pgf-blur package
version 1.01

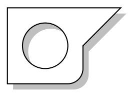
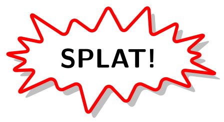
Martin Giese[*]

2012/04/24

Acknowledgement
This package evolved from a discussion on `http://tex.stackexchange.com/`. In particular, the author received invaluable help from Andrew Stacey.

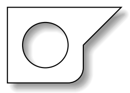
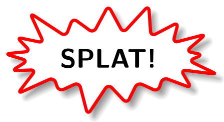
1 Introduction

The ability to paint shadows on arbitrary shapes is a standard feature of TikZ/PGF. However, these shadows are usually ‘sharp’:

Such shadows are often optically too intrusive. A more pleasing effect is achieved if the edges of the shadow are ‘blurred,’ i.e., getting gradually lighter and more transparent toward the outside. This effect can be achieved in TikZ/PGF with the `circular drop shadow` key, but that works only with ellipses and circles.

The `pgf-blur` package provides blurred shadows that can be added to any closed path, including node borders:

The new TikZ options provided by the package are described in section 2 of this document. Section 3, if present, documents the implementation consisting of a generic TeX file.

[*]`martingi@ifi.uio.no`

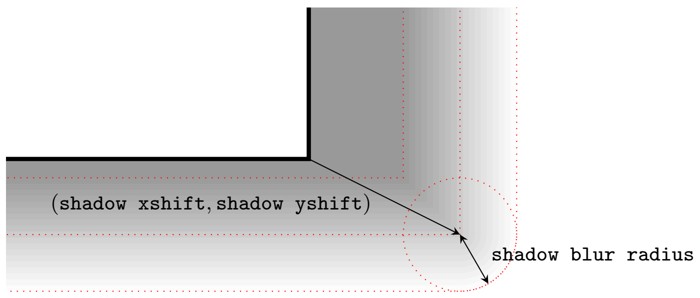


Figure 1: Shadow geometry

2 Package Usage

To use the package, the document needs to say `\usepackage{tikz}` and

```
\usetikzlibrary{shadows.blur}
```

in the preamble.

`blur shadow` A blurred shadow is added to a path or node by the option `blur shadow`, e.g.

```
\begin{tikzpicture}
  \filldraw[fill=white, draw=black, blur shadow] (0,0) circle (1);
  \node[fill=white, draw=black, blur shadow] at (4,0.5) {\Large Node};
\end{tikzpicture}
```

which gives

Note that this usually makes sense only with closed paths that are filled (otherwise the shadow is visible through the path, and one wonders what is throwing the shadow) and often looks best if used with a drawn outline.

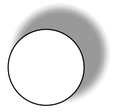
The appearance of the shadow can be fine-tuned by giving arguments to the `blur shadow` options. These options do not have any effect if given outside of the argument of either `blur shadow` or `every shadow` (described later)

`shadow xshift` `shadow yshift` `shadow scale` The shadow is based on a shifted, and possibly scaled copy of the original path. These options are described in the Ti*k*Z/PGF user manual, and they work in the same way for blurred shadows. See also Fig. 1. The default values are 3ex for `shadow xshift`, −3ex for `shadow yshift`, and 1 for `shadow scale`.

Here is an example for the usage of these options:

```
\begin{tikzpicture}
   \filldraw[fill=white, draw=black,
             blur shadow={shadow xshift=1ex,
                                shadow yshift=1ex,
                                shadow scale=1.2}]
      (0,0) circle (0.5);
\end{tikzpicture}
```

Which gives:

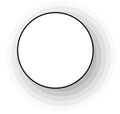


shadow blur radius Fig. 1 shows how the blur shadow spreads out the boundary of the path over a circular region. The intent is to mimic the effect of a circular light source over the shape. the radius of the "blurring" can be set with the `shadow blur radius` option, which has a default value of .4ex.

Here is an example of a drastically enlarged blur radius:

```
\begin{tikzpicture}
   \filldraw[fill=white, draw=black,
             blur shadow={shadow blur radius=1.5ex}]
      (0,0) circle (0.5);
\end{tikzpicture}
```

Which gives:

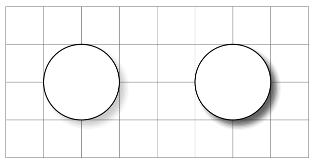


shadow opacity Shadows are transparent. They are always black since a shadow is the absence of light. The opacity of the interior of the shadow, i.e. the darkest region can be controlled with the `shadow opacity` option. It is given as a percentage, i.e. a number between 0 and 100. The default is 40.

Here are examples of lighter and darker shadows:

```
\begin{tikzpicture}
   \draw[help lines,step=0.5] (-1,-1) grid (3,1);
   \filldraw[fill=white, draw=black,
             blur shadow={shadow opacity=20}]
      (0,0) circle (0.5);
   \filldraw[fill=white, draw=black,
             blur shadow={shadow opacity=60}]
      (2,0) circle (0.5);
\end{tikzpicture}
```

Which gives:



`shadow blur extra rounding` A close inspection of the shadow in Fig. 1 reveals that if the original path has a corner, then the “lines of equal opacity” in the inner part of the shadow will also have corners. This is a consequence of the way the shadows are rendered by default. It gives the impression of the darkest part of the shadow extruding a little too much into the faded part. This effect can be reduced by adding the option `shadow extra rounding`, which has the effect of rounding all corners of the path before rendering the shadow. The rounding inset can be given as an argument; it defaults to the current value of the blur radius.

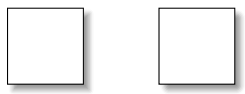
Several points should be remembered when using this option:

- It uses the mechanism that Ti*k*Z also uses for the `rounded corners` option. This works badly, distorting the path in strange ways, if the individual segments of the path are too short to be rounded with the given inset. If the shadows look strange, try reducing the inset, or drop this option altogether.
- Rounding many complex paths can slow TEX down considerably.
- The ideal rounding inset depends on the angle the path has at each corner. This is not taken into account, the same inset is used everywhere. The default value is the one that works best with 90° angles, but it also looks fairly good with other angles.
- With or without this option, the shadows will not be photorealistic.

Here is an example of squares without and with extra rounding applied:

```
\begin{tikzpicture}
   \filldraw[fill=white, draw=black,
             blur shadow]
      (0,0) rectangle (1,1);
   \filldraw[fill=white, draw=black,
             blur shadow={shadow blur extra rounding}]
      (2,0) rectangle (3,1);
\end{tikzpicture}
```

Which gives:



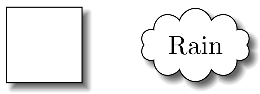
`shadow blur steps` The transition of opacity in these shadows is actually not smooth, but proceeds in a finite number of discrete steps. Specifically, there is a number n, such that the (shifted and scaled) original path fades outward to complete transparency in n steps and within the selected `shadow blur radius`, and inward to the maximum opacity of the shadow, also within n steps and the `shadow blur radius`. This number of steps n can be selected using the `shadow blur steps` option. It defaults to 4, which is enough e.g. for inconspicuous shadows in presentations that nobody examines with a magnifying glass. The examples in the introduction use 8, under the assumption that readers will have a close look. Fig. 1 uses 10 because the blur radius is so large.

A large number of steps will slow down both the TEX processing and the PDF rendering, usually with very little visible impact.

`every shadow` To apply the same set of options to every shadow, it is possible to define the *style* `every shadow`, which is taken from the standard shadow library. For instance, in the following, darker shadows with more steps are selected for several shapes:

```
\begin{tikzpicture}
  [every shadow/.style={shadow opacity=60,
                        shadow blur steps=7}]
   \filldraw[fill=white, draw=black, blur shadow]
      (0,0) rectangle (1,1);
   \node[cloud,shape aspect=2,fill=white, draw=black,
            blur shadow]
      at (2.5,0.5) {Rain};
\end{tikzpicture}
```

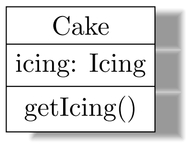
Which gives:



2.1 A Note on Nodes

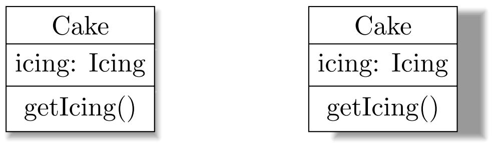
While this library plays nicely with most closed paths, it doesn't like paths with non-closed parts. This is particularly annoying when used with standard node shapes, where you can't do much about the paths. Take the `rectangle split` shape for instance:

```
\begin{tikzpicture}
    \node[rectangle split, rectangle split parts=3, draw, fill=white,
      blur shadow={shadow xshift=1em}] {...};
\end{tikzpicture}
```



Note how the part separation lines produce white bars in the shadow. This is really hard to avoid in general. There are two simple workarounds: a) choosing a very small `shadow xshift` so most the "bar" gets hidden behind the shape, or b) putting the node inside a simple rectangle node and add a shadow to the latter. For solution b) to work, the inner padding of the outer node has to be removed. This in turn requires it to be reinstated for the inner node. Here's the example showing both workarounds:

```
\begin{tikzpicture}
\node[rectangle split, rectangle split parts=3,
      draw, fill=white,
      blur shadow={shadow xshift=0.25ex}] at (-2,0) {...};
\node[blur shadow={shadow xshift=1em},inner sep=0pt] at (2,0) {
      \tikz\node[rectangle split, rectangle split parts=3,
                 draw, fill=white, inner sep=0.333em] {...};};
\end{tikzpicture}
```



Solution b) is fine for rectangular outlines, but it wont work for e.g. the `cylinder` shape.

3 Implementation

`\fileversion` `\filedate` We define the file version and date, and import the original shadow code for the offset and scale parameters.

```
⟨*texfile⟩
\def\fileversion{1.01}
\def\filedate{2012/04/24}
\message{ v\fileversion, \filedate}
\usetikzlibrary{shadows}
\usetikzlibrary{calc}
```

`shadow blur radius` `shadow blur extra rounding` `shadow blur steps` `shadow opacity` Next we define the various Ti*k*Z options, with their default values. The options `shadow xshift`, `shadow yshift`, `shadow scale` are imported from the standard Ti*k*Z shadow library, so we don't need to do anything for them. Note how the default value of `shadow blur extra rounding` is by default set to the blur radius. This works because the value of this key is evaluated before it is used. All other options just store values in a couple of macros.

```
\tikzset{
  /tikz/shadow blur radius/.store in=\pgfbs@radius,
  /tikz/shadow blur radius=.4ex,
  /tikz/shadow blur extra rounding/.store in=\pgfbs@extra@rounding,
  /tikz/shadow blur extra rounding=\pgfutil@empty,
  /tikz/shadow blur extra rounding/.default=\pgfbs@radius,
  /tikz/shadow blur steps/.store in=\pgfbs@steps,
  /tikz/shadow blur steps=4,
  /tikz/shadow opacity/.store in=\pgfbs@opacity,
  /tikz/shadow opacity=40,
```

`blur shadow` The user level option `blur shadow` sets the `shadow xshift`, `shadow yshift`, `shadow scale` options to more useful defaults than the ones inherited from the shadows library. It includes any options set in the `every shadow` style, and the argument to `blur shadow`. Rendering the shadow is declared as a `.preaction` on the path. Ti*k*Z will take care of saving the path for us.

```
  /tikz/blur shadow/.style={
    shadow scale=1,
    shadow xshift=.5ex,
    shadow yshift=-.5ex,
    preaction=render blur shadow,
    every shadow,
    #1,
  },
```

`render blur shadow` The following does the actual shadow rendering. After some preliminary computation of dimensions, shifting and scaling is done using a canvas transform. The actual blurring effect is done using a special fading. Since PGF insists on centering every fading when it's constructed, it has to be shifted back again when it's installed. The shadow is painted by filling a large black rectangle using the constructed fading.

```
  /tikz/render blur shadow/.code={
    \pgfbs@savebb
    \pgfsyssoftpath@getcurrentpath{\pgfbs@input@path}%
    \pgfbs@compute@shadow@bbox
    \pgfbs@process@rounding{\pgfbs@input@path}{\pgfbs@fadepath}%
    \pgfbs@apply@canvas@transform
    \colorlet{pstb@shadow@color}{white!\pgfbs@opacity!black}%
    \pgfdeclarefading{shadowfading}{\pgfbs@paint@fading}%
    \pgfsetfillcolor{black}%
    \pgfsetfading{shadowfading}%
       {\pgftransformshift{\pgfpoint{\pgfbs@midx}{\pgfbs@midy}}}%
    \pgfbs@usebbox{fill}%
    \pgfbs@restorebb
  },
}
```

`\pgfbs@savebb` Shadow rendering works with a fading. For a fading to contain PGF code, it must contain a `pgfpicture`. And nested `pgfpicture`s mess up the bounding box of the surrounding picture. Which is why we save the current picture bounding box at the beginning of the shadow code and restore it at the end. Thanks go to Andrew Stacey for this!

```
\def\pgfbs@savebb{%
  \edef\pgfbs@restorebb{%
    \global\pgf@picminx=\the\pgf@picminx\relax
    \global\pgf@picmaxx=\the\pgf@picmaxx\relax
    \global\pgf@picminy=\the\pgf@picminy\relax
    \global\pgf@picmaxy=\the\pgf@picmaxy\relax
  }%
}
```

\pgfbs@restorebb Executing \pgfbs@savebb sets this bounding box restoring macro to something useful.

```
\def\restorebb{}%
```

\pgfbs@process@rounding This macro prepares the path by taking care of all things having to do with rounding. First, it applies extra rounding to the path if requested by the `shadow blur extra rounding` option. Second, it removes all rounding tokens in the path by calling the \pgfprocessround macro. This is because rounding tokens don't work well with the fading code for some reason or other. #1 must be a PGF soft path. #2 must be a macro into which the resulting soft path will be stored.

```
\def\pgfbs@process@rounding#1#2{
  \expandafter\ifx\pgfbs@extra@rounding\pgfutil@empty%
    \pgfprocessround{#1}{#2}%
  \else%
    \pgfmathsetmacro\pgfbs@exrd@val{\pgfbs@extra@rounding}%
    \pgfbs@roundpath{#1}{\pgfbs@exrd@val pt}%
    \pgfsyssoftpath@getcurrentpath{\pgfbs@extraroundedpath}%
    \pgfprocessround{\pgfbs@extraroundedpath}{#2}%
  \fi%
}
```

\pgfbs@roundpath \pgfbs@roundpath{#1}{#2} rounds every potential corner in path #1 with an inset of at least #2. Corners that are already rounded in #1 are either left intact if their insets are $\geq$#2, or the insets are increased to #2. Any rectangle tokens are resolved into moveto/lineto/closepath with rounding. The result is appended to PGF's "current path."

The code works by giving an appropriate definition for each of the PGF soft path tokens and then executing the path.

```
\def\pgfbs@roundpath#1#2{%
  {%
    \def\pgfbs@rp@skipround{%
      \let\pgfbs@rp@possibleround\pgfbs@rp@insertround}%
    \def\pgfbs@rp@insertround{\pgfsyssoftpath@specialround{#2}{#2}}%
    \let\pgfbs@rp@possibleround\pgfbs@rp@insertround%
    %
    \def\pgfsyssoftpath@movetotoken##1##2{%
      \pgfsyssoftpath@moveto{##1}{##2}}%
    \def\pgfsyssoftpath@linetotoken##1##2{%
      \pgfbs@rp@possibleround\pgfsyssoftpath@lineto{##1}{##2}}%
    \def\pgfsyssoftpath@rectcornertoken##1##2##3##4##5{%
      \pgf@xa=##1\relax%
      \advance\pgf@xa by##4%
      \pgf@ya=##2\relax%
      \advance\pgf@ya by##5%
      \pgfsyssoftpath@moveto{##1}{##2}%
      \pgfbs@rp@possibleround%
      \pgfsyssoftpath@lineto{\the\pgf@xa}{##2}%
      \pgfbs@rp@possibleround%
      \pgfsyssoftpath@lineto{\the\pgf@xa}{\the\pgf@ya}%
      \pgfbs@rp@possibleround%
      \pgfsyssoftpath@lineto{##1}{\the\pgf@ya}%
      \pgfbs@rp@possibleround%
      \pgfsyssoftpath@closepath}%
    \def\pgfsyssoftpath@curvetosupportatoken%
       ##1##2##3##4##5##6##7##8{%
      \pgfbs@rp@possibleround%
      \pgfsyssoftpath@curveto{##1}{##2}{##4}{##5}{##7}{##8}}%
    \def\pgfsyssoftpath@closepathtoken##1##2{%
      \pgfbs@rp@possibleround\pgfsyssoftpath@closepath}%
    \def\pgfsyssoftpath@specialroundtoken##1##2{%
      \pgfmathsetmacro\pgfbs@rp@ra{max(##1,#2)}%
      \pgfmathsetmacro\pgfbs@rp@rb{max(##2,#2)}%
      \pgfsyssoftpath@specialround%
          {\pgfbs@rp@ra pt}{\pgfbs@rp@rb pt}%
      \let\pgfbs@rp@possibleround\pgfbs@rp@skipround%
    }
    #1%
  }
}
```

`\pgfbs@compute@shadow@bbox` `\pgfbs@minx` `\pgfbs@midx` `\pgfbs@maxx` `\pgfbs@miny` `\pgfbs@midy` `\pgfbs@maxy` `\pgfbs@shadow@bbox`

This macro figures out the bounding box of the shadow: it's the same as the bounding box of the current path, but enlarged by the blur radius in each direction. It also computes the coordinates of the center of the bounding box. These are stored in macros `\pgfbs@{min|mid|max}{x|y}`. It also creates a soft path for the bounding box which is stored in `\pgfbs@shadow@bbox`.

```
\def\pgfbs@compute@shadow@bbox{%
  \edef\pgfbs@minx{\the\pgf@pathminx}%
  \edef\pgfbs@miny{\the\pgf@pathminy}%
  \edef\pgfbs@maxx{\the\pgf@pathmaxx}%
  \edef\pgfbs@maxy{\the\pgf@pathmaxy}%
  \pgfmathsetmacro\pgfbs@midx{0.5*(\pgfbs@minx + \pgfbs@maxx)}%
  \pgfmathsetmacro\pgfbs@midy{0.5*(\pgfbs@miny + \pgfbs@maxy)}%
  \pgfmathsetmacro\pgfbs@minx{\pgfbs@minx - \pgfbs@radius}%
  \pgfmathsetmacro\pgfbs@miny{\pgfbs@miny - \pgfbs@radius}%
  \pgfmathsetmacro\pgfbs@maxx{\pgfbs@maxx + \pgfbs@radius}%
  \pgfmathsetmacro\pgfbs@maxy{\pgfbs@maxy + \pgfbs@radius}%
  \pgfmathsetmacro\pgfbs@wd{\pgfbs@maxx - \pgfbs@minx}%
  \pgfmathsetmacro\pgfbs@ht{\pgfbs@maxy - \pgfbs@miny}%
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty%
  \pgfsyssoftpath@rect{\pgfbs@minx pt}{\pgfbs@miny pt}%
                       {\pgfbs@wd pt}{\pgfbs@ht pt}%
  \pgfsyssoftpath@getcurrentpath{\pgfbs@shadow@bbox}%
  \pgfsyssoftpath@setcurrentpath\pgfutil@empty%
}
```

`\pgfbs@set@fading@pic@bbox`

Set the bounding box of the fading picture painted by `\pgfbs@paint@fading`. Normally, the bounding box is updated automatically, but this doesn't happen when PGF's soft paths are used. The code that applies a fading needs the dimensions of the fading to be correct. So we explicitly set the picture bounding box according to the previously computed values. Thanks again to Andrew Stacey for pointing out this subtlety!

```
\def\pgfbs@set@fading@pic@bbox{
  \global\pgf@picminx=\pgfbs@minx pt\relax
  \global\pgf@picminy=\pgfbs@miny pt\relax
  \global\pgf@picmaxx=\pgfbs@maxx pt\relax
  \global\pgf@picmaxy=\pgfbs@maxy pt\relax
}
```

`\pgfbs@usefadepath` The code of the `render blur shadow` option stores the pre-processed soft path into the macro `\pgfbs@fadepath`. This macro 'uses' this path by executing `\pgfusepath{#1}` on it.

```
\def\pgfbs@usefadepath#1{%
  \pgfsyssoftpath@setcurrentpath{\pgfbs@fadepath}%
  \pgfsyssoftpath@flushcurrentpath%
  \pgfusepath{#1}%
}
```

`\pgfbs@usebbox` This is similar to the previous macro, but it 'uses' the bounding box path.

```
\def\pgfbs@usebbox#1{%
  \pgfsyssoftpath@setcurrentpath{\pgfbs@shadow@bbox}%
  \pgfsyssoftpath@flushcurrentpath%
  \pgfusepath{#1}%
}
```

`\pgfbs@apply@canvas@transform` This achieves the scaling and shifting of the shadow. It is done by a canvas transform to avoid iterating through a soft path and transforming many coordinates inside TeX. Scaling is 'around' the bounding box mid point computed by `\pgfbs@compute@shadow@bbox`.

```
\def\pgfbs@apply@canvas@transform{
  \pgflowlevel{
    \pgftransformshift{\pgfpoint{\pgfbs@midx}{\pgfbs@midy}}
    \pgftransformscale{\pgfkeysvalueof{/tikz/shadow scale}}
    \pgftransformshift{\pgfpoint%
      {\pgfkeysvalueof{/tikz/shadow xshift}-\pgfbs@midx}
      {\pgfkeysvalueof{/tikz/shadow yshift}-\pgfbs@midy}
    }
  }
}
```

`\pgfbs@paint@fading` This paints the actual fading picture. It works by repeatedly drawing the `\pgfbs@fadepath` with different line widths and different shades of gray, leading to different opacity. First, the outer part of the fading is drawn. Then, remaining operations are clipped to the inside of the path. Next, the path is filled with the shadow opacity. Finally, the inner part of the fading is drawn. As mentioned previously, the bounding box needs to be set explicitly.

```
\def\pgfbs@paint@fading{
  \pgfpicture
    % fix bounding box.
    \pgfbs@set@fading@pic@bbox
    % compute increments for line width and opacity
    \pgfmathsetmacro\pgfbs@op@step{50/\pgfbs@steps}
    \pgfmathsetmacro\pgfbs@wth@step{4*\pgfbs@radius/(2*\pgfbs@steps-1)}
    % draw the outer part of the fading,
    % starting with lightest, outermost line
    \pgfsetroundjoin
    \pgfmathsetmacro\pgfbs@max@i{\pgfbs@steps-2}
    \pgfmathsetmacro\pgfbs@wth{2*\pgfbs@radius}
    \pgfmathsetmacro\pgfbs@op{100-0.5*\pgfbs@op@step}
    \foreach \pgfbs@i in {0,...,\pgfbs@max@i} {
      \pgfsetlinewidth{\pgfbs@wth pt}
      \pgfsetstrokecolor{black!\pgfbs@op!pstb@shadow@color}
      \pgfbs@usefadepath{stroke}
      \pgfmathsetmacro\pgfbs@wth{\pgfbs@wth-\pgfbs@wth@step}
      \global\let\pgfbs@wth=\pgfbs@wth
      \pgfmathsetmacro\pgfbs@op{\pgfbs@op-\pgfbs@op@step}
      \global\let\pgfbs@op=\pgfbs@op
    }
    % clip to inside of path
    \scope
    \pgfbs@usefadepath{clip}
    % fill inside with final darkest shadow color
    \pgfsetfillcolor{pstb@shadow@color}
    \pgfbs@usebbox{fill}
    % draw the inner part of the fading,
    % starting with the darkest, innermost line
    \pgfmathsetmacro\pgfbs@wth{2*\pgfbs@radius}
    \pgfmathsetmacro\pgfbs@op{0.5*\pgfbs@op@step}
    \foreach \pgfbs@i in {0,...,\pgfbs@max@i} {
      \pgfsetlinewidth{\pgfbs@wth pt}
      \pgfsetstrokecolor{black!\pgfbs@op!pstb@shadow@color}
      \pgfbs@usefadepath{stroke}
      \pgfmathsetmacro\pgfbs@wth{\pgfbs@wth-\pgfbs@wth@step}
      \global\let\pgfbs@wth=\pgfbs@wth
      \pgfmathsetmacro\pgfbs@op{\pgfbs@op+\pgfbs@op@step}
      \global\let\pgfbs@op=\pgfbs@op
    }
    \endscope
    % a final stroke to hide clip/antialiasing artifcats
    \pgfsetstrokecolor{black!50!pstb@shadow@color}
    \pgfsetlinewidth{0.5*\pgfbs@wth@step}
    \pgfbs@usefadepath{stroke}
  \endpgfpicture
}
⟨/texfile⟩
```