its *Intensity Image* port to *chocolate-bar.am*, and then click **Apply**.
- Attach an *Ortho Slice* to *chocolate-bar.label* to visualize the result.

Your Project View should now look similar to Figure 15.5.
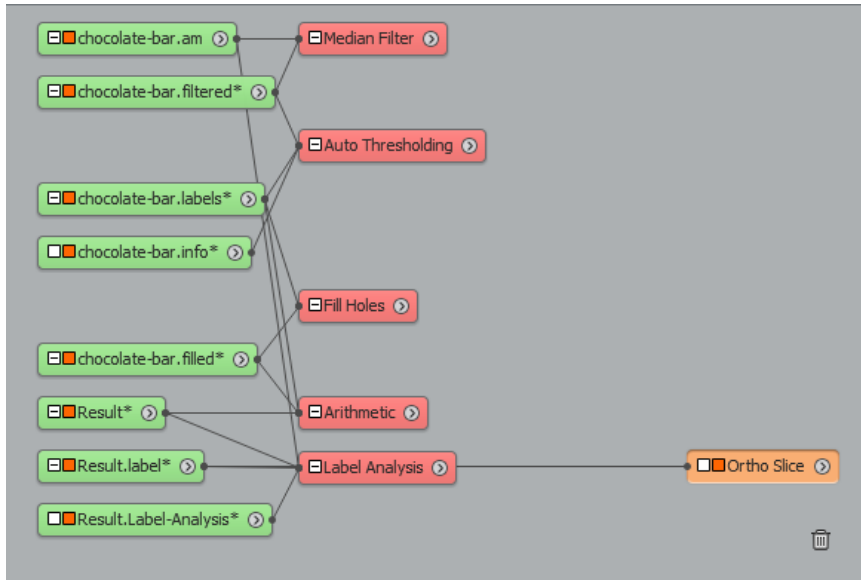
Figure 15.5: Bubble Analysis Workflow

- Attach a *Create Recipe* module to *Result.label* (result from previous *Label Analysis* module), and click **Apply**. Avizo will switch automatically to the *Recipes* workroom (see Figure 15.6).
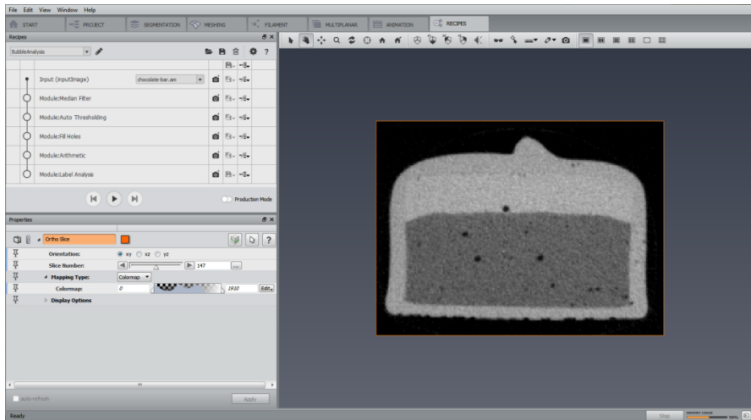
Figure 15.6: Recipes Workroom

Test this recipe and apply it to the raw data again:

- Switch to the *Project* workroom by selecting its icon in the *Workroom Toolbar*.
- Go to *Project > Remove All Objects* from the main menu.
- Load `AVIZO_ROOT/data/tutorials/chocolate-bar.am`
- Switch to the *Recipes* workroom by clicking its tab in the *Workroom Toolbar*.
- Make sure that `chocolate-bar.am` is shown in the *Input* dropdown box of the *Recipes* panel
- Play the recipe on the *chocolate-bar.am*. The recipe will compute statistics as shown in Figure 15.7.

Creating and Editing Recipes   **865**
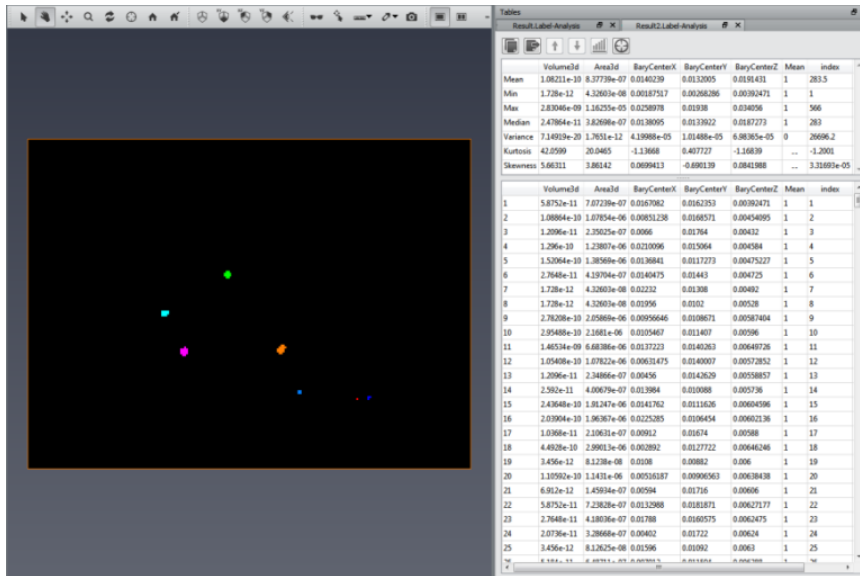
Figure 15.7: Chocolate Bar Bubble Statistics

When you are satisfied with your recipe, save the recipe to be used again and shared.

- In the *Recipes* panel, click **Save Recipe**.
- Use the *File Browser* to navigate to the directory where you want to save your recipe. By default, recipes are stored in `%APPDATA%/FEI/Recipes`, but you may save them to any other location.

15.2   Using Breakpoints

Breakpoints can be used to halt the recipe at a particular step to let you manually change one or more parameters. In this tutorial, you will change the *Measure Group* in the *Label Analysis* module.

- Start Avizo.
- Load `AVIZO_ROOT/data/tutorials/chocolate-bar.am`
- Attach a *Volume Rendering* to `chocolate-bar.am`
- Attach an *Extract Subvolume* module to `chocolate-bar.am` data and extract a small portion of the image (see Figure 15.8 and Figure 15.9).
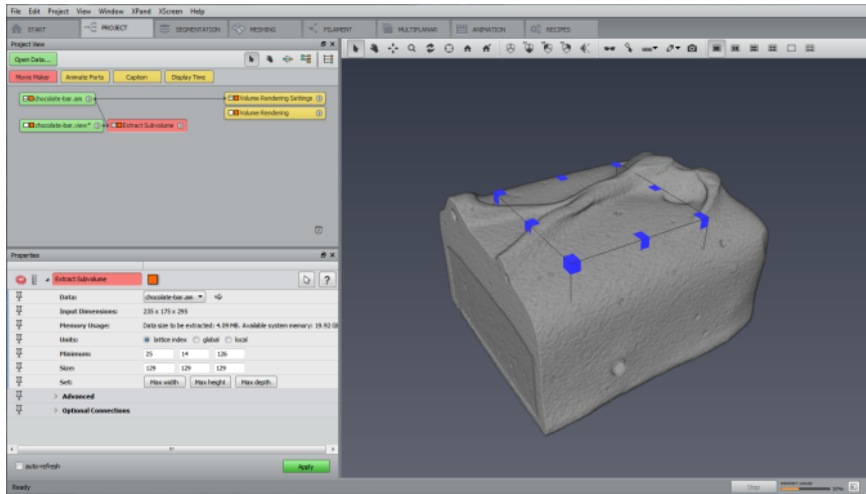
Figure 15.8: Extract Subvolume on *chocolate-bar.am*
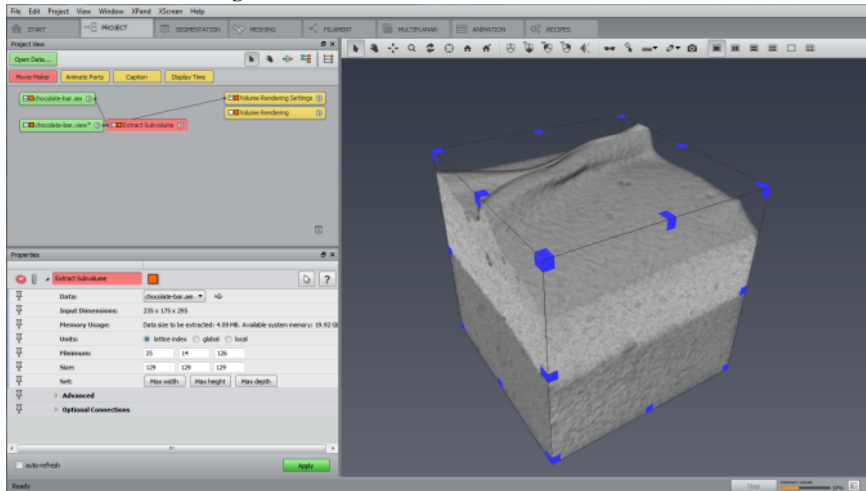
Figure 15.9: Extracted Subvolume

- Load   AVIZO_ROOT/data/tutorials/recipes/Bubble-Analysis.hxrecipe
  from the *Recipes* panel.
- Set the recipe input to the extracted image (i.e.,chocolate-bar.view) and set a breakpoint
  at the *Label Analysis* step (see Figure 15.10).
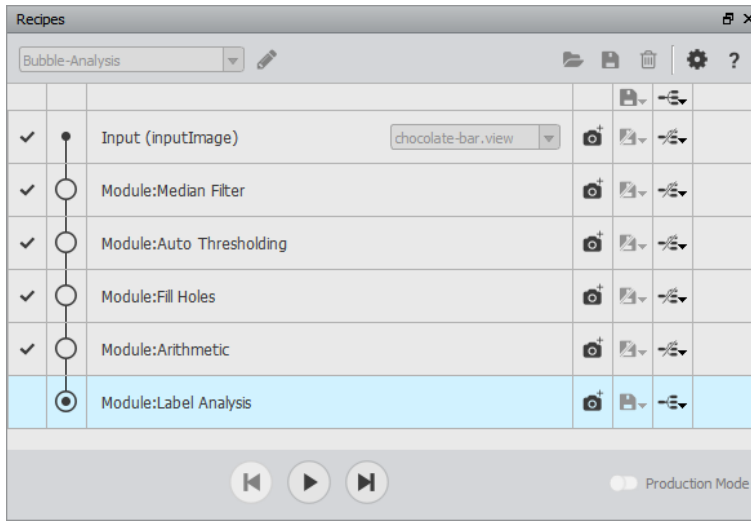
Figure 15.10: Breakpoint at the *Label Analysis* Step

- Run the recipe until it reaches the breakpoint. At this stage, it is not possible to change the input data or to quit the *Recipes* workroom. Only the properties of the tool are available for change. In port *Measures*, select *Standard Shape Analysis*. Click *Play* to continue (see Figure 15.11).
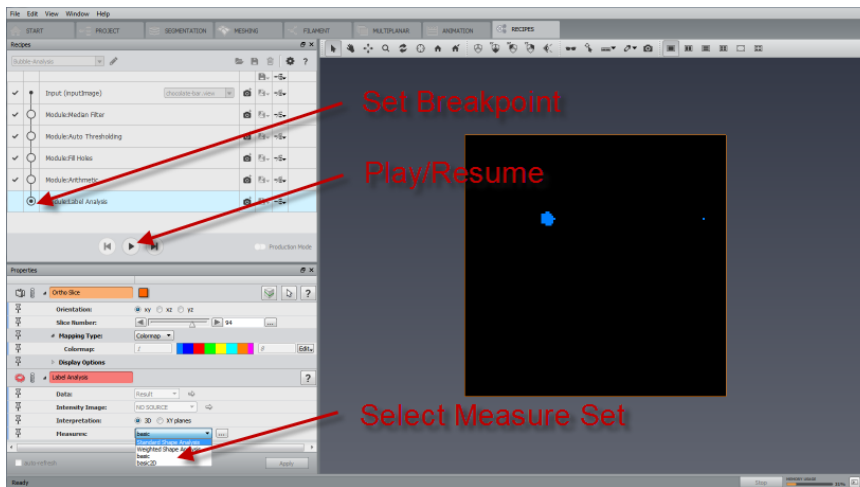
Figure 15.11: Continue a Recipe After Modifications

The recipe described in this tutorial can also be loaded directly from `AVIZO_ROOT/data/tutorials/recipes/Bubble-Analysis.hxrecipe` using **Load Recipe** on the *Recipe* panel.

15.3   Recipes in a TCL Command

When recipes have been saved on disk, they can be replayed from a TCL script. The following TCL commands are available:

- Set a recipe: `theRecipesController setCurrentMacro recipeName`
- Set an input to the recipe: `theRecipesController setInput numInput currentInput`
- Play the recipe: `theRecipesController playMacroNoPause`
- Get the last result of the recipes: `theRecipesController getLastResult`
- Enable/disable production mode: `theRecipesController setProductionMode value [0: off | 1: on]`

15.4   Recipe Limitations

There are some known limitations to the recipe mechanism:

- Step Deletion

    - You can only delete the first step or last step of a recipe except snapshot steps that you can always delete.
    - If a recipe's last step has more than one input, then the step cannot be deleted. Deleting the step would imply having more than one recipe result as the deleted step inputs would become the new outputs. For now, the recipe mechanism can only handle one final recipe result.
    - If only one step remains in the recipe, it cannot be deleted.
      Example: A step with a module like *Or Image* cannot be deleted if it is the last step.
    - Input steps cannot be deleted.

- Step Insertion

    - For now, only snapshot steps can be inserted in a recipe.

- Default Display

    - The *Ortho Views* module cannot be used since there is only one instance of this module.

- Clean of recipe intermediate data

    - When an intermediate result of a recipe has a connection to a previous intermediate result, this last result must be explicitly exported to the Project workroom to not be removed by

recipe cleaning process.

Example: This issue can appear when doing consecutively a *Generate Surface* step (generates a surface data) and then a *Surface Normals* step (generates a vector field data from the surface). The vector field data will imperatively have a connection to the surface data as it cannot exist without it. However the current recipe mechanism cannot detect such dependencies between data. So, the surface data may be removed by the automatic recipe cleaning process whereas it is still needed in a next step via the vector field. To avoid this kind of issue, in this example, the *Generate Surface* step result must be explicitly exported to the Project workroom, ensuring it to not being removed by the recipe cleaning process.

Chapter 16

Avizo XReporting Extension

Avizo allows for the creation of reports. This is performed through the Reporting Workroom (see Figure 16.1). From this room, report templates can be loaded. A report template is an HTML page with frames. The Reporting Workroom allows for the user to associate snapshot and statistical results to the HTML template. This can be done automatically by assigning a particular snapshot or table generated when running a recipe to a specific frame; or by dragging-and-dropping a snapshot generated through the Avizo snapshot tool, or a table resulting from an analysis.
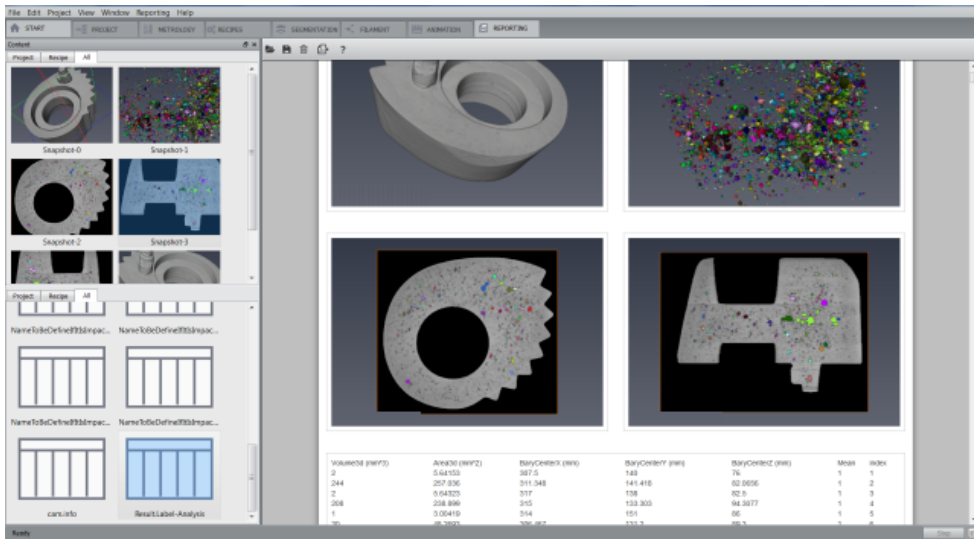
Figure 16.1: The Reporting Workroom

Any HTML editor can be used to create a template suitable for the Reporting Workroom of Avizo. The

page should consist of frames and static elements, such as text or images. Each frame containing the class="reportContentContainer" will be able to receive a snapshot or a table from an Avizo session. The Reporting Workroom is not designed to be an HTML viewer, it is designed to load a template, populate a template during the session, and save a report. A saved report in HTML can be viewed with any browser or via PDF with any PDF reader. Default report template provided with Avizo is loaded when switching to the Reporting Workroom. From there, this default report can be used or a new one can be loaded (see Figure 16.2).
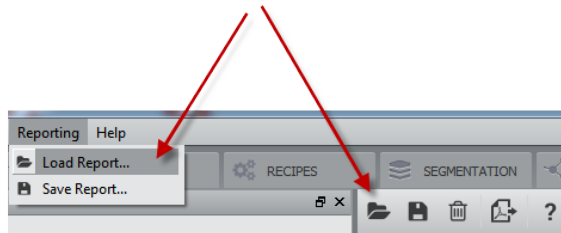
Figure 16.2: Loading a New Template

The *Content* panel of the Reporting Workroom is split in two sub-panels. The top one lists the available snapshots that can be imported to the template and the bottom one lists the tables. There are two ways of populating these sub-panels. The table sub-panel will be automatically populated when a spreadsheet resulting from any analysis is produced, such as a Label Analysis module. For the snapshot panel, snapshots can be interactively generated from the Avizo snapshot tool where the Reporting Workroom can be selected as destination for the snapshot (see Figure 16.3).
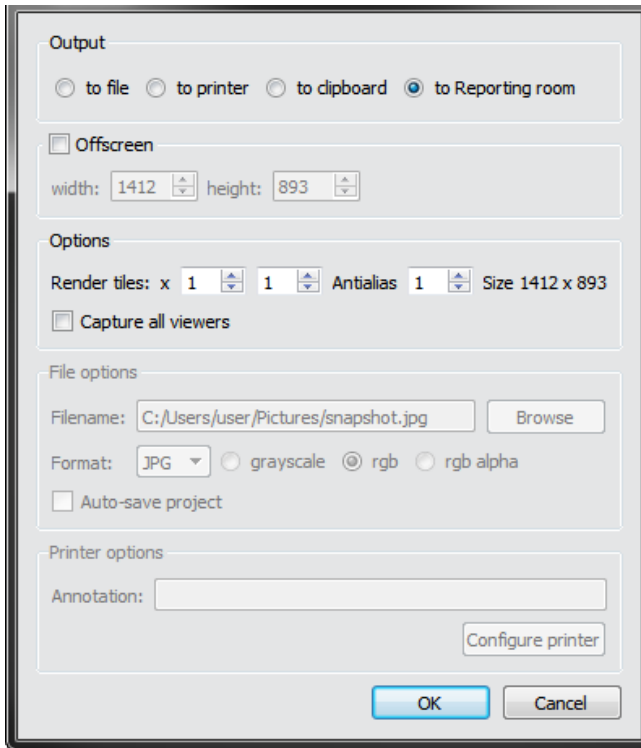
Figure 16.3: The Avizo Snapshot Tool

- Load AVIZO_ROOT/data/tutorials/spring_loaded_cam.am and attach any visualization modules to it.
- Select the snapshot tool and send a snapshot to the Reporting Workroom.
- Switch to the Reporting Workroom. The snapshot appears in the top sub-panel.
- Drag-and-drop the snapshot from the sub-panel into one of the frames of the report template.
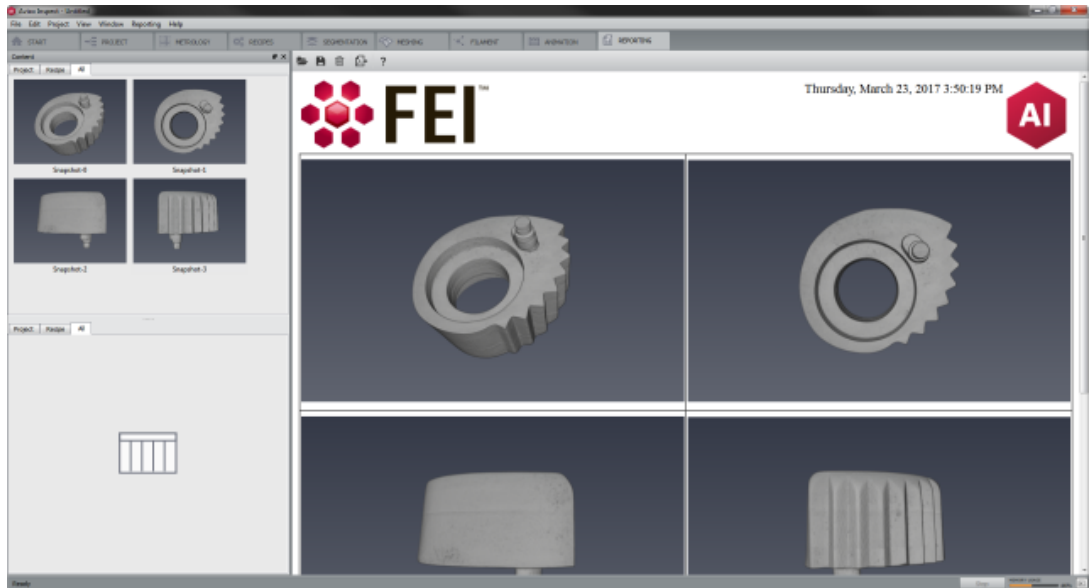
Figure 16.4: Drag-and-Drop of Snapshots

The same way, tables resulting from analysis will be sent to the Reporting Workroom where they can be dragged-and-dropped into a frame of the template. The second way is by using recipes. A recipe may automatically produce snapshots and tables resulting from analysis. Each snapshot and table produced by a recipe is sequentially identified and will populate the sub-panels in the same order when played. Replaying a recipe will replace in the same order the snapshots and tables in the two sub-panels.

1. Load `AVIZO_ROOT/data/tutorials/spring_loaded_cam.am`.
2. Switch to the *Recipes Workroom*.
3. Load the `AVIZO_ROOT/data/tutorials/recipes/porosity.hxrecipe`.
4. Add a snapshot step between each step.
5. Play the recipe.
6. Switch to the Reporting Workroom. The snapshots and tables appear in the sub-panels. Note that when loading a recipe, Avizo automatically detects the snapshots generated by the recipe, and the snapshot sub-panel is automatically populated with *empty* snapshot placeholders. This is not true for tables since the recipe must be played once for Avizo to determine which tables are generated (see Figure 16.5).
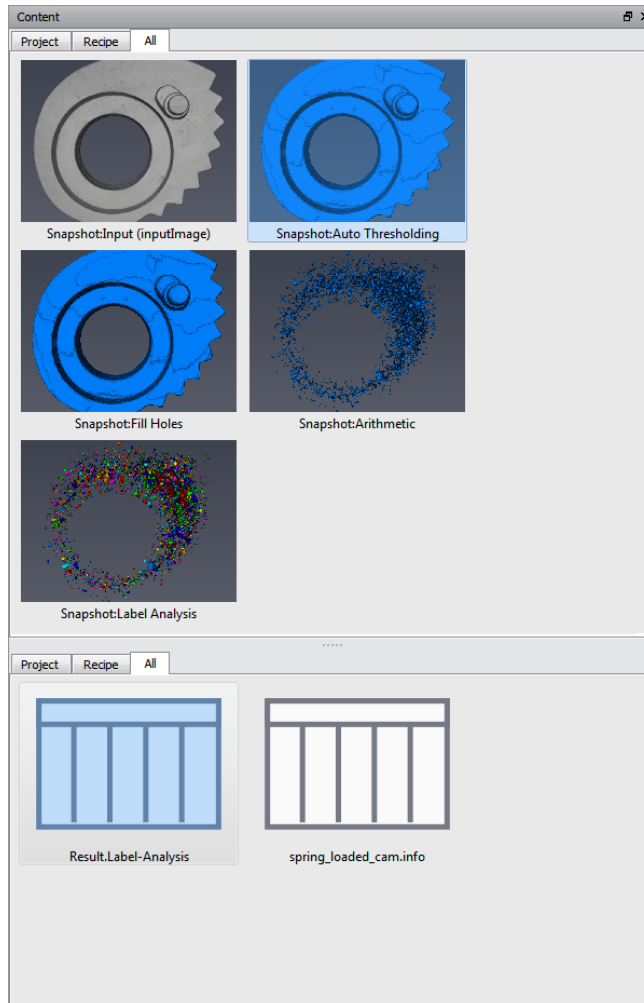
Figure 16.5: Snapshots and Table from Recipe

7. Snapshots and tables can then be dragged-and-dropped to each frame of the template. Once this association is complete, the snapshots and tables will be automatically updated in the template each time the recipe is played. Once the association has been set, the template can be saved as a new template. This new template can be reloaded in Avizo and will be populated automatically each time the associated recipe is played (see Figure 16.6).
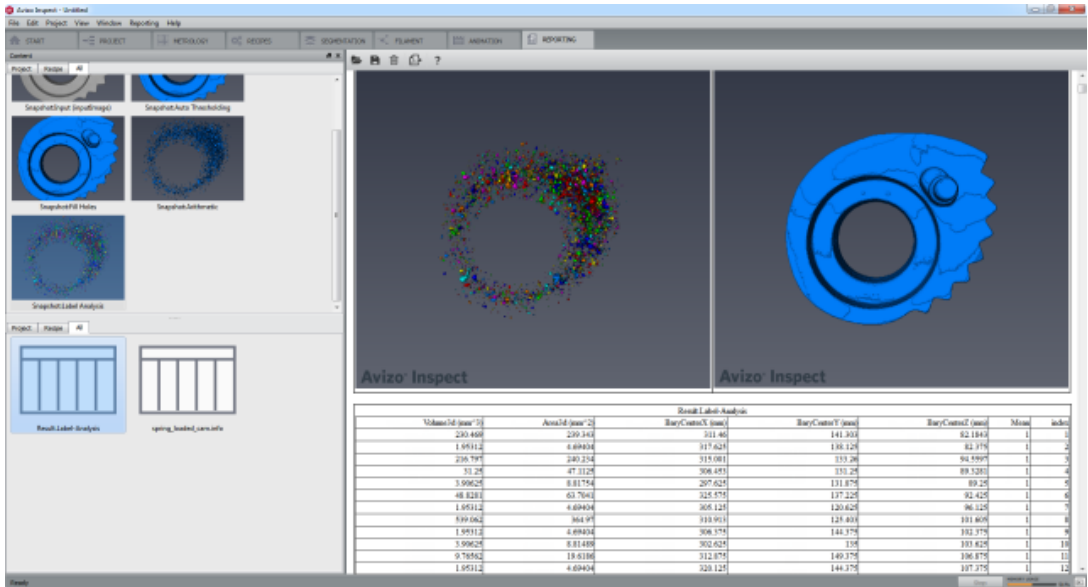
Figure 16.6: Report Example

8. To remove a snapshot or a spreadsheet from the report, click on it and press *Delete*.

The report can then be saved as an HTML page or exported to PDF (see Figure 16.7).
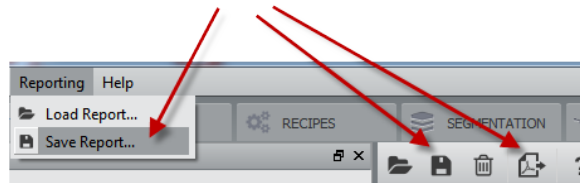
Figure 16.7: Saving or Exporting a Report

A report can be filled using an Avizo script. To create such a report, set a unique HTML tag in each frame (it is recommended using: `accesskey="key"`).
To set a snapshot in a given frame, use the following command line:

```
theReportingWorkroom insertSnapshot "Snapshot-#" "td\[accesskey=\"key\"\]"
```

To set a spreadsheet in a given frame, use the following command line:

```
theReportingWorkroom insertSpreadsheet "sheet_name" "td\[accesskey=\"key\"\]"
```

To save a report, use the following command line:

```
theReportingWorkroom saveReport "C:/Path_to_your_file"
```

To clean the report, use .

To export the report in PDF, use .

Chapter 17

Avizo XPoreNetworkModeling Extension User's Guide

The Avizo XPoreNetworkModeling Extension is required to play the tutorial and unlock the features described in this chapter. It allows for access to different statistics from a labeled and separated pore space. The statistics include distribution of the following parameters:

- Pore volume
- Pore area
- Pore equivalent radius
- Pore center of gravity
- Pore coordination number (i.e., number of connected neighbors)
- Intersection percentage between pore network model and original pore space
- Throat area
- Throat equivalent radius
- Throat channel length
- Throat connection (i.e., Id of pore 1, Id of pore 2)

The extension also allows for pore network code reading from the *Pore Network Node-Link DATA FORMAT*.

The general workflow for accessing statistics includes the following:

- Create a binary pore space from the grayscale data (see *Segmentation Workroom*)
- If connected porosity needs to be analyzed, unconnected pores can be removed with the *Axis Connectivity* module
- Separate the pore space into a set of connected and labeled pores using the *Separate Objects* module. The extension provides a mode optimized for arbitrary pore shapes (i.e., Skeleton -Aggressive). Spherical pores can be well separated with the default mode (i.e., Chamfer -

conservative).

- Generate a pore network model via the *Generate Pore Network Model* to extract network code.

The generated *pore network model (PNM)* contains the network code and can be used with the *Histogram* to plot the distribution of the different parameters. To learn more about the tools available in the extension, refer to the following links:

- *Pore Space Analysis Tutorial*
- *Generate Pore Network Model*
- *Pore Intersection*
- *Pore Network Model Filter*
- *Pore Network Model View*
- *Separate Objects*
- *Pore Network Node-Link*

Note: It is also possible to visualize the model on top of the original data. Figure 17.1 is a setup showing the labeled and separated pore space overlaid on top of the model and the model within the grayscale data.
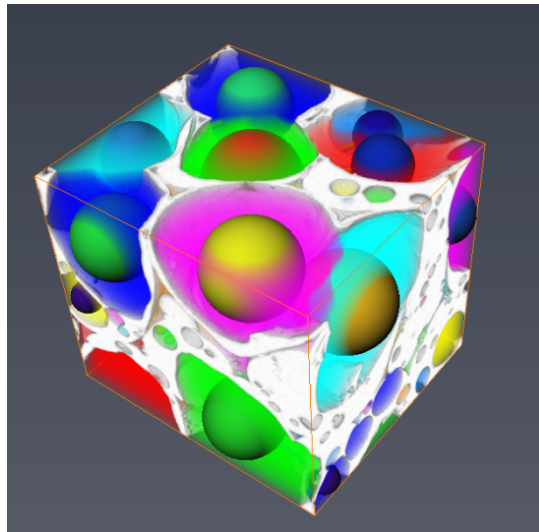
Figure 17.1: PNM Displayed within Pore Space

The properties of a PNM generated with the *Generate Pore Network Model* can optionally be provided by the same module. The Generate Pore Network Model module will then also outputs:

- The absolute permeability of the material
- The tortuosity of the material

- The flow rate per second of the fluid passing through each throat
- The total flow rate per second used to calculate the absolute permeability
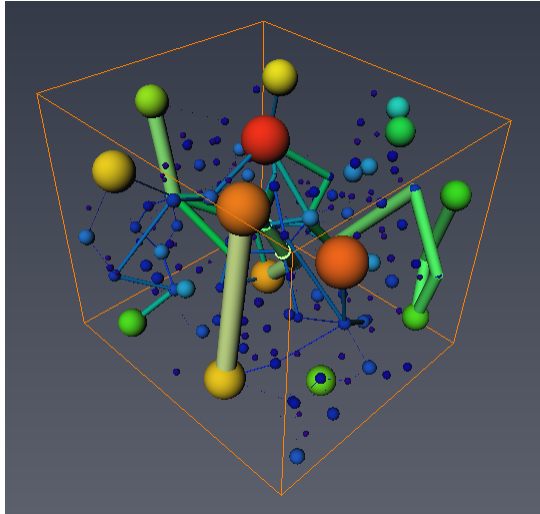
Figure 17.2: PNM Displayed with Throat Scale based on Flow Rate per Second

17.1 Pore Space Analysis

This tutorial describes how to analyze and characterize a pore space. The ceramic data used for this tutorial is a courtesy of Zellwerk GmbH.

17.1.1 Preparing the Data

In this section, we will prepare the data to separate the void and the solid phase of the ceramic.

1. Load the file data/tutorials/PNM/Sponceram.am and display it using the *Volume Rendering* module. Set the Colormap port max to *4590* to have a better rendering (see Figures 17.3 and 17.4 with colormap configured to hide the void phase).

2. Separate the void of the material using an *Auto Thresholding* module. Set the Type port to *Auto Threshold Low*. It will create two outputs Sponceram.info and Sponceram.labels (see Figure 17.5).

3. Apply an *Axis Connectivity* module on the Sponceram.labels. Select a Neighborhood of 26 voxels in the neighborhood port. This allows keeping only the connected porosity of the sample (see Figure 17.6).
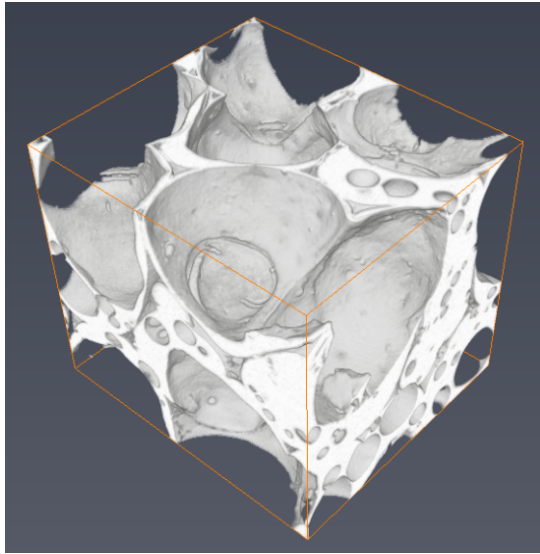
Figure 17.3: Original Data Display
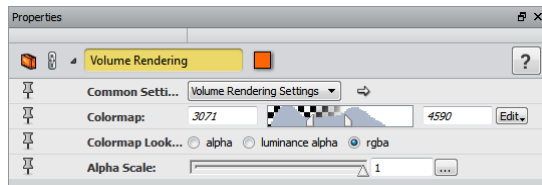
Figure 17.4: Volume Rendering Module Configuration
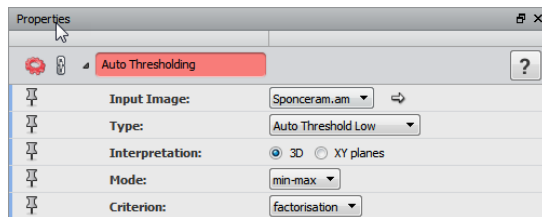
Figure 17.5: Auto Thresholding Module Configuration

4. Compute the connected porosity vs. total porosity using a *Volume Fraction* module connected to the output of the axis connectivity module (for amount of connected porosity), or connected to the output of the *Auto Thresholding* module (i.e., for amount of total porosity), respectively.
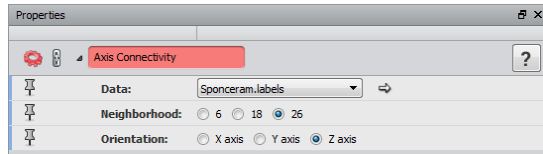
Figure 17.6: Axis Connectivity Module Configuration

This example reports 51% of connected porosity for a total porosity of 54%.

5. To visualize the connected porosity vs. unconnected porosity: attach a *Subtract Image* module to the output of the *Auto Thresholding* module and to the output of the *Axis Connectivity* module. This will create a label representing the unconnected porosity (floating void). Attach a *Volume Rendering* to this data, and use the seismic colormap. This colormap will colorize in blue the data at 0 (connected porosity) and in red the data at 1 (unconnected porosity) Set the opacity to 0.2: it allows better visualization of the solid phase. (see Figure 17.7).
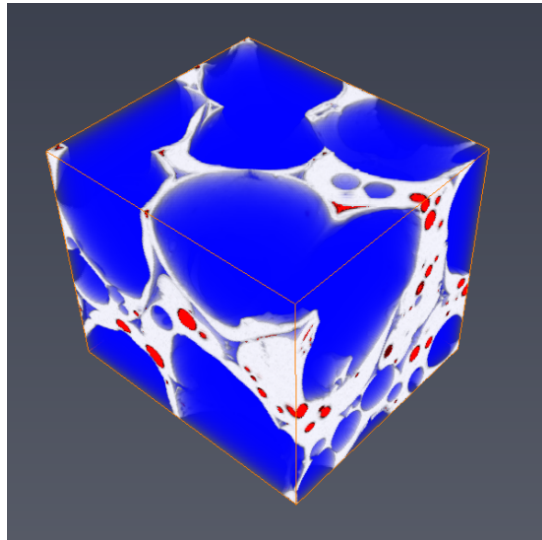
Figure 17.7: Connected Porosity vs. Unconnected Porosity

6. Apply a *Separate Objects* module on the output of the *Axis Connectivity* module. Set the Marker Extent port value to *2* and the Output Type port to *connected object*. In this example, we keep the default method (Chamfer - Conservative) since the pores are mostly spherical (see Figure 17.8).

7. Display the result by creating another *Volume Rendering* module on the previous output (see Figures 17.9 and 17.10).

Pore Space Analysis **882**
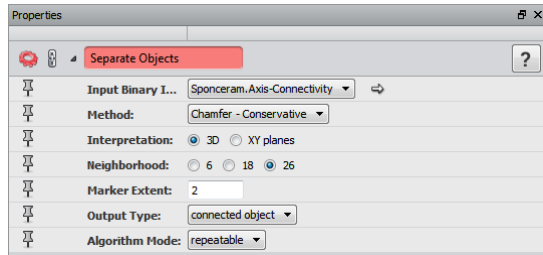
Figure 17.8: Separate Objects Module Configuration
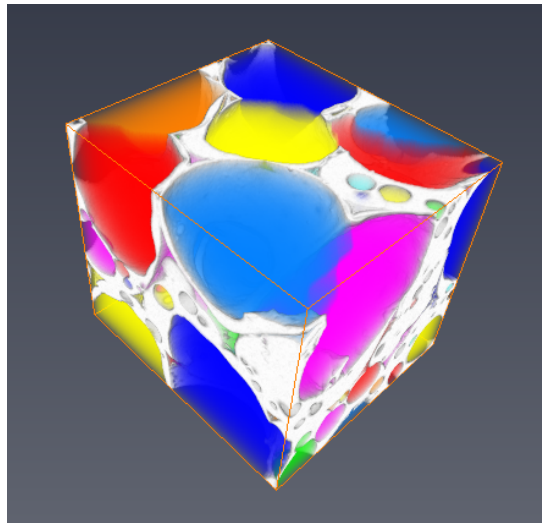
Figure 17.9: Void Data Mixed with Original Data
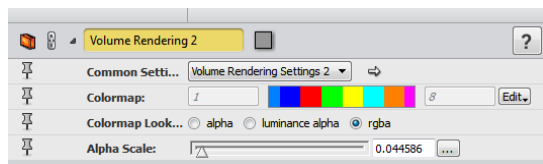
Figure 17.10: Volume Rendering Module Configuration for Void

The project data/tutorials/PNM/1-PrepareData.hx allows jumping directly to this step.

17.1.2 Generating and Analyzing the Network

1. Generate a *PNM* using the *Generate Pore Network Model* module. Set the input to the separated and labeled pore space.
   You can also turn on the *Generate Properties* switch to generate the *absolute permeability*, *tortuosity*, *total flow rate per second* (used to compute the absolute permeability) and *flow rate per second* for each throat. This last one will be added in the *PNM* generated by the module, whereas the others will be available in a dedicated spreadsheet.

2. Create a *Pore Network Model View* on the new network and hide the *Volume Rendering* module linked to the separated data. You can now visualize the network in the original data (see Figures 17.11, 17.12, and 17.13).
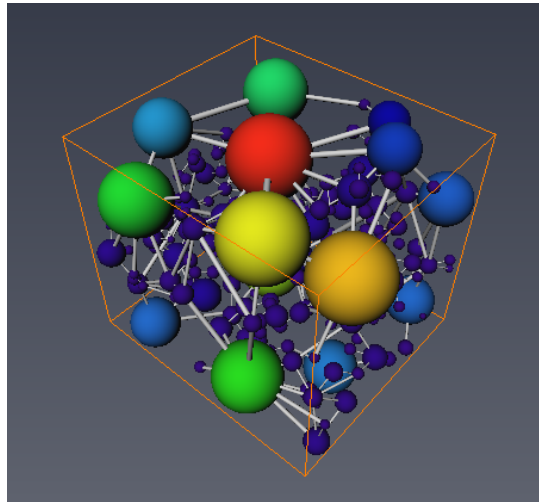
Figure 17.11: Pore Network Model View
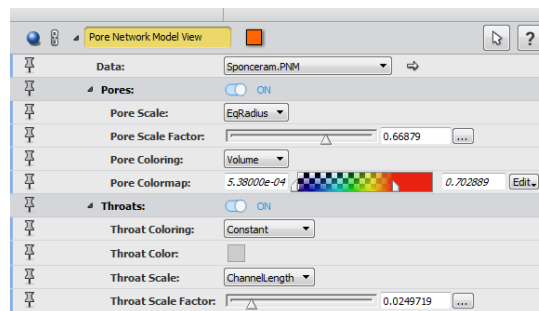
Figure 17.12: Pore Network Model View Parameters
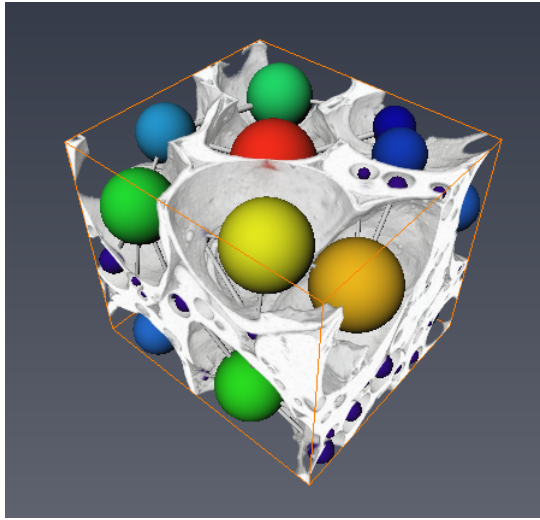
Figure 17.13: Pore Network Model View Mixed with Matter

If you have experience generating the properties of the *PNM* as well, you can then visualize the network with Throat Scale based on the *FlowRatePerSec* value for instance. The purpose is to provide a view that highlights the "main roads" of the fluid passing through the material (see Figures 17.14 and 17.15).
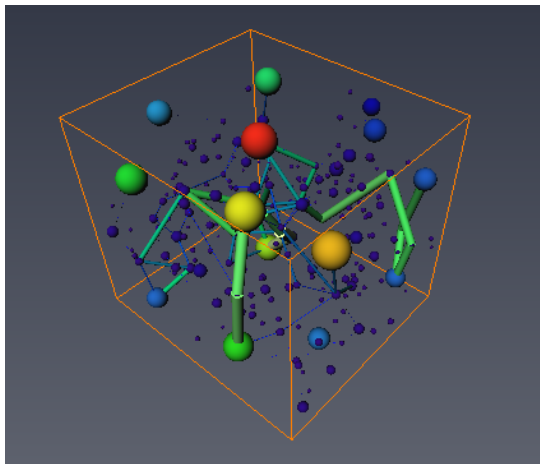
Figure 17.14: Pore Network Model View based on Total Flow Rate

3. The network code can be displayed as a spreadsheet in the table panel by clicking **Show** of any
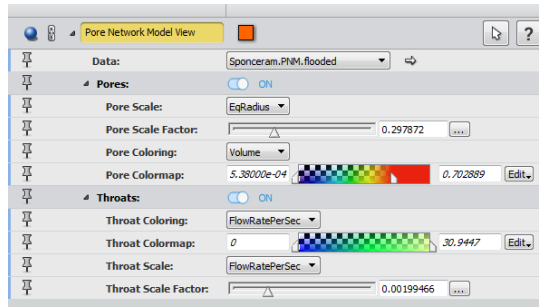
Figure 17.15: Pore Network Model View based on total flow rate Parameters

PNM. The PNM spreadsheet has two tabs, one for pores and one for throats (see Figure 17.16).

Figure 17.16: PNM Code Displayed as a Spreadsheet

4. When displaying the PNM with a *Pore Network Model View* and displaying the spreadsheet at the same time in the table view, it is possible to highlight the pores or the throats that are selected in the table from the Pores table or the Throats table. Select pores in the table by left-clicking on one or several lines (i.e., left click + hold on a line and move up or down the table without releasing). The corresponding pores will be highlighted in red in the 3D view (see Figure 17.17).

5. The generated PNM can be used to plot the statistics of the pore space (see Figures 17.18 and 17.19).

6. It is possible to plot the pore size distribution of the sample, using the *Distribution Analysis* module (i.e., equivalent radius vs. cumulated pore volume within a bin).

   - Attach a *Distribution Analysis* module to the generated PNM.
   - Set the parameters as shown in Figure 17.20 and click **Apply**.
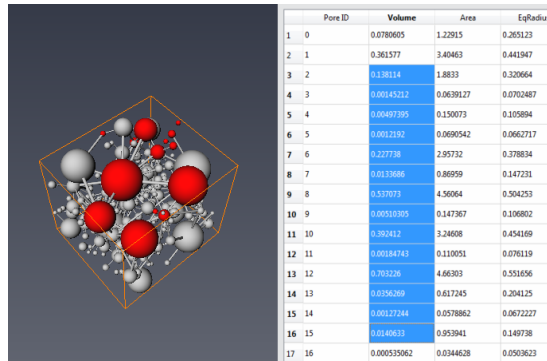
Figure 17.17: Highlighted Pores
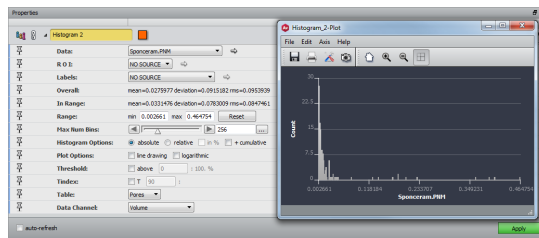
Figure 17.18: Pore Volume Histogram
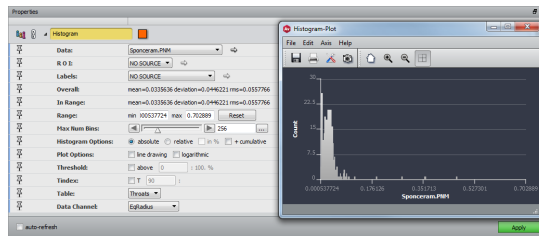
Figure 17.19: Throat Equivalent Radius Histogram

- The distribution curve as shown in Figure 17.21 is displayed using a *Plot SpreadSheet* module (see Figure 17.22). The plot shows that the majority of the pore space is filled by big pores, but the predominant small pores have a radius around 0.15.

7. Filter the data to keep only pores with volume greater than 0.02. Create a *Pore Network Model Filter* module on the generated PNM and then add *Volume > 0.02* in the Filter port. You can visualize the filtered network by changing the Data port of the Pore Network Model View and setting it to the new network (see Figures 17.23 and 17.24).
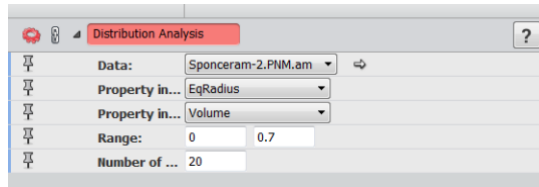
Figure 17.20: Distribution Analysis
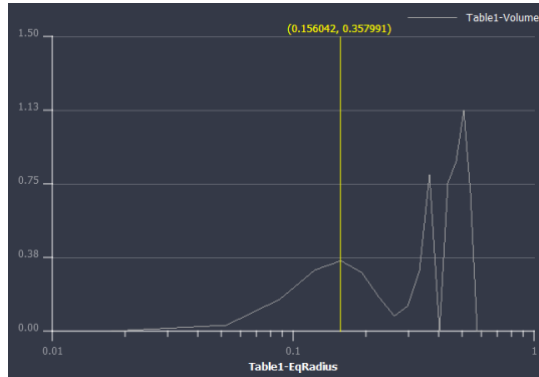
Figure 17.21: Pore Size Distribution
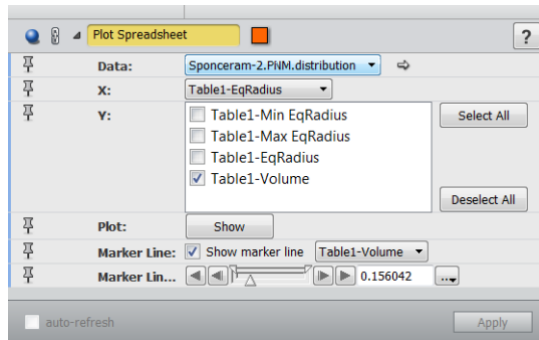
Figure 17.22: Plot SpreadSheet

8. Compute the intersection percentage of the pores for the filtered network. Create a *Pore Intersection* module on the last result and connect the result of the *Separate Objects* module to the Binary Image port. The module will output a new pore network code, including an intersection column that represents the percentage of intersection with the real pore space (see Figures 17.25 and 17.26).
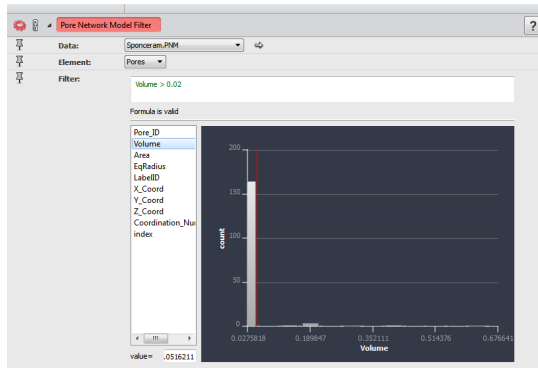
Figure 17.23: Parameters of the Pore Network Model Filter Module
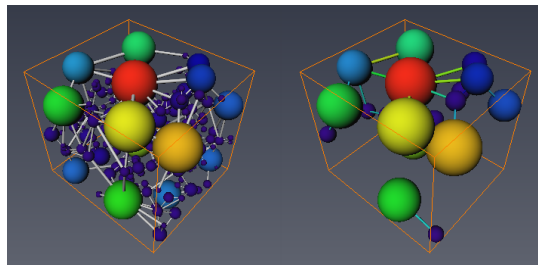
Figure 17.24: Comparison before Filtering, and after Filtering
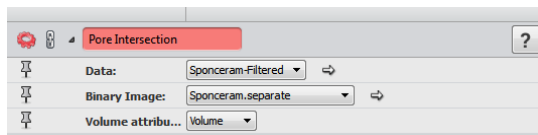
Figure 17.25: Parameters of the Pore Intersection Module

	Pore ID	Volume	Area	EqRadius	LabelID	Intersection	X Coord	Y C
1	0	0.0780605	1.22915	0.265123	1	75.1842	3.66852	1.33587
2	1	0.361577	3.40463	0.441947	2	77.4941	2.5525	1.47724
	2	0.120114	1.0022	0.220664	2	60.4022	2.02002	1.65015

Figure 17.26: Pore Intersection Spreadsheet

The project data/tutorials/PNM/2-GeneratePNM.hx allows reproducing all the steps of the tutorial.

Note: Further pore space analysis can be done with the *Sieve Analysis* module (which allows

for the classification of pores based on statistics), or with the Avizo XLabSuite Extension to calculate the actual absolute permability of the material. In the latter, the Avizo XLabSuite Extension provides you a different method to compute the absolute permeability: it runs a full fluid simulation, whereas the *Generate Pore Network Model* module uses a mathematical modelization and approximation.

Chapter 18

Avizo XBioFormats Extension

About Bio-Formats

Bio-Formats is a popular Java library for handling life science image data (http://www.openmicroscopy.org/site/products/bio-formats). The Bio-Formats integration enables Avizo to read image and metadata from over 140 file formats. For a list of all supported file formats and additional information, please visit: http://www.openmicroscopy.org/site/support/bio-formats5.1/supported-formats.html. To facilitate data exchange between different software packages and organizations, Bio-Formats converts data stored in proprietary file formats into an open standard called the *OME Data Model*, in particularly the OME-TIFF file format.

Bio-Formats Metadata

Bio-Formats categorizes metadata in three tiers:

- **Core metadata**: This is information necessary to understand the basic structure of the images including image dimension, number of focal planes, time points, channels, byte order, dimension order, color arrangement (i.e., RGB, indexed color or separate channels), and thumbnail dimensions.
- **Original metadata**: This is information specific to a particular file format. These fields are key/value pairs in the original format with no guarantee of cross-format naming consistency or compatibility. The nomenclature often differs between formats since vendors are free to use their own terminology.
- **OME metadata**: This is information from Core metadata and Original metadata converted by Bio-Formats into the OME data model. Performing this conversion is the primary task of Bio-Formats.

Bio-Formats Integration

The integration into Avizo then organizes the opened image and metadata into the native Avizo data structure with its well-known parameter bundles. During this process, standard image information

(e.g., voxel size, matrix dimensions etc.) is assigned to the corresponding standard Avizo parameter bundles. All metadata is stored in a dedicated Bio-Formats parameter bundle. The Bio-Formats integration is able to open data as scalar-, color-, and multi-channel fields as well as tiled images and time series.

The Bio-Formats library is seamlessly integrated into Avizo. Avizo prioritizes its own format implementation over Bio-Formats, so if a file format is known by Avizo, the native Avizo reader is used for performance reasons. Reading such a file with Bio-Formats requires opening it using the **Open Data As...** file dialog box and selecting Bio-Formats as the preferred reader. Files that Avizo does not have a native reader for are automatically loaded using Bio-Formats.

Is it also possible to directly *convert large data* sets that may not fit into the system memory to Avizo's *LDA multi-resolution format*. This direct conversion will create individual LDA files for each 3D volume, e.g. a time series with 10 time steps will result into 10 LDA files loaded as a time series into Avizo. This will allow visualization and *sub-volume extraction* for further processing from data that is larger than the system memory. The conversion can be initiated using the Avizo's LDA dialog that opens if the data size exceeds a threshold specified in the *Preferences Dialog*, or using the *Convert to Large Data Format* module.

Note: Currently, settings for multi-channel time series such as color map & range per channel cannot be stored in Avizo project files.

Commands

Caution: Bio-Formats reader is part of a separate library. When you use the TCL command directly, this will require you to use the following command to manually load the separate library:

```
dso open hxbioformatreader
```

The following TCL commands can be used to parametrize the Bio-Formats reader.

```
bioFormats read [-series <seriesId>] [-time <timeId>] [-channel
<channelId>] [-xmin <xmin> -xmax <xmax> -ymin <ymin> -ymax <ymax>
-zmin <zmin> -zmax <zmax>] [-filledRGB <status>] <fileName>
```
This command reads an image file `<fileName>` using an absolute or relative path.

- It is possible to read a specific series number using the `-series` option. If the `-series` option is not defined, the default setting is to read all the series.
- It is possible to read a specific time step using the `-time` option. If the `-time` option is not defined, the default setting is to read all the time steps.
- It is possible to read a specific channel number using the `-channel` option. If the `-channel` option is not defined, the default setting is to read all channels.
- It is possible to read a specific volume using the bounding box parameters (in pixels) `-xmin, -xmax, -ymin, -ymax, -zmin` and `-zmax`. If these optional parameters

are not defined, the default setting is to read the volume max of the file.

- It is possible to force to fill the RGB color channels using the `-filledRGB` option (`status=1`: Force to fill, `status=0`: Do not fill). If the `-filledRGB` option is not defined, the default setting is to do not fill RGB color channels if it is not necessary.

The command returns the labels of the data objects created in the Project View. **Note**: All the metadata that Bio-Formats reader gets from the file are copied as parameters of the created object.

`bioFormats canRead <fileName>`
This command checks if the image file can be read.
The command returns the value 1 if the `<fileName>` can be read, otherwise 0.

`bioFormats format <fileName>`
This command asks Bio-Formats the file format of the `<fileName>`.
The command returns the format (as a string) detected by Bio-Formats.

`bioFormats info [-series <seriesId>] <fileName>`
This command reads the image file information (all the metadata Bio-Formats gets). It is possible to get the metadata of a specific series number using the -series option. If the -series option is not defined, the default setting is to get the metadata of all the series successively.
The command returns a list of *key:value* pairs available.

`bioFormats getInfo [-series <seriesId>] -param <parameter>`
`<fileName>`
This command gets the information of the `<parameter>` for the ¡seriesId¿ image file. If the `-series` option is not defined, the default setting is to read only the series 0. If the `<parameter>` is found several times in the meta data dictionaries (Basics, Extended), the list of values will be returned.
The command returns the information of the `<parameter>`.

`bioFormats omexml <fileName>`
This command reads the available or populated OME XML header file.
This command returns the metadata associated to presented as an XML tree. The XML format is OME-XML.

`bioFormats getSeriesCount <fileName>`
This command gets the series count associated with the image file.
This command returns the series count.

`bioFormats getDims [-series <seriesId>] <fileName>`
This command gets the data dimension {SizeX SizeY SizeZ SizeC SizeT} of the `<seriesId>`. If the `-series` option is not defined, the default setting is to read all the series.
This command returns the list of the data dimension.

Environment Variables:
The -Xms and -Xmx command line options are available to change the behaviour of JRockit JVM to better suit the needs of the Bio-Formats reader Java application.

The -Xms option sets the initial and minimum Java heap size. The default allocated memory is 256MB. To change the size, add the following environment variable and set the new size:
HX_JAVA_HEAP_SIZE_MIN=`<size>[g|G|m|M|k|K]`

The -Xmx option sets the maximum Java heap size. The default allocated memory is 64GB. To change the size, add the following environment variable and set the new size:
HX_JAVA_HEAP_SIZE_MAX=`<size>[g|G|m|M|k|K]`

Chapter 19

Avizo XDigitalVolumeCorrelation Extension User's Guide

The Avizo XDigitalVolumeCorrelation Extension is required to unlock the tutorials and features described in this chapter.

The Avizo XDigitalVolumeCorrelation Extension provides Digital Volume Correlation (DVC) techniques to compute 3D full-field continuous displacement and strain maps from volume images acquired during a deformation process of an object. You will input two images and a mesh that will be used to measure the displacement and strain maps.

The extension can for instance be used to visualize and quantify deformation-induced microstructural changes during dynamic processes, such as localization phenomena induced by heterogeneities or thermal expansion mismatch between materials. Furthermore, the output displacement field can be used to enrich a numerical simulation by using measured boundary conditions, or to optimize this simulation by comparing numerical and measured data.

To learn more about the tools available in the extension, please refer to the following links:

- *Digital Volume Correlation Module*
- *Digital Volume Correlation Analysis*

Note: see section 1.4 System Requirements about system requirements and hardware platform availability.

Acknowledgements

Avizo XDigitalVolumeCorrelation Extension has been developed in cooperation with the companies 3Dmagination (Didcot, UK) and Eikosim (Paris, France).

19.1 Digital Volume Correlation Analysis

This tutorial describes how to perform a DVC (Digital Volume Correlation) analysis and visualize the corresponding displacement and strain fields. A subset-based (local) approach is used to capture the *large* displacements on a *coarse* regular grid and the resulting data are used to initialize a most robust Finite-Element-based (global) DVC technique on a *finer* mesh. The bone-cement data used in this tutorial is a courtesy of Dr Gianluca Tozzi at University of Portsmouth and represents one of the possible configurations of interdigitated bone structures in joint fixation [1].

To follow this tutorial, you should be familiar with the basic concepts of Avizo such as interaction with the 3D viewer, segmentation editor, etc. All these topics are discussed in Avizo*Getting Started* chapter. **Note:** see section 1.4 System Requirements about system requirements and hardware platform availability.

19.1.1 Preparing the Subset-Based Approach

In this section, you will compute the characteristic length scale of the microstructure to tune the subset size of the *Digital Volume Correlation* module with a sensible value.

- Load the file data/tutorials/xvolumecorrelation/BCI_reference.am
- Display it using an *Ortho Slice* module and set the *Orientation* port to *xz*.
  The microstructure consists of pure trabecular bone (white) interdigitated with cement (grey).
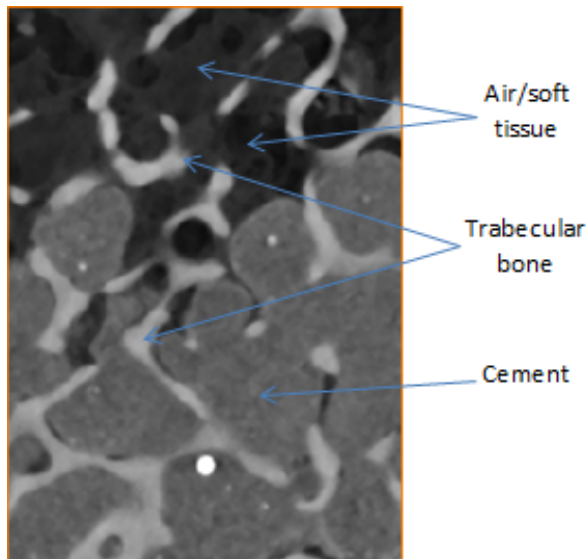
Figure 19.1: Dataset Description

- Extract a region of interest in the bone-cement using an *Extract Subvolume* as shown in Figure
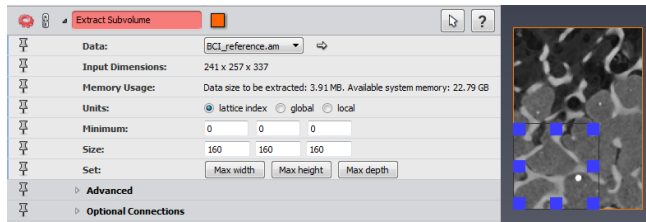
Figure 19.2: Extract Subvolume Module Properties

- Compute the length scale of the microstructure using the *Radial Autocorrelation* module. This module calculates the two-point correlation function of the tomographic image.
- Attach a *Plot Spreadsheet* module as shown in Figure 19.3
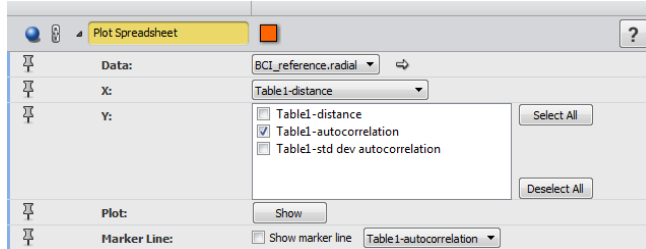
Figure 19.3: Plot Spreadsheet Module Properties

- Plot the autocorrelation versus distance. The autocorrelation decreases progressively until an asymptote is reached. The distance from which this asymptote reaches about 30 voxels is the range of the microstructure. A rule of thumb for an optimal subset size is to take about 3-4 times this value. We will use 110 voxels in this study (see Figure 19.4).
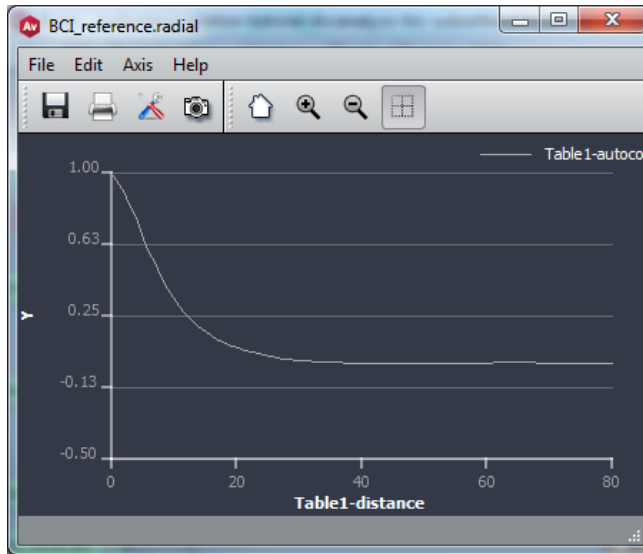
Figure 19.4: Autocorrelation versus Distance Plot

19.1.2 Compute a Good Initial Guess using the Subset-Based Approach

In this section, you will capture the large displacements using the subset-based (local) approach from the *Digital Volume Correlation* module and use these results to initialize the most robust Finite-Element-based (global) approach.

- Load the file data/tutorials/xvolumecorrelation/BCI_peak_force.am corresponding to the previous data after compression resulting in internal displacement.
- Display it using an *Ortho Slice* module and set the *Orientation* port to *xz*.
- Link the Ortho Slices attached to *BCI_reference* and *BCI_peak_force* using the Connection Editor (drag the *Slice number* port and the *Orientation* port of one Ortho Slice to the other on the Project View) (see Figure 19.5).
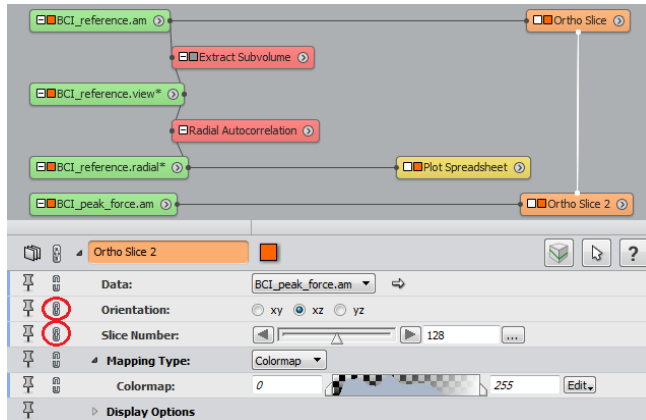
Figure 19.5: Connections between Ortho Slices

- Turn the viewer toggle of the *Ortho Slice 2* ON/OFF to visualize the regions that are highly deformed. You should observe that most of the deformation is localized in the trabecular bone region (top).
- Estimate the maximum displacement in this region. To do so, ensure that the Quick Probe from the Viewer Toolbar is set to *Continuous Update* (Figure 19.6) then place the mouse on the 3D viewer and read the coordinates displayed in the progress bar. Be sure to be on Interact mode (if not, press **ESC**).
- Alternatively, you can use the Measure tool from the Viewer Toolbar. Maximum displacement occurs for the latter slices (around slice 230, xz orientation). Choose a slice, then by switching between the two ortho slices, place the measurement tool and try to measure approximately the maximum displacement.
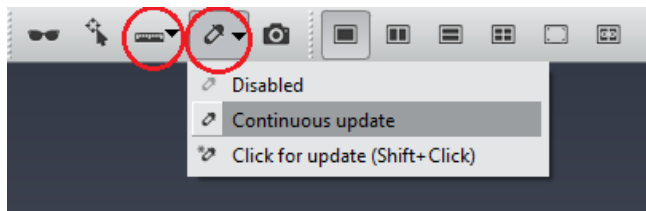
Figure 19.6: Quick Probe or Measure tool can be used to Estimate the Maximum Displacement

- The maximum displacement should be around 10 (see Figure 19.7).
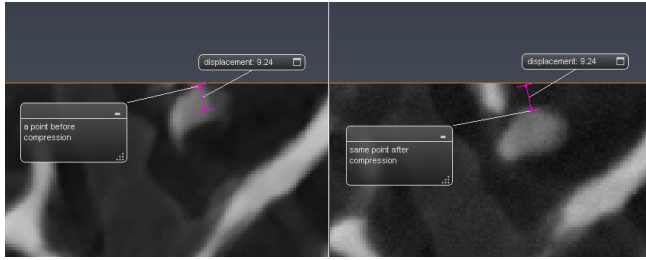
Figure 19.7: Displacement Measurement between the two ortho slices

- Attach a *Digital Volume Correlation* module to the data *BCI_reference*. Set the *Deformed volume* port to *BCI_peak_force.am* and *DVC approach* port to *Subset-based (local)*. Activate the Advanced toggle switch. Set the *Sub-volume size* port to 110, *Max displacement* to 10, and *Correlation Filter* to 0.7. Turn *Enable Display* port to *ON* to visualize the sub-volumes. Set the *Metric* port to *Correlation* and *Transform* port to *Translation + Rotation*. For each sub-volume, the algorithm finds iteratively the six parameters (three translations, three rotations) of a rigid transformation matrix that satisfy the best match between the grey level intensity of the reference and deformed image (Metric = 0: no match; Metric = 1: perfect match) (see Figure 19.8). Click **Apply**.
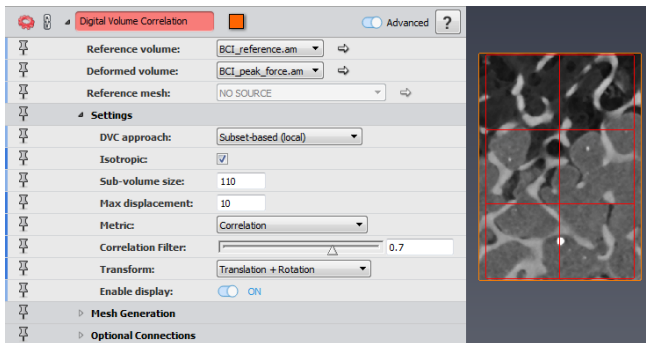
Figure 19.8: Digital Volume Correlation module with Subset-based (local) approach.

This action creates four outputs (one is hidden):

1. *dvc_mesh_cellSize_110x110x110_*: grid representing the subsets
2. *BCI_reference.disp*: displacement field
3. *BCI_reference.strain*: strain tensor
4. *BCI_reference.metric*: metric map. Enter *"DigitalVolumeCorrelation" showLaDvcMetricOutput* in the TCL console to see it.

- Plot the statistics of the metric map by attaching a *Histogram* module to *BCI_reference.metric*. On average, the metric is about 0.98 indicative of a good correlation.

- Visualize the metric map by attaching a *Grid View* module to *BCI_reference.metric*. In Grid View, set *Colouring* port to *Data Mapping*, *Opacity* port to 0.4 and *Value Mapping* to *Cell Values*. In *BCI_reference.metric* set the *Colormap* port to *physics* and adjust the range by clicking on *Adjust the range to* and select *BCI_reference.metric*. Press *create legend* in the *Options* port to visualize the legend. The metric is the lowest (0.96) in the region where the displacement is high (trabecular bone region at the top).

- Visualize the displacement vectors by attaching a *Vector Field* module to *BCI_reference.disp*. Set the *Scale* port to 25 (open more options on Scale port line, and change Max value setting to 25) , *Coloring* port to *Uniform* and select the white color from the port *Uniform color* (see Figure 19.9).

- Attach a *Magnitude* module to *BCI_reference.disp* if you want to visualize the displacement magnitude.
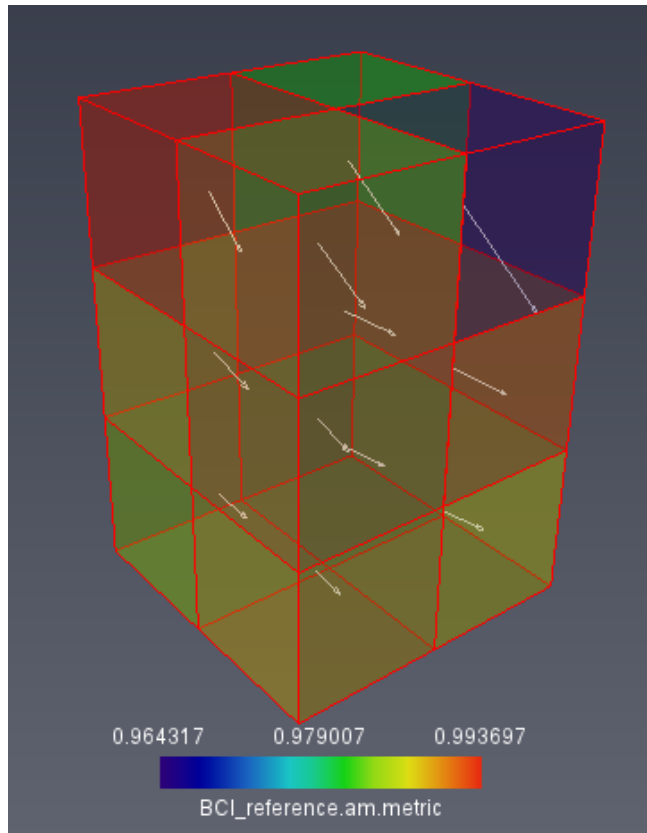
Figure 19.9: Metric Map and Displacement Vectors

19.1.3 Run a Robust FE-Based DVC Technique

In this section, the displacement field obtained using the subset-based (local) approach is used to initialize a Finite-Element-based (global) DVC algorithm using a finer mesh.

- Hide all previous visualization modules except the ortho slices.
- Select again the *Digital Volume Correlation* module.
- Turn off the *Enable Display* port of the subset-based (local) approach.
- Generate a tetrahedral grid using the *Mesh Generation* section. Set the *Cell size* to 48 and turn on the *Enable Display* port to see the nodes of the mesh (see Figure 19.10). Click *Generate Mesh* to create the mesh.
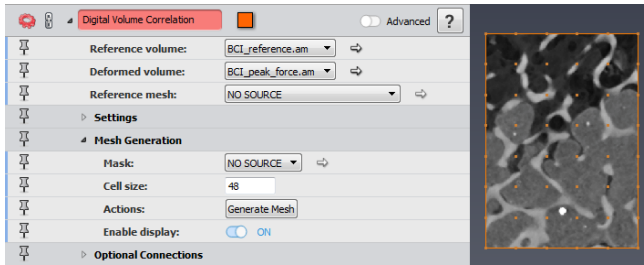
Figure 19.10: Generate a Tetrahedral Grid (yellow points represent the nodes of the mesh to create).

- Visualize the mesh by attaching a *Grid View* module to the created mesh dvc_mesh_cellSize_48x48x48_. Set the *Draw Style* port to *Solid Outline* to visualize the elements (see Figure 19.11).
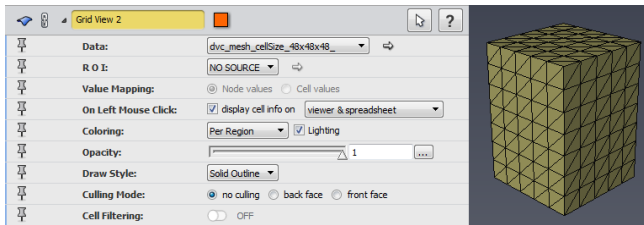
Figure 19.11: Grid View Module Properties and Tetrahedral Grid Visualization

- Change the port *DVC Approach* to *Finite-Element-based (global)*. Set the *Max Iterations* port to 30 and *Convergence criterion* to 0.001. Attach the mesh *dvc_mesh_cellSize_48x48x48_* to the port *Reference mesh*. In the *Optional Connections* section, set the *Initial displacement* to *BCI_reference.disp* (displacements computed using the local approach with a sub-volume size of 110 voxels) (see Figure 19.12). Click **Apply**.
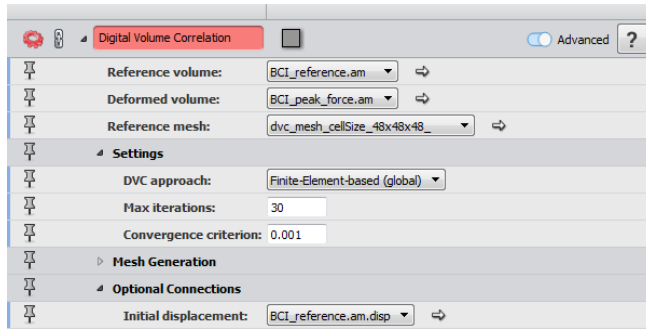
Figure 19.12: Finite-Element-Based (global) Approach.

This action creates three outputs (one is hidden):

1. *BCI_reference2.disp*: displacement field
2. *BCI_reference2.strain*: strain tensor
3. *BCI_reference.res*: residual. Enter *"DigitalVolumeCorrelation" showGaDvcResidualOutput* in the TCL console to see it.

In the global approach, the displacement field (also known as optical flow) is obtained by minimizing the correlation residuals, therefore it is recommended that you first check the correlation residuals.

- Attach an *Histogram* module to the residuals image *BCI_reference.res* and another to *BCI_reference.am*.
- Uncheck logarithmic plot option and change range to [0;255] for both histograms. Click **Apply** for both (see result on Figure 19.13).
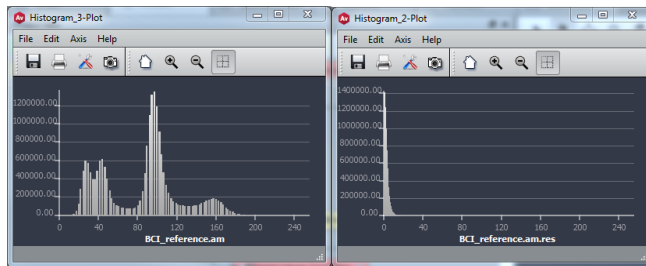
Figure 19.13: Histogram of the Reference Volume (left) and Residual (right)

As shown in the *In Range* port of the *Histogram* module for the residuals image, mean is about 2.6 and standard deviation is 2.8, indicative of a good correlation.

- Attach an *Ortho Slice* module to the residuals image *BCI_reference.res* and change the Colormap

by clicking on **Edit**, **Adjust Range To** then **Data Histogram**.

High intensity voxels, i.e. the highest residuals, are found at the top in the bone region interface where the trabecular-like morphology is more complex and the deformation is high (see Figure 19.14).
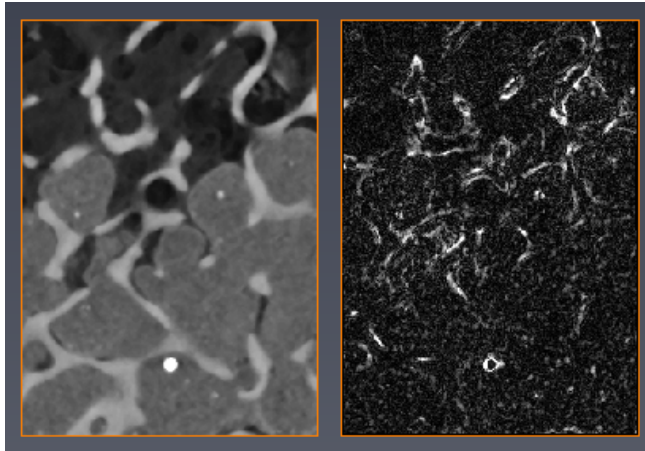
Figure 19.14: Comparing Transverse Section

19.1.4 Visualize and Animate the Results of the Global Approach

In this section, we will visualize the displacement and strain fields using different methods.

19.1.4.1 Visualize the Vertical Displacement and Axial Strain over the DVC Mesh

- Attach an *Arithmetic* module to *BCI_reference2.disp*. Set *Output Data Type* port to *scalar* and *Expression* port to *Az*. Rename the output *Result* dataset to *Uz* (select *Result* and press F2).

- Attach an *Arithmetic* module to *BCI_reference2.strain*. Set *Output Data Type* port to *scalar* and *Expression* port to *Akk*. Rename the output *Result* dataset to *e33* (select *Result* and press F2).

- Connect a *Grid View* module to Uz and e33 and visualize the vertical displacement and axial strain.

- To visualize the displacement vectors, connect a *Vector Field* module to *BCI_reference2.disp*. Set the *Scale* port to 25 (open more options on Scale port line and change Max value setting to 25), *Coloring* port to Uniform and select the yellow color from the port Uniform color. (see Figure 19.15).
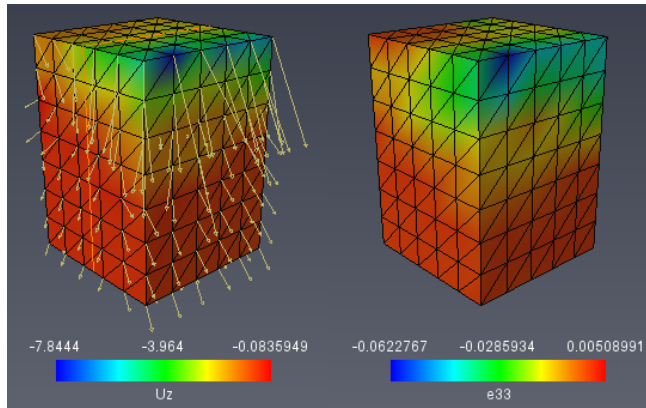
Figure 19.15: 3D visualization of the vertical displacement (left) and axial strain (right)

19.1.4.2 Visualize the Axial Strain using a Surface Mesh

The first method is to remap the displacements and strains into a regular grid (image) and visualize the axial strain over the surfaces of phases.

- Load segmented data data/tutorials/xvolumecorrelation/BCI_reference3.labels.am
- Attach an *Arithmetic* module to *BCI_reference2.disp* with the following parameters:
  - Set *Output Data Type* to *same as input*.
  - Set *Expression X* port to *Ax*.
  - Set *Expression Y* port to *Ay*.
  - Set *Expression Z* port to *Az*.
  - Set *Output grid Type* to *regular*.
  - Set *Resolution* port to the reference volume dimensions (241, 241, 337).
- Press **Apply** and rename the output to *DispVectors.RegularGrid*.
- Attach an *Arithmetic* module to *e33* dataset computed previously with the following parameters:
  - Set *Output Data Type* to *scalar*.
  - Set *Expression* port to *A*.
  - Set *Output grid Type* port to *regular*.
  - Set *Resolution* port to the reference volume dimensions (241, 241, 337).
- Press **Apply** and rename the output *e33.RegularGrid*.
- Attach a *Generate Surface* module to *BCI_reference3.labels*, keep all parameters and press **Apply**. With this action a surface model of the bone-cement phases has been generated.
- Attach a *Surface View* module to *BCI_reference3.labels.surf* with the following parameters:

- Set *Color Field* port to *e33.RegularGrid*
- Adjust the *Colormap* port to *physics*.
- Using the Buffer port, click on **Clear** action button then select a Material of interest (bone or cement) and click on **Add** action button to display it. (see Figure 19.16).
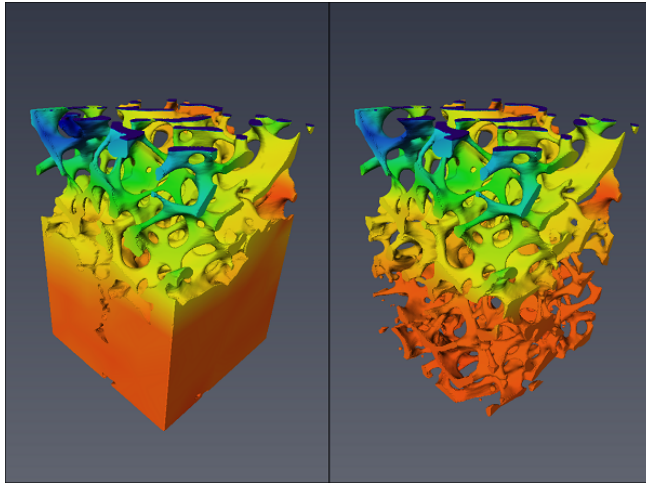
Figure 19.16: 3D Visualization of the Axial Strain in the Bone-Cement (left) and Bone phase (right)

- To better reveal the internal strains, connect a *Volume Edit* module to *BCI_reference3.labels* with following parameters:
  - Set *Tool* port to *TabBox*
  - Select a quarter of the volume.
  - Click on the button *Inside* from the *Cut* port. This action will remove the quarter of volume selected.
- Repeat the previous steps on the new volume *BCI_reference3.labels.modif* (generate a surface by connecting a *Generate Surface* module and visualize it with a *Surface View* module).
- Connect a *Vectors Slice* module to *DispVectors.RegularGrid* with following parameters:
  - Set the *Resolution* to *20* (x and y).
  - Set the *Colormap* to *Constant Color* then select white.
  - Set *Orientation* port to *yz* on the Clipping Plane.
- See result on Figure 19.17.
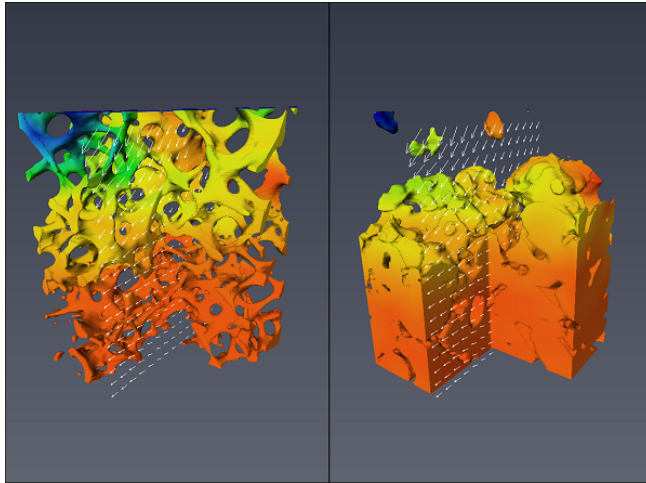
Figure 19.17: Displacement Vectors Superimposed on the 3D Axial Strain in the Trabecular Bone (left) and Cement Phase (right)

It is observed that most of the deformation is taken by the trabecular bone with a poor strain transfer in the cement region. Other bone-cement composite formulations containing more cortical bone have been tested and proved to be more efficient in diffusing the strains in the cement [1].
To see the implant deformation:

- Attach an *Apply Deformation* module to the surface created with following parameters:

    - Set *Vector Field* port to *DispVectors.RegularGrid*.

- Press **Apply**.
- Attach a *Surface View* module.
- Switch between the view modules for the reference and the deformed surfaces to see the deformation.

19.1.4.3 Visualize the Axial Strain using a Tetrahedral Mesh

The second method is to visualize the displacements and strains in a tetrahedral grid.

- Load tetrahedral grid data data/tutorials/xvolumecorrelation/BCI_reference3.labels.tetra.am (Figure 19.18).
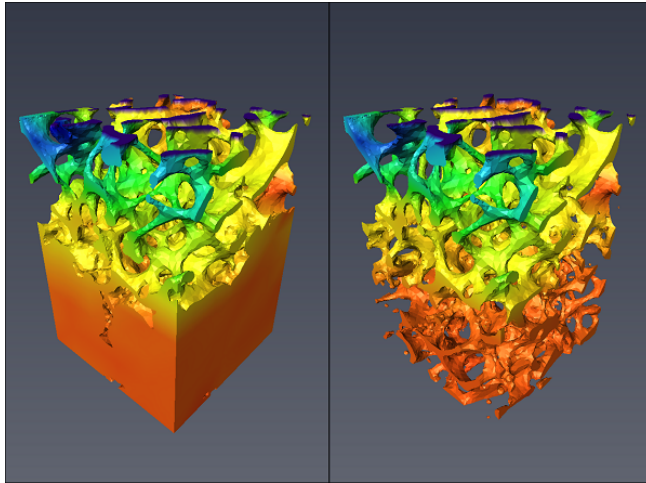
Figure 19.18: Tetragrid Visualisation (left - cement/bone ; right - bone)

- Attach an *Apply Deformation* module to the tetrahedral grid with following parameters:
  - Set *Vector Field* port to *DispVectors.RegularGrid*.
- Press **Apply**.
- Attach a *Tetra Grid View* module to reference tetra grid data. Set:
  - Set *Color Field* port to *e33.RegularGrid*.
  - Set *Colormap* to *physics* and adjust range to *e33.RegularGrid* then *Data Histogram*.
- Switch *Data* port between reference and deformed data to see the deformation.

19.1.5 Dialog between experience and simulation

The output displacement field can be used to enrich a numerical simulation by using measured boundary conditions, or to optimize this simulation by comparing numerical and measured data. The generated mesh can be exported in FE software packages (Abaqus, Ansys, etc.) to perform finite element computations at different scales: (i) continuum scale (DVC module grid output), (ii) scale of the constituents (tetrahedral grid surface). Realistic boundary conditions can be prescribed by applying the DVC displacements at the boundaries of the mesh.

19.1.6 References

[1] TOZZI et al., Microdamage assessment of bone-cement interfaces under monotonic and cyclic compression. Journal of Biomechanics 47:3466-3474 (2014)

Index