User's Guide to the PGF Package, Version 0.65

http://latex-beamer.sourceforge.net

Till Tantau
tantau@users.sourceforge.net

November 4, 2004

Contents

1 Introduction **1**
1.1 Overview . . . 1
1.2 Installation . . . 2
1.3 Installing Prebundled Packages . . . 2
1.3.1 Temporary Installation . . . 2
1.3.2 Installation in a texmf Tree . . . 2
1.4 Quick Start . . . 3
1.5 Gallery . . . 4

2 Basic Graphic Drawing **6**
2.1 Main Environments . . . 6
2.2 How to Specify a Point . . . 7
2.3 Coordinate Systems . . . 9
2.4 Path Construction . . . 10
2.5 Stroking and Filling . . . 11
2.6 Clipping . . . 13
2.7 Shape and Line Drawing . . . 14
2.8 Image Inclusion . . . 15
2.9 Text Drawing . . . 17
2.10 Drawing Arrows at Line Ends . . . 18
2.11 Placing Labels on Lines . . . 20
2.12 Shadings . . . 20

3 Using Nodes **22**
3.1 Node Creation . . . 22
3.2 Coordinates Relative to Nodes . . . 23
3.3 Connecting Nodes . . . 24
3.4 Placing Labels on Node Connections . . . 25

4 Extended Color Support **25**

1 Introduction

1.1 Overview

This user's guide explains the functionality of the PGF package. PGF stands for 'portable graphics format'. It is a TEX macro package that allows you to create graphics in your TEX documents using a special `pgfpicture` environment and special macros for drawing lines, curves, rectangles, and many other kind of graphic objects. Its usage closely resembles the PSTRICKS package or the normal picture environment of LATEX.

Although PGF is less powerful than PSTRICKS, it has the advantage that it can generate both PostScript output and PDF output from the same file. The PGF package works together both with `dvips` and `pdftex`. In particular, packages that rely on `pdftex` or `pdflatex` (like some packages for creating presentations) can be used together with PGF.

The package consists of the core style `pgf.sty` and a number of extension styles that are build on top of it. Currently, the documented ones are

- `pgfarrows.sty`, used to draw a large variety of arrows.
- `pgfnodes.sty`, used to draw nodes in diagrams and to connect them in a convenient way.
- `pgfshade.sty`, used to create shadings (also called gradients).

In order to use PGF you will have to include the command

```
\usepackage{pgf}
```

at the beginning of your main TEX file. If you also wish to use the extensions, you also have to include them. For example, you will typically use the following command:

```
\usepackage{pgf,pgfarrows,pgfnodes}
```

In this guide you will find the descriptions of all "public" commands provided by the `pgf` package. In each such description, the described command, environment or option is printed in red. Text shown in green is optional and can be left out. Note that (currently) many commands take arguments in square brackets that are *not* optional. In some future version of PGF it will possible to omit these optional arguments.

1.2 Installation

To use PGF, you just need to put all files with the ending `.sty` of the pgf package in a directory that is read by TEX. You need to have the package `xcolor` installed, version 2.00 or higher. To uninstall `pgf`, simply remove these files once more. Unfortunately, there are different ways of making TEX "aware" of files. Which way you should choose depends on how permanently you intend to use it.

1.3 Installing Prebundled Packages

I do not create or manage prebundled packages of PGF, but, fortunately, nice other people do. I cannot give detailed instructions on how to install these packages, since I do not manage them, but I *can* tell you were to find them. You install them the "usual way" you install packages. If anyone has any hints and additional information on this, please email me.

For Debian, you need the packages at

```
http://packages.debian.org/pgf
http://packages.debian.org/latex-xcolor
```

For MiKTEX, you need the packages called `pgf` and `xcolor`.

1.3.1 Temporary Installation

If you only wish to install PGF for a quick appraisal, do the following: Obtain all files from the directory `http://www.ctan.org/tex-archive/graphics/pgf/` (most likely, you have already done this). Place all files in a new directory. For example, `/home/tantau/pgf/` would work fine for me. Then setup the environment variable called `TEXINPUTS` to be the following string (how exactly this is done depends on your operating system and shell):

```
.:/home/tantau/pgf:
```

Naturally, if the `TEXINPUTS` variable is already defined differently, you should *add* the directories to the list. Do not forget to place a colon at the end (corresponding to an empty path), which will include all standard directories.

1.3.2 Installation in a texmf Tree

For a more permanent installation, you can place the files of the the PGF package (see the previous subsection on how to obtain them) in an appropriate `texmf` tree.

When you ask TEX to use a certain class or package, it usually looks for the necessary files in so-called `texmf` trees. These trees are simply huge directories that contain these files. By default, TEX looks for files in three different `texmf` trees:

- The root `texmf` tree, which is usually located at `/usr/share/`, `c:\texmf\`, or `c:\Program Files\TeXLive\texmf\`.
- The local `texmf` tree, which is usually located at `/usr/local/share/`, `c:\localtexmf\`, or `c:\Program Files\TeXLive\texmf-local\`.
- Your personal `texmf` tree, which is located in your home directory.

You should install the packages either in the local tree or in your personal tree, depending on whether you have write access to the local tree. Installation in the root tree can cause problems, since an update of the whole TEX installation will replace this whole tree.

Inside whatever `texmf` directory you have chosen, create the sub-sub-sub-directory `texmf/tex/latex/pgf` and place all files in it. Then rebuild TEX's filename database. This done by running the command `texhash` or `mktexlsr` (they are the same). In MikTeX, there is a menu option to do this.

If you want to be really tidy, you can place the documentation in the directory `texmf/doc/latex/pgf`.

For a more detailed explanation of the standard installation process of packages, you might wish to consult `http://www.ctan.org/installationadvice/`. However, note that the PGF package does not come with a `.ins` file (simply skip that part).

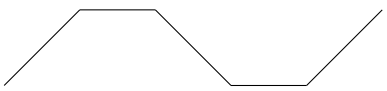
1.4 Quick Start

This section presents some simple examples. By copying these examples and modifying them slightly, you can create your first pictures using PGF.

The first example draws a rectangle and a circle next to each other.

```
\begin{pgfpicture}{0cm}{0cm}{5cm}{2cm}
  % (0cm,0cm) is the lower left corner,
  % (5cm,2cm) is the upper right corner.

  \pgfrect[stroke]{\pgfpoint{0cm}{0cm}}{\pgfpoint{2cm}{10pt}}
  % Paint a rectangle (stroke it, do not fill it)
  % The lower left corner is at (0cm,0cm)
  % The rectangle is 2cm wide and 10pt high.

  \pgfcircle[fill]{\pgfpoint{3cm}{1cm}}{10pt}
  % Paint a filled circle
  % The center is at (3cm,1cm)
  % The radius is 10pt
\end{pgfpicture}
```

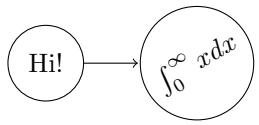
The `\pgfpoint` command is used to specify a point. It is often more convenient to use the command `pgfxy` instead, which lets you specify a point in terms of multiples of a x-vector and a y-vector. They are predefined to `\pgfpoint{1cm}{0cm}` and `\pgfpoint{0cm}{1cm}`, but you can change these settings.



```
\begin{pgfpicture}{0cm}{0cm}{5cm}{1.25cm}

  \pgfline{\pgfxy(0,0)}{\pgfxy(1,1)}
  % Draws a line from (0cm,0cm) to (1cm,1cm)
  % Command \pgfline{\pgfpoint{0cm}{0cm}}{\pgfpoint{1cm}{1cm}}
  % would have the same effect.

  \pgfline{\pgfxy(1,1)}{\pgfxy(2,1)}
  \pgfline{\pgfxy(2,1)}{\pgfxy(3,0)}
  \pgfline{\pgfxy(3,0)}{\pgfxy(4,0)}
```

```
  \pgfline{\pgfxy(4,0)}{\pgfxy(5,1)}
\end{pgfpicture}
```

You can put text into a picture using the `\pgfbox` command.



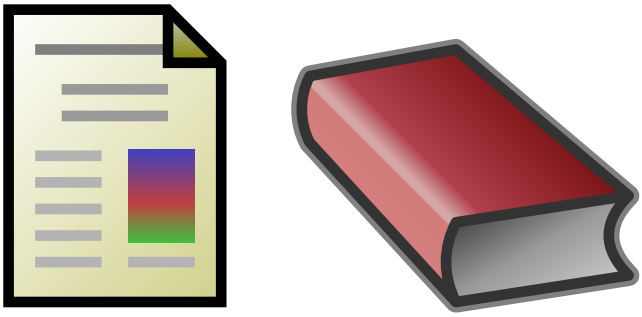

```
\begin{pgfpicture}{0cm}{0cm}{5cm}{2cm}
  \pgfputat{\pgfxy(1,1)}{\pgfbox[center,center]{Hi!}}
  % pgfputat places something at a certain position
  % pgfbox shows the text 'hi!'. The horizontal alignment
  % is centered (other options: left, right). The vertical
  % alignment is also centered (other options: top, bottom,
  % base).

  \pgfcircle[stroke]{\pgfxy(1,1)}{0.5cm}

  \pgfsetendarrow{\pgfarrowto}
  % In the following, all lines will end with an arrow that looks like
  % the arrow of TeX's \to command

  \pgfline{\pgfxy(1.5,1)}{\pgfxy(2.2,1)}

  \pgfputat{\pgfxy(3,1)}{
    \begin{pgfrotateby}{\pgfdegree{30}}
      % You can rotate things like this
      \pgfbox[center,center]{$\int_0^\infty xdx$}
    \end{pgfrotateby}}
  \pgfcircle[stroke]{\pgfxy(3,1)}{0.75cm}
\end{pgfpicture}
```

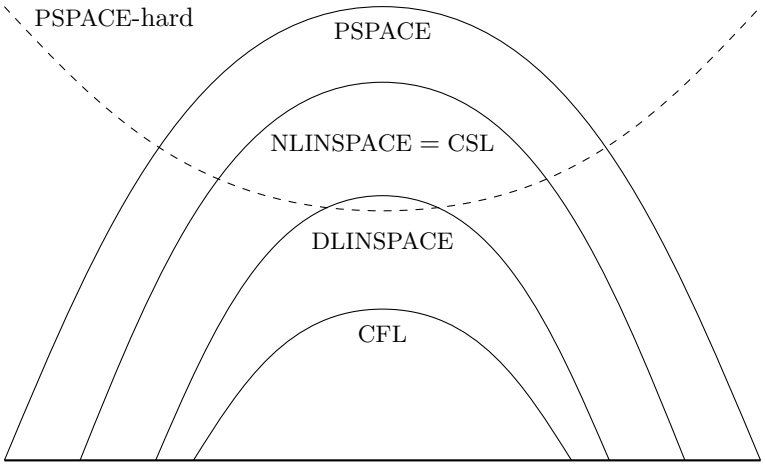
In order to draw curves and complicated lines, you can use the commands `pgfmoveto`, `pgflineto`, and `pgfcurveto`. To actually draw or fill the painted area, you use `pgfstroke` or `pgffill`.

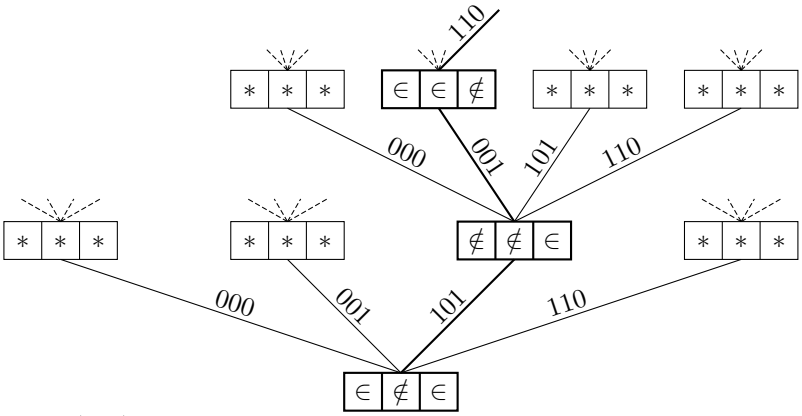
```
\begin{pgfpicture}{0cm}{0cm}{5cm}{2cm}
  \pgfmoveto{\pgfxy(0,1)}
  \pgfcurveto{\pgfxy(1,0.5)}{\pgfxy(1,1.5)}{\pgfxy(2,1)}
  \pgfstroke

  \pgfsetdash{{3pt}{3pt}}{0pt}
  \pgfmoveto{\pgfxy(0,1)}
  \pgflineto{\pgfxy(1,0.5)}
  \pgflineto{\pgfxy(1,1.5)}
  \pgflineto{\pgfxy(2,1)}
  \pgfstroke

  \pgfmoveto{\pgfxy(3,1)}
  \pgfcurveto{\pgfxy(3,0)}{\pgfxy(4,0)}{\pgfxy(4,1)}
  \pgfcurveto{\pgfxy(4,2)}{\pgfxy(3,2)}{\pgfxy(3,1)}
  \pgfclosepath
  \pgffill
\end{pgfpicture}
```

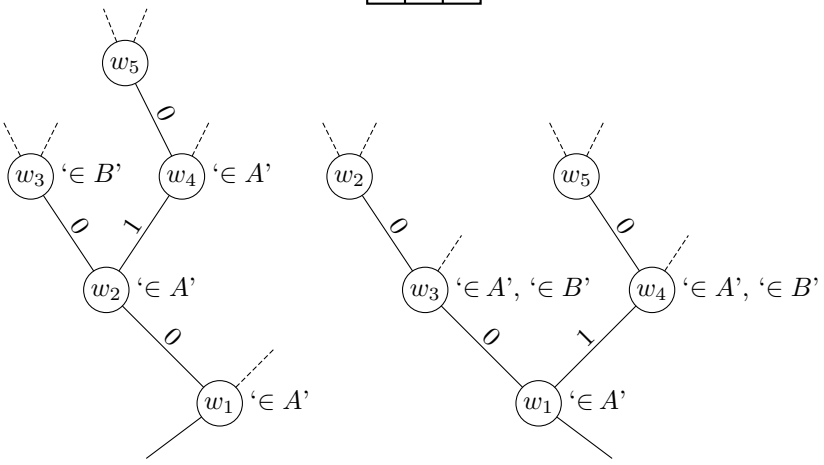
1.5 Gallery

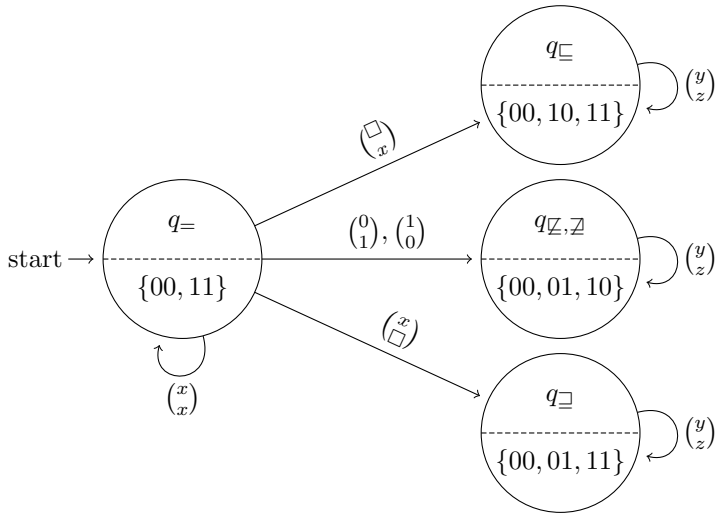
In the following, a number of figures are shown that have been created using PGF. Please see the source code for how they are created.

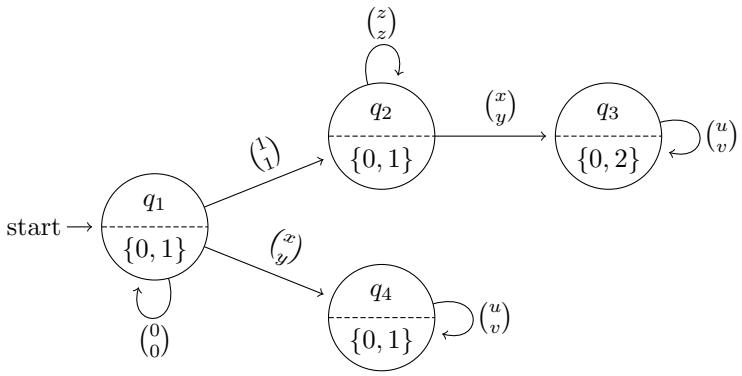


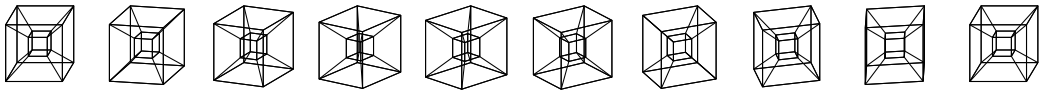

PSPACE-hard
PSPACE
NLINSPACE = CSL
DLINSPACE
CFL


110
000
001
101
110
000
001
101
110


w_5
0
w_3 '$\in B$'
w_4 '$\in A$'
0
1
w_2 '$\in A$'
0
w_1 '$\in A$'
w_2
0
w_3 '$\in A$', '$\in B$'
w_5
0
w_4 '$\in A$', '$\in B$'
0
1
w_1 '$\in A$'







2 Basic Graphic Drawing

2.1 Main Environments

In order to draw a picture using PGF, you have to put the picture inside the environment `pgfpicture` or the environment `pgfpictureboxed`.

`\begin{pgfpicture}{`⟨*lower left x*⟩`}{`⟨*lower left y*⟩`}{`⟨*upper right x*⟩`}{`⟨*upper right y*⟩`}`
⟨*environment contents*⟩
`\end{pgfpicture}`

Inserts a PGF-picture into the text. The sizes are used as follows: Think of the ⟨*environment contents*⟩ as commands that draw something on an infinite two-dimensional plane. After you are done with drawing, you "cut out" a rectangle from this plane whose lower left corner is at (⟨*lower left x*⟩, ⟨*lower left y*⟩) and whose upper right corner is at (⟨*upper right x*⟩, ⟨*upper right y*⟩). The size of this rectangle will hence be ⟨*upper right x*⟩ − ⟨*lower left x*⟩ times ⟨*upper right y*⟩ − ⟨*lower left y*⟩. This rectangle is then inserted into the normal text at the position where the `{pgfpicture}` environment is used.

The environment performs no clipping, thus if you draw outside the rectangle what you draw will protrude outside the area that TEX reserves for the picture.

Example:

```
\begin{pgfpicture}{0cm}{0cm}{1cm}{1cm}
  \pgfline{\pgforigin}{\pgfpoint{10pt}{10pt}}
```

```
\end{pgfpicture}
```

`\begin{pgfpictureboxed}`{⟨*lower left x*⟩}{⟨*lower left y*⟩}{⟨*upper right x*⟩}{⟨*upper right y*⟩}
⟨*environment contents*⟩
`\end{pgfpictureboxed}`

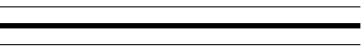
Identical to `pgfpicture`, except that a frame of the size of the picture is drawn around it.

Inside a picture, you can create nested scopes using `pgfscope`. Changes made inside a `pgfscope` are undone when the scope ends.

`\begin{pgfscope}`
⟨*environment contents*⟩
`\end{pgfscope}`

All changes made inside a scope are local to that scope.

Example:



```
\begin{pgfpicture}{0cm}{0cm}{5cm}{0.75cm}
  \pgfxyline(0,0)(5,0)
  \begin{pgfscope}
    \pgfsetlinewidth{2pt}
    \pgfxyline(0,0.25)(5,0.25)
  \end{pgfscope}
  \pgfxyline(0,0.5)(5,0.5)
\end{pgfpicture}
```

2.2 How to Specify a Point

PGF uses a two dimensional coordinate system that is local to the current picture been drawn. A point inside the coordinate system can be specified using the command `pgfpoint`. You can use all dimensions available in TEX when specifying a dimension.

`\pgforigin`

Yields the origin.

Example: `\pgmoveto{\pgforigin}`

`\pgfpoint`{⟨*x coordinate*⟩}{⟨*y coordinate*⟩}

Yields a point location. The coordinates are given as TEX dimensions.

Example: `\pgfline{\pgfpoint{10sp}{-1.5cm}}{\pgfpoint{10pt}{1cm}}`

`\pgfpolar`{⟨*degree*⟩}{⟨*radius*⟩}

Yields a point location given in polar coordinates.

Example: `\pgfmoveto{\pgfpolar{30}{1cm}}`

`\pgfdirection`{⟨*direction string*⟩}

Returns the degree that corresponds to the direction. Allowed values for ⟨*direction string*⟩ are `n[orth]`, `s[south]`, `e[east]`, `w[est]`, `ne`, `nw`, `se`, and `sw`.

Example: `\pgfmoveto{\pgfpolar{\pgfdirection{n}}{1cm}}`

`\pgfextractx`{⟨*dimension*⟩}{⟨*point*⟩}

Sets the TEX-⟨*dimension*⟩ to the x-coordinate of the point.

Example:

```
\newdimen\mydim
\pgfextractx{\mydim}{\pgfpoint{2cm}{4pt}}
% \mydim is now 2cm
```

\pgfextracty{⟨*dimension*⟩}{⟨*point*⟩}

Like \pgfextractx, except for the y-coordinate.

Coordinates can also be specified as multiples of an x-vector and a y-vector. Normally, the x-vector points one centimeter in the x-direction and the y-vector points one centimeter in the y-direction, but using the commands pgfsetxvec and pgfsetyvec they can be changed.

It is also possible to specify a point as a multiple of three vectors, the x-, y-, and z-vector. This is useful for creating simple three dimensional graphics.

\pgfxy(⟨s_x⟩,⟨s_y⟩)

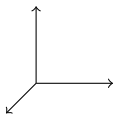
Yields a point that is situated at s_x times the x-vector plus s_y times the y-vector.

Example: \pgfline{\pgfxy(0,0)}{\pgfxy(1,1)}

\pgfxyz(⟨s_x⟩,⟨s_y⟩,⟨s_z⟩)

Yields a point that is situated at s_x times the x-vector plus s_y times the y-vector plus s_z times the z-vector.

Example:



```
\pgfsetendarrow{\pgfarrowto}
\pgfline{\pgfxyz(0,0,0)}{\pgfxyz(0,0,1)}
\pgfline{\pgfxyz(0,0,0)}{\pgfxyz(0,1,0)}
\pgfline{\pgfxyz(0,0,0)}{\pgfxyz(1,0,0)}
```

\pgfsetxvec{⟨*point*⟩}

A point that replaces the current x-vector. The commands \pgfsetyvec and \pgfsetzvec are defined the same way.

Example:

```
\pgfsetxvec{\pgfpoint{2cm}{0cm}}
\pgfline{\pgfxy(0,0)}{\pgfxy(1,1)}
% Same as \pgfline{\pgforigin}{\pgfpoint{2cm}{1cm}}
```

There exist different commands for treating points as vectors.

\pgfdiff{⟨*point* p_1⟩}{⟨*point* p_2⟩}

Yields the difference vector $p_2 - p_1$.

Example: \pgfmoveto{\pgfdiff{\pgfxy(1,1)}{\pgfxy(2,3)}}

\pgfrelative{⟨*point* p_1⟩}{⟨*point* p_2⟩}

Yields the the sum $p_1 + p_2$

Example: \pgfmoveto{\pgfrelative{\pgfxy(0,1)}{\pgfpoint{1pt}{2pt}}}

\pgfpartway{⟨*scalar* r⟩}{⟨*point* p_1⟩}{⟨*point* p_2⟩}

Yields a point that is the rth fraction between p_1 and $p2$, that is, $p_1 + r(p_2 - p_1)$. For $r = 0.5$ the middle between p_1 and p_2 is returned.

Example: \pgfmoveto{\pgfpartway{0.5}{\pgfxy(1,1)}{\pgfxy(2,3)}}

\pgfbackoff{⟨*distance*⟩}{⟨*start point*⟩}{⟨*end point*⟩}

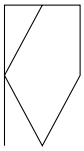
Yields a point that is located ⟨*distance*⟩ many units removed from the start point in the direction of the end point.

Example:

```
\pgfline{\pgfbackoff{2pt}{\pgfxy(1,1)}{\pgfxy(2,3)}}
        {\pgfbackoff{3pt}{\pgfxy(2,3)}{\pgfxy(1,1)}}
```

`\pgfcorner[`⟨*direction*⟩`]{`⟨*first point*⟩`}{`⟨*second point*⟩`}`

Image the rectangle whose corners are ⟨*first point*⟩ and second ⟨*second point*⟩. If you specify `se` as ⟨*direction*⟩ you will get the south-east (or lower left) corner of this rectangle. Similarly, `ne`, `nw`, and `sw` yield the other three corners. If you specify `s` for south, you get the middle of the lower side of the rectangle. Similarly for the other three directions `n`, `e`, and `w`.

Example: 

```
\pgfmoveto{\pgfcorner[sw]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[nw]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[ne]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[e]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[s]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[w]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgflineto{\pgfcorner[n]{\pgfpoint{2cm}{4pt}}{\pgfpoint{3cm}{2cm}}}
\pgfstroke
```

2.3 Coordinate Systems

Coordinate systems can be translated, rotated, and magnified using two environments. *Please note that these operations are incompatible with the node drawing commands.* Note also that the magnify operation also makes lines appear bigger. If this is not desired, you might wish to enlarge the x- and y-vectors instead.

`\begin{pgftranslate}{`⟨*new origin*⟩`}`
⟨*environment contents*⟩
`\end{pgftranslate}`

Makes ⟨*new origin*⟩ the new origin within the scope of the environment.

Example:

```
\begin{pgftranslate}{\pgfpoint{0cm}{1cm}}
  \pgfline{\pgforigin}{\pgfxy(1,0)}
\end{pgftranslate}
```

`\pgftranslateto{`⟨*new origin*⟩`}`

Makes the parameter the new origin.

Example:

```
\pgftranslateto{\pgfpoint{0cm}{1cm}}
\pgfline{\pgforigin}{\pgfxy(1,0)}
```

`\pgfputat{`⟨*an origin*⟩`}{`⟨*commands*⟩`}`

Executes the commands after having translated the origin to the given location.

Example: `\pgfputat{\pgfxy(1,0)}{\pgfbox[center,center]{Hello world}}`

`\begin{pgfrotateby}{`⟨*sin/cos of rotation degree*⟩`}`
⟨*environment contents*⟩
`\end{pgfrotateby}`

Rotates the current coordinate system by `{`⟨*sin/cos of rotation degree*⟩`}` within the scope of the environment. Use `\pgfdegree` to calculate the rotation degree.

Example:

```
\begin{pgfrotateby}{\pgfdegree{30}}
  \pgfline{\pgforigin}{\pgfxy(1,0)}
\end{pgfrotateby}
```

`\begin{pgfmagnify}{`⟨*x magnification*⟩`}{`⟨*y magnification*⟩`}`
⟨*environment contents*⟩

\end{pgfmagnify}

Magnifies everything within the environment by the given factors.

Example:

```
\begin{pgfmagnify}{2}{2}
  \pgfline{\pgforigin}{\pgfxy(1,0)}
\end{pgfmagnify}
```

2.4 Path Construction

Lines and shapes can be drawn by constructing paths and by then stroking and filling them. In order to construct a path, you must first use the command `\pgfmoveto`, followed by a series of `\pgflineto` and `\pgfcurveto` commands. You can use `\pgfclosepath` to create a closed shape. You can also use `\pgfmoveto` commands while constructing a path.

\pgfmoveto{⟨*point*⟩}

Makes ⟨*point*⟩ the current point.

Example: `\pgfmoveto{\pgforigin}`

\pgflineto{⟨*point*⟩}

Extends the path by a straight line from the current point to ⟨*point*⟩. This point is then made the current point.

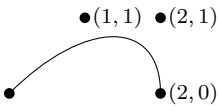
Example:

```
\pgfmoveto{\pgforigin}
\pgflineto{\pgfxy(1,1)}
```

\pgfcurveto{⟨*support point 1*⟩}{⟨*support point 2*⟩}{⟨*point*⟩}

Extends the path by a curve from the current point to ⟨*point*⟩. This point is then made the current point. The support points govern in which direction the curves head at the start and at the end. At the start it will head in a straight line towards {⟨*support point 1*⟩}, at the destination it will head in a straight line towards the destination as if it came from {⟨*support point 2*⟩}.

Example:




```
\pgfmoveto{\pgforigin}
\pgfcurveto{\pgfxy(1,1)}{\pgfxy(2,1)}{\pgfxy(2,0)}
\pgfstroke
```

\pgfclosepath

Connects the current point to the point where the current path started.

\pgfzerocircle{⟨*radius*⟩}

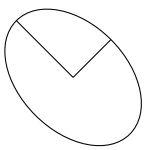
Appends a circle around the origin of the given radius to the current path.

Example: `\pgfzerocircle{1cm}`

\pgfzeroellipse{⟨*axis vector 1*⟩}{⟨*axis vector 2*⟩}

Appends an ellipse with the given axis vectors centered at the origin to the current path.

Example:



```
\pgfzeroellipse{\pgfxy(0.5,0.5)}{\pgfxy(-0.75,0.75)}
\pgfstroke
\pgfline{\pgforigin}{\pgfxy(0.5,0.5)}
\pgfline{\pgforigin}{\pgfxy(-0.75,0.75)}
```

The basic drawing commands also come in 'quick' versions. These versions get plain numbers as input that represent TEX points. These commands are executed much quicker than the normal commands. They are useful if you need to do construct very long or numerous paths.

\pgfqmoveto{⟨*x bp*⟩}{⟨*y bp*⟩}

Makes the given point the current point. The real numbers given are interpreted as TEX "big points," which are a 1/72th of an inch (as opposed to TEX points, which are a 1/72.27th of an inch).

Example: `\pgfqmoveto{10}{20}`

\pgfqlineto{⟨*x bp*⟩}{⟨*y bp*⟩}

Extends the path by a straight line from the current point to the parameter point. The parameter point is then made the current point.

Example:

```
\pgfqmoveto{0}{0}
\pgfqlineto{100}{100}
\pgfstroke
```

\pgfqcurveto{⟨s_x^1 *bp*⟩}{⟨s_y^1 *bp*⟩}{⟨s_x^2 *bp*⟩}{⟨s_y^2 *bp*⟩}{⟨*x bp*⟩}{⟨*y bp*⟩}

Quick version of the `\pgfcurveto` command.

Example:

```
\pgfqmoveto{0}{0}
\pgfqcurveto{100}{100}{200}{100}{200}{0}
\pgfstroke
```

2.5 Stroking and Filling

Once you have constructed a path, you can use the commands `\pgfstroke` and `\pgffill` to paint the path. How the path is painted depends on a number of parameters: For filling, the fill color is important (the fill color is the same as the stroke color and it set by using the standard `\color` commands from the `color` package or any compatible package). For stroking, the line width, the line dashing, the miter join, and the cap form are furthermore of importance.

\pgfstroke

Draws the current path with current color, thickness, dashing, miter, and cap. If an arrow type is set up, arrows are drawn at the beginning and at the end.

Example:

```
\pgfmoveto{\pgforigin}
\pgflineto{\pgfxy(1,1)}
\pgfstroke
```

\pgfqstroke

Like `\pgfstroke`, except that no arrows are drawn.

\pgfclosestroke

Closes the current path and then draws it.

Example:

```
\pgfmoveto{\pgforigin}
\pgflineto{\pgfxy(1,1)}
\pgflineto{\pgfxy(0,1)}
\pgfclosestroke
```

\pgffill

Closes the current path, if necessary, and then fill the area with the current color.

Example:

```
\pgfmoveto{\pgforigin}
\pgflineto{\pgfxy(1,1)}
\pgfstroke
```

\pgfeofill

Same as \pgffill, except that the even-odd rule is used.

\pgffillstroke

Strokes the current path, the closes the current path, if necessary, and then fills the area with the current color.

\pgfeofillstroke

Same as \pgffillstroke, except that the even-odd rule is used.

\pgfsetlinewidth{⟨*line width*⟩}

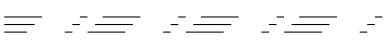
Sets the line width for subsequent stroking commands to ⟨*line width*⟩. A dimension of 0pt corresponds to the thinnest drawable line. On high resolution printers these will be impossible to see.

Example: \pgfsetlinewidth{3pt}

\pgfsetdash{⟨*list of even length of dimensions*⟩}{⟨*phase*⟩}

Sets the dashing of a line. The first entry in the list specifies the length of the first solid part of the list. The second entry specifies the length of the following gap. Then comes the length of the second solid part, following by the length of the second gap, and so on. The ⟨*phase*⟩ specifies where the first solid part starts relative to the beginning of the line.

Example:



```
\pgfsetdash{{0.5cm}{0.5cm}{0.1cm}{0.2cm}}{0cm}
\pgfxyline(0,1)(5,1)
\pgfsetdash{{0.5cm}{0.5cm}{0.1cm}{0.2cm}}{0.1cm}
\pgfxyline(0,0.9)(5,0.9)
\pgfsetdash{{0.5cm}{0.5cm}{0.1cm}{0.2cm}}{0.2cm}
\pgfxyline(0,0.8)(5,0.8)
```

\pgfsetbuttcap

Set a butt line cap for subsequent stroking commands.

\pgfsetroundcap

Set a round line cap for subsequent stroking commands.

\pgfsetrectcap

Set a rectangular line cap for subsequent stroking commands.

\pgfsetbeveljoin

Set a bevel line join for subsequent stroking commands.

\pgfsetroundjoin

Set a round line join for subsequent stroking commands.

\pgfsetmiterjoin

Set a miter line join for subsequent stroking commands.

\pgfsetmiterlimit{⟨*miter limit*⟩}

Sets the miter limit for subsequent stroking commands. See the PDF manual for details on what the miter limit is.

Example: \pgfsetmiterlimit{3pt}

2.6 Clipping

Paths can also be used to clip subsequent drawings. Executing the clip operator intersects the current clipping area with the area specified by the path. There is no way of enlarging the clipping area once more. However, if a clipping operations is done inside a pgfscope environment, the end of the scope restores the original clipping area.

\pgfclip

Closes the current path and intersect it with the current clipping path to form a new clipping path.

Example:

```
\pgfmoveto{\pgfxy(0,0)}
\pgflineto{\pgfxy(0,1)}
\pgflineto{\pgfxy(1,0)}
\pgfclip

\pgfcircle[fill]{\pgfxy(0.25,0.25)}{14pt}
```

\pgfstrokeclip

Stroke the current path, then close it, and intersect it with the current clipping path to form a new clipping path.

Example:

```
\pgfmoveto{\pgfxy(0,0)}
\pgflineto{\pgfxy(0,1)}
\pgflineto{\pgfxy(1,0)}
\pgfstrokeclip

\pgfcircle[fill]{\pgfxy(0.25,0.25)}{14pt}
```

\pgfclosestrokeclip

Close the current path, strokes it, and intersect it with the current clipping path to form a new clipping path.

Example:

```
\pgfmoveto{\pgfxy(0,0)}
\pgflineto{\pgfxy(0,1)}
\pgflineto{\pgfxy(1,0)}
\pgfclosestrokeclip

\pgfcircle[fill]{\pgfxy(0.25,0.25)}{14pt}
```

\pgffillclip

Closes the current path, fills it, and intersect it with the current clipping path to form a new clipping path.

\pgffillstrokeclip

Closes the current path, fills it, strokes it, and intersect it with the current clipping path to form a new clipping path.

2.7 Shape and Line Drawing

There are several commands that make drawing shapes and lines easier. However, in principle these could be implemented using the path construction and stroking and filling commands introduced above.

`\pgfline{`⟨*start point*⟩`}{`⟨*end point*⟩`}`

Draws a line from ⟨*start point*⟩ to ⟨*end point*⟩. This command is equivalent to constructing a path from the start to the end point and then stroking it.

Example: `\pgfline{\pgfxy(0,0)}{\pgfxy(1,1)}`

`\pgfxyline(`⟨x_1⟩`,`⟨y_1⟩`)(`⟨x_2⟩`,`⟨y_2⟩`)`

Like the `\pgfline` command, except the start and end points are given in xy-coordinates.

Example: `\pgfxyline(0,0)(1,1)`

`\pgfcurve{`⟨*start point*⟩`}{`⟨*support point 1*⟩`}{`⟨*support point 2*⟩`}{`⟨*end point*⟩`}`

Draws a curve from the start to the end point with given support points.

Example: `\pgfcurve{\pgfxy(0,0)}{\pgfxy(0,1)}{\pgfxy(1,1)}{\pgfxy(1,0)}`

`\pgfxycurve(`⟨x_1⟩`,`⟨y_1⟩`)(`⟨x_1'⟩`,`⟨y_1'⟩`)(`⟨x_2'⟩`,`⟨y_2'⟩`)(`⟨x_2⟩`,`⟨y_2⟩`)`

Like the `\pgfcurve` command, except that all points are given in xy-coordinates.

Example: `\pgfxycurve(0,0)(0,1)(1,1)(1,0)`

`\pgfrect[`⟨*drawing type*⟩`]{`⟨*lower left corner*⟩`}{`⟨*height/width vector*⟩`}`

Draws a rectangle. The ⟨*drawing type*⟩ can be `stroke`, `fill`, `fillstroke`, or `clip`.

Example:

```
% Draw a filled rectangle with corners (2,2) and (3,3)
\pgfrect[fill]{\pgfxy(2,2)}{\pgfxy(1,1)}
```

`\pgfcircle[`⟨*drawing type*⟩`]{`⟨*center*⟩`}{`⟨*radius*⟩`}`

Draws a circle centered at ⟨*center*⟩ of radius ⟨*radius*⟩. The ⟨*drawing type*⟩ can be `stroke`, `fill`, or `fillstroke`.

Example: `\pgfcircle[stroke]{\pgfxy(1,1)}{10pt}`

`\pgfellipse[`⟨*drawing type*⟩`]{`⟨*center*⟩`}{`⟨*axis vector 1*⟩`}{`⟨*axis vector 2*⟩`}`

Draws an ellipse at a given position. The drawing type can be `stroke`, `fill`, or `fillstroke`.

Example: `\pgfellipse[fill]{\pgforigin}{\pgfxy(2,0)}{\pgfxy(0,1)}`

`\pgfgrid[`⟨*options*⟩`]{`⟨*lower left*⟩`}{`⟨*upper right*⟩`}`

Draws a grid. The origin is part of the grid and the grid is clipped to the rectanlge specified by the ⟨*lower left*⟩ and the ⟨*upper right*⟩ corner. Allowed ⟨*options*⟩ are:

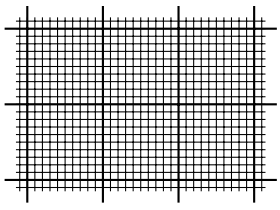
`stepx=`⟨*dimension*⟩ Sets the horizontal steping to ⟨*dimension*⟩. Default is 1cm.

`stepy=`⟨*dimension*⟩ Sets the vertical steping to ⟨*dimension*⟩. Default is 1cm.

`step=`⟨*vector*⟩ Sets the horizontal stepping to the x-coordinate of ⟨*vector*⟩ and the vertical steping its y-coordinate.

Example:

```
\pgfsetlinewidth{0.8pt}
\pgfgrid[step={\pgfpoint{1cm}{1cm}}]{\pgfxy(-.3,-.3)}{\pgfxy(3.3,2.3)}{}
\pgfsetlinewidth{0.4pt}
\pgfgrid[stepx=0.1cm,stepy=0.1cm]{\pgfxy(-.15,-.15)}{\pgfxy(3.15,2.15)}
```



2.8 Image Inclusion

The PGF package offers an abstraction of the image inclusion process, but you can still use the usual image inclusion facilities of the `graphics` package. The main reason why you might wish to use PGF's image inclusion instead is that file extensions are added automatically, depending on whether `.pdf` or `.dvi` is requested (this is important for packages that must work with both).

The general approach to including an image is the following: First, you use `\pgfdeclareimage` to declare the image. This must be done prior to the first use of the image. Once you have declared an image, you can insert it into the text using `\pgfuseimage`. The advantage of this two-phase approach is that, at least for PDF, the image data will only be included once in the file. This can drastically reduce the file size if you use an image repeatedly, for example in an overlay. However, there is also a command called `\pgfimage` that declares and then immediately uses the image.

`\pgfdeclareimage[`⟨*options*⟩`]{`⟨*image name*⟩`}{`⟨*filename*⟩`}`

Declares an image, but does not paint anything. To draw the image, use `\pgfuseimage{`⟨*image name*⟩`}`. The ⟨*filename*⟩ may not have an extension. For PDF, the extensions `.pdf`, `.jpg`, and `.png` will automatically tried. For PostScript, the extensions `.eps`, `.epsi`, and `.ps` will be tried.

The following options are possible:

- `height=`⟨*dimension*⟩ sets the height of the image. If the width is not specified simultaneously, the aspect ratio of the image is kept.
- `width=`⟨*dimension*⟩ sets the width of the image. If the height is not specified simultaneously, the aspect ratio of the image is kept.
- `page=`⟨*page number*⟩ selects a given page number from a multipage document. Specifying this option will have the following effect: first, PGF tries to find a file named

  ⟨*filename*⟩`.page`⟨*page number*⟩`.`⟨*extension*⟩

  If such a file is found, it will be used instead of the originally specified filename. If not, PGF inserts the image stored in ⟨*filename*⟩`.`⟨*extension*⟩ and if a recent version of `pdflatex` is used, only the selected page is inserted. For older versions of `pdflatex` and for `dvips` the complete document is inserted and a warning is printed.
- `interpolate=`⟨*true or false*⟩ selects whether the image should "smoothed" when zoomed. False by default.
- `mask=`⟨*mask name*⟩ selects a transparency mask. The mask must previously be declared using `\pgfdeclaremask` (see below). This option only has an effect for `pdf`. Not all viewers support masking.

Example:

```
\pgfdeclareimage[interpolate=true,height=1cm]{image1}{pgf-tu-logo}
\pgfdeclareimage[interpolate=true,width=1cm,height=1cm]{image2}{pgf-tu-logo}
\pgfdeclareimage[interpolate=true,height=1cm]{image3}{pgf-tu-logo}
```

`\pgfuseimage{`⟨*image name*⟩`}`

Inserts a previously declared image into the text. If you wish to use it in a picture environment, you should put a `\pgfbox` around it.

If the macro `\pgfalternateextension` expands to some nonempty ⟨*alternate extension*⟩, PGF will first try to use the image names ⟨*image name*⟩`.`⟨*alternate extension*⟩. If this image is not defined, PGF will next check whether ⟨*alternate extension*⟩ contains a `!` character. If so, everythings up to this exclamation mark and including it is deleted from ⟨*alternate extension*⟩ and the PGF again tries to use the image ⟨*image name*⟩`.`⟨*alternate extension*⟩. This is repeated until ⟨*alternate extension*⟩ no longer contains a `!`. Then the original image is used.

The `xxcolor` package sets the alternate extension to the current color mixin.

Example:

```
\begin{pgfpictureboxed}{0cm}{0cm}{7cm}{2.1cm}
  \pgfputat{\pgfxy(1,1)}{\pgfbox[left,base]{\pgfuseimage{image1}}}
  \pgfputat{\pgfxy(3,1)}{\pgfbox[left,base]{\pgfuseimage{image2}}}
```

```
  \pgfputat{\pgfxy(5,1)}{\pgfbox[left,base]{\pgfuseimage{image3}}}

  \pgfrect[stroke]{\pgfxy(1,1)}{\pgfxy(1,1)}
  \pgfrect[stroke]{\pgfxy(3,1)}{\pgfxy(1,1)}
  \pgfrect[stroke]{\pgfxy(5,1)}{\pgfxy(1,1)}

  \pgfputat{\pgfxy(1,0)}{\pgfbox[left,base]{Some text.}}
\end{pgfpictureboxed}
```

The following example demonstrates the effect of using `\pgfuseimage` inside a color mixin environment.

```
\pgfdeclareimage[interpolate=true,height=1cm]{image1.!25!white}{pgf-tu-logo.25}
\pgfdeclareimage[interpolate=true,width=1cm,height=1cm]{image2.!25!white}{pgf-tu-logo.25}
\pgfdeclareimage[interpolate=true,height=1cm]{image3.!25!white}{pgf-tu-logo.25}
\begin{colormixin}{25!white}
\begin{pgfpictureboxed}{0cm}{0cm}{7cm}{2.1cm}
  \pgfputat{\pgfxy(1,1)}{\pgfbox[left,base]{\pgfuseimage{image1}}}
  ... % as above
\end{pgfpictureboxed}
\end{colormixin}
```

`\pgfalternateextension`

You should redefine this command to install a different alternate extension.

Example: `\def\pgfalternateextension{!25!white}`

`\pgfaliasimage{`⟨*new image name*⟩`}{`⟨*existing image name*⟩`}`

The `{`⟨*existing image name*⟩`}` is "cloned" and the `{`⟨*new image name*⟩`}` can now be used whenever original image is used. This command is useful for creating aliases for alternate extensions and for accessing the last image inserted using `\pgfimage`.

Example: `\pgfaliasimage{image.!30!white}{image.!25!white}`

`\pgfimage[`⟨*options*⟩`]{`⟨*filename*⟩`}`

Declares the image under the name `pgflastimage` and immediately uses it. You can "save" the image for later usage by invoking `\pgfaliasimage` on `pgflastimage`.

Example:

```
\begin{pgfpictureboxed}{0cm}{0.9cm}{7cm}{2.1cm}
  \pgfputat{\pgfxy(1,1)}{\pgfbox[left,base]
    {\pgfimage[interpolate=true,width=1cm,height=1cm]{pgf-tu-logo}}}
  \pgfputat{\pgfxy(3,1)}{\pgfbox[left,base]
    {\pgfimage[interpolate=true,width=1cm]{pgf-tu-logo}}}
  \pgfputat{\pgfxy(5,1)}{\pgfbox[left,base]
    {\pgfimage[interpolate=true,height=1cm]{pgf-tu-logo}}}

  \pgfrect[stroke]{\pgfxy(1,1)}{\pgfxy(1,1)}
  \pgfrect[stroke]{\pgfxy(3,1)}{\pgfxy(1,1)}
  \pgfrect[stroke]{\pgfxy(5,1)}{\pgfxy(1,1)}
\end{pgfpictureboxed}
```

`\pgfdeclaremask[`⟨*options*⟩`]{`⟨*mask name*⟩`}{`⟨*filename*⟩`}`

Declares a transparency mask named ⟨*mask name*⟩ (called a *soft mask* in the PDF specification). This mask is read from the file ⟨*filename*⟩. This file should contain a grayscale image that is as large as the actual image. A white pixel in the mask will correspond to "transparent," a black pixel to "solid," and grey values correspond to intermediate values. The maks must have a single "color channel." This means that the maks must be a "real" grayscale image, not an RGB-image in which all RGB-triples happen to have the same components.

You can only mask images the are in a "pixel format." These are `.jpg` and `.png`. You cannot mask `.pdf` images in this way. Also, again, the mask file and the image file have to have the same size.

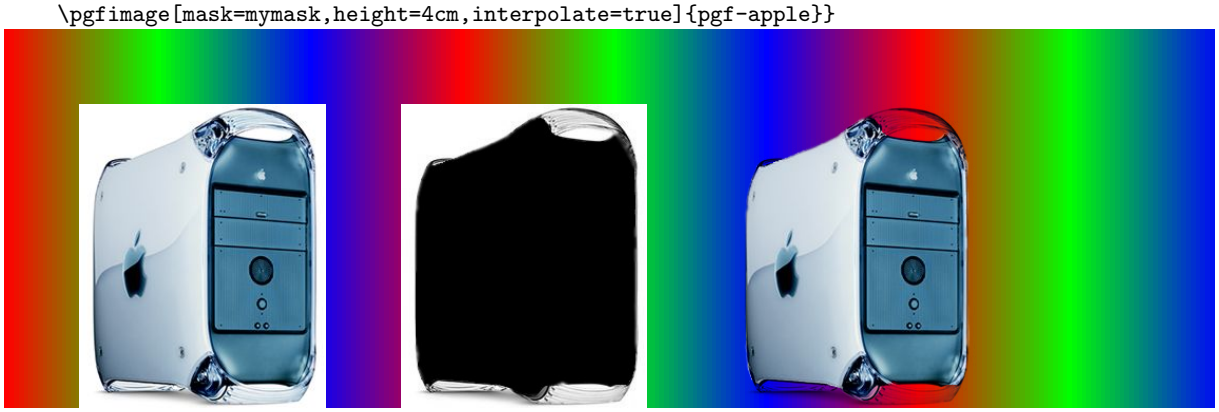
The following options may be given:

- `matte={`⟨*color components*⟩`}` sets the so-called *matte* of the actual image (strangely, this has to be specified together with the mask, not with the image itself). The matte is the color that has been used to preblend the image. For example, if the image has been preblended with a red background, then ⟨*color components*⟩ should be set to `{1 0 0}`. The default is `{1 1 1}`, which is white in the rgb model.

  The matte is specified in terms of the parent's image color space. Thus, if the parent is a grayscale image, the matte has to be set to `{1}`.

Example:

```
% Draw a large colorful background
\pgfdeclarehorizontalshading{colorful}{5cm}{color(0cm)=(red);
color(2cm)=(green); color(4cm)=(blue); color(6cm)=(red);
color(8cm)=(green); color(10cm)=(blue); color(12cm)=(red);
color(14cm)=(green); color(16cm)=(blue)}
\hbox{\pgfuseshading{colorful}\hskip-16cm\hskip1cm
\pgfimage[height=4cm]{pgf-apple}\hskip1cm
\pgfimage[height=4cm]{pgf-apple.mask}\hskip1cm
\pgfdeclaremask{mymask}{pgf-apple.mask}
\pgfimage[mask=mymask,height=4cm,interpolate=true]{pgf-apple}}
```



To speedup the compilation, you may wish to use the following class option:

`\usepackage[draft]{pgf}`

In draft mode boxes showing the image name replace the images. It is checked whether the image files exist, but they are not read. If either height or width is not given, 1cm is used instead.

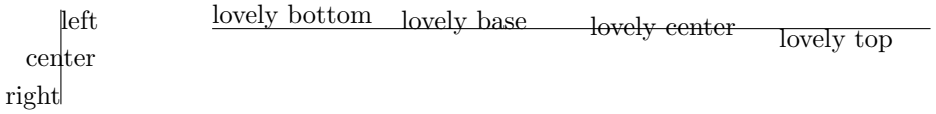
2.9 Text Drawing

In order to draw text, you must use the `pgfbox` command. It draws some text with a given alignment at the origin. Typically, you will use a `pgfputat` to put the text at some other location instead.

`\pgfbox[`⟨*horizontal alignment*⟩`,`⟨*vertical alignment*⟩`]{`⟨*TEX text*⟩`}`

Draws the given text with the given alignment at the origin. Allowed alignments are `left`, `center`, and `right` horizontally; and `bottom`, `base` (the base line of the text), `center`, and `top` vertically.

Example:




```
\pgfxyline(1,1.25)(1,0)
\pgfputat{\pgfxy(1,1)}{\pgfbox[left,base]{left}}
\pgfputat{\pgfxy(1,0.5)}{\pgfbox[center,base]{center}}
\pgfputat{\pgfxy(1,0)}{\pgfbox[right,base]{right}}

\pgfxyline(3,1)(12.5,1)
\pgfputat{\pgfxy(3,1)}{\pgfbox[left,bottom]{lovely bottom}}
\pgfputat{\pgfxy(5.5,1)}{\pgfbox[left,base]{lovely base}}
\pgfputat{\pgfxy(8,1)}{\pgfbox[left,center]{lovely center}}
\pgfputat{\pgfxy(10.5,1)}{\pgfbox[left,top]{lovely top}}
```

2.10 Drawing Arrows at Line Ends

When you stroke a line or curve, PGF can append arrows at the start and at the end of the line or curve. There is a wide variety of arrows available.

`\pgfsetstartarrow{`⟨*arrow type*⟩`}`

Henceforth, the specified ⟨*arrow type*⟩ is added to all stroked lines and curves. This does not apply to lines constructed using quick commands or lines that are stroked using `\pgfqstroke`. The allowed arrow types are listed below.

Example:


```
\pgfsetstartarrow{\pgfarrowto}
\pgfsetendarrow{\pgfarrowsingle}
\pgfxycurve(0,0.25)(0.5,0.5)(1,0)(1.5,0.25)
```

`\pgfsetendarrow{`⟨*arrow type*⟩`}`

Like `\pgfsetstartarrow`, except that the type of arrow at the end is specified.

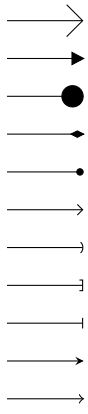
`\pgfclearstartarrow`

Clears the setting for the start arrows.

`\pgfclearendarrow`

Clears the setting for the end arrows.

The arrow types are explained below. Some arrow types take a parameter that govern its size.

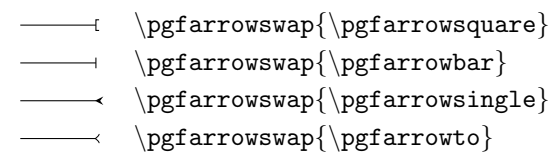


You can build more complicated arrow types by applying the following modifiers.

\pgfarrowswap{⟨*arrow type*⟩}

Yields an arrow type that has a swapped direction.

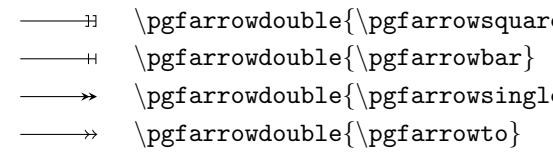
Example:



\pgfarrowdouble{⟨*arrow type*⟩}

Yields an arrow type that doubles the given arrow.

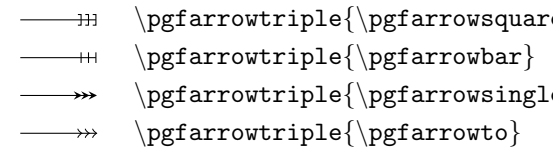
Example:



\pgfarrowtriple{⟨*arrow type*⟩}

Yields an arrow type that triples the given arrow.

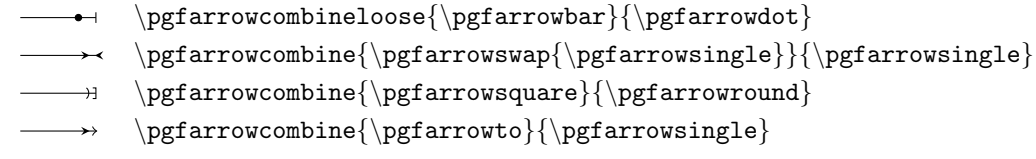
Example:



\pgfarrowcombine{⟨*first arrow type*⟩}{⟨*second arrow type*⟩}

Yields an arrow type that is made up from the two given arrow types, one after the other. The command \pgfarrowcombineloose does the same, but gives more spacing.

Example:



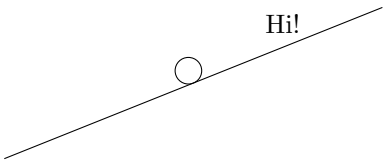
2.11 Placing Labels on Lines

Two commands can be used to place labels on lines.

\pgflabel{⟨*fraction*⟩}{⟨*start point*⟩}{⟨*end point*⟩}{⟨*orthogonal offset*⟩}

This command yields a position for placing a label on a straight line between two points. Note that this command does not draw a line; it only yields a position. The ⟨*offset*⟩ is orthogonal to the line. A ⟨*fraction*⟩ of 0 means ⟨*start point*⟩, 1 means ⟨*end point*⟩, and 0.5 means the middle.

Example:

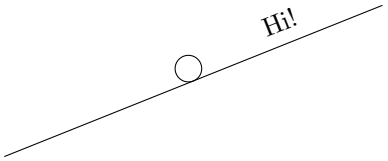



```
\pgfxyline(0,0)(5,2)
\pgfputat
  {\pgflabel{.5}{\pgfxy(0,0)}{\pgfxy(5,2)}{1pt}}
  {\pgfcircle[stroke]{\pgforigin}{5pt}}
\pgfputat{\pgflabel{.75}{\pgfxy(0,0)}{\pgfxy(5,2)}{5pt}}{\pgfbox[center,base]{Hi!}}
```

\pgfputlabelrotated{⟨*fraction*⟩}{⟨*start point*⟩}{⟨*end point*⟩}{⟨*orthogonal offset*⟩}{⟨*commands*⟩}

This command executes the graphics commands, after having translated are rotated the coordinate system to the label position on a straight line between the two end points.

Example:




```
\pgfxyline(0,0)(5,2)
\pgfputlabelrotated{.5}{\pgfxy(0,0)}{\pgfxy(5,2)}{1pt}
  {\pgfcircle[stroke]{\pgforigin}{5pt}}
\pgfputlabelrotated{.75}{\pgfxy(0,0)}{\pgfxy(5,2)}{5pt}{\pgfbox[center,base]{Hi!}}
```

2.12 Shadings

The package `pgfshade` can be used to create shadings. A shading is an area in which the color changes smoothly between different colors. Note that you need a recent version of `pdflatex` for the shadings to work in PDF. Note also that `ghostview` may do a poor job at displaying shadings when doing anti-aliasing.

Similarly to an image, a shading must first be declared before it can be used. Also similarly to an image, a shading is put into a TEX-box. Hence, in order to include a shading in a `pgfpicture`, you have to place it in a `\pgfbox`.

There are three kinds of shadings: horizontal, vertical, and radial shadings. However, you can rotate and clip shadings like any other graphics object, which allows you to create more complicated shadings. Horizontal shadings could be created by rotating a vertical shading by 90 degrees, but explicit commands for creating both horizontal and vertical shadings are included for convenience.

Once you have declared a shading, you can insert it into text using the command `\pgfuseshading`.

A horizontal shading is a horizontal bar of a certain height whose color changes smoothly. You must at least specify the colors at the left and at the right end of the bar, but you can also add color specifications for points in the middle. For example, suppose you which to create a bar that is red at the left end, green in the middle, and blue at the end. Suppose you would like the bar to be 4cm long. This could be specified as follows:

```
rgb(0cm)=(1,0,0); rgb(2cm)=(0,1,0); rgb(4cm)=(0,0,1)
```

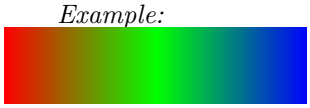
This line means that at 0cm (the left end) of the bar, the color should be red, which has red-green-blue (rgb) components (1,0,0). At 2cm, the bar should be green, and at 4cm it should be blue. Instead of `rgb`, you can currently also specify `gray` as color model, in which case only one value is needed, or `color`, in which case

you must provide the name of a color in round brackets. In a color specification the individual specifications must be separated using a semicolon, which may be followed by a whitespace (like a space or a newline). Individual specifications must be given in increasing order.

`\pgfdeclarehorizontalshading[`⟨*color list*⟩`]{`⟨*shading name*⟩`}{`⟨*shading height*⟩`}{`⟨*color specification*⟩`}`

Declares a horizontal shading named ⟨*shading name*⟩ of the specified ⟨*height*⟩ with the specified colors. The length of the bar is automatically deduced from the maximum specification.

Example:

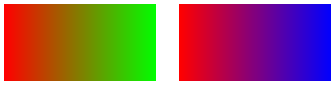


```
\pgfdeclarehorizontalshading{myshading}{1cm}%
  {rgb(0cm)=(1,0,0); color(2cm)=(green); color(4cm)=(blue)}
\pgfuseshading{myshading}
```

The effect of the ⟨*color list*⟩, which is a comma-separated list of colors, is the following: Normally, when this list is empty, once a shading is declared it becomes "frozen." This means that even if you change a color that was used in the declaration of the shading later on, the shading will not change. By specifying a ⟨*color list*⟩ you can specify that the shading should be recalculated whenever one of the colors listed in the list changes (this includes effects like color mixins). Thus, when you specify a ⟨*color list*⟩, whenever the shading is used, PGF first converts the colors in the list to RGB triples using the current values of the colors and taking any mixins and blendings into account. If the resulting RGB triples have not yet been used, a new shading is internally created and used. Note that if the option ⟨*color list*⟩ is used, then no shading is created until the first use of `\pgfuseshading`. In particular, the colors mentioned in the shading need not be defined when the declaration is given.

When a shading is recalculated because of a change in the colors mentioned in ⟨*color list*⟩, the complete shading is recalculated. Thus even colors not mentioned in the list will be used with their current values, not with the values they had upon declaration.

Example:

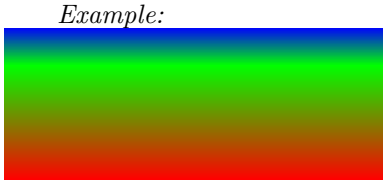


```
\pgfdeclarehorizontalshading[mycolor]{myshading}{1cm}{rgb(0cm)=(1,0,0); color(2cm)=(mycolor)}
\colorlet{mycolor}{green}
\pgfuseshading{myshading}
\colorlet{mycolor}{blue}
\pgfuseshading{myshading}
```

`\pgfdeclareverticalshading[`⟨*color list*⟩`]{`⟨*shading name*⟩`}{`⟨*shading width*⟩`}{`⟨*color specification*⟩`}`

Declares a vertical shading named ⟨*shading name*⟩ of the specified ⟨*width*⟩. The height of the bar is automatically deduced from the maximum specification. The effect of color list is the same as for horizontal shadings.

Example:



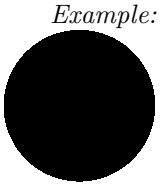
```
\pgfdeclareverticalshading{myshading}{5cm}%
  {rgb(0cm)=(1,0,0); rgb(1.5cm)=(0,1,0); rgb(2cm)=(0,0,1)}
\pgfuseshading{myshading}
```

`\pgfdeclareradialshading[`⟨*color list*⟩`]{`⟨*shading name*⟩`}{`⟨*center point*⟩`}{`⟨*color specification*⟩`}`

Declares an radial shading. A radial shading is a circle whose inner color changes as specified by the color specification. Assuming that the center of the shading is at the origin, the color of the center will be the color specified for 0cm and the color of the border of the circle will be the color for the maximum specification. The radius of the circle will be the maximum specification. If the center coordinate is not at the origin, the whole shading inside the circle (whose size remains exactly the same) will be distorted

such that the given center now has the color specified for 0cm. The effect of color list is the same as for horizontal shadings.

Example:



```
\pgfdeclareradialshading{sphere}{\pgfpoint{0.5cm}{0.5cm}}%
 {rgb(0.3cm)=(0.9,0,0);
  rgb(0.7cm)=(0.7,0,0);
  rgb(1cm)=(0.5,0,0);
  rgb(1.05cm)=(1,1,1)}
\pgfuseshading{sphere}
```

\pgfuseshading{⟨*shading name*⟩}

Inserts a previously declared shading into the text. If you wish to use it in a `pgfpicture` environment, you should put a `\pgfbox` around it. Like `\pgfuseimage`, alternate extensions are tried before the actual shading is used.

Example:

```
\pgfputat{\pgfxy(1,1)}{\pgfbox[center,center]{\pgfuseshading{myshading}}}
```

\pgfaliasshading{⟨*new shading name*⟩}{⟨*existing shading name*⟩}

The ⟨*existing shading name*⟩ is "cloned" and the shading ⟨*new shading name*⟩ can now be used whenever original shading is used. This command is mainly useful for creating aliases for environments that use alternate extensions.

Example: `\pgfaliasshading{shading!30}{shading!25}`

3 Using Nodes

The package `pgfnodes` allows you to draw all sorts of graphs in a convenient way. You draw them by first defining *nodes*. Once you have defined a node, you can connect nodes using lines or curves. The advantage of using nodes is that if, later on, you decide to move a node slightly, all connecting lines 'follow' automatically.

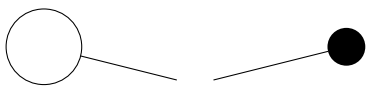
3.1 Node Creation

In all of the following command, the possible drawing types are `stroke`, `fill`, `fillstroke`, and `virtual` (draws nothing).

\pgfnodecircle{⟨*node name*⟩}[⟨*drawing type*⟩]{⟨*center*⟩}{⟨*radius*⟩}

Creates a circular node with the given radius at the given position.

Example:

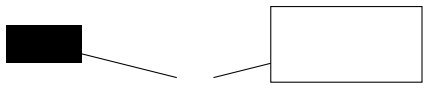


```
\pgfnodecircle{Node1}[stroke]{\pgfxy(1,1)}{0.5cm}
\pgfnodecircle{Node2}[virtual]{\pgfxy(3,0.5)}{0.25cm}
\pgfnodecircle{Node3}[fill]{\pgfxy(5,1)}{0.25cm}

\pgfnodeconnline{Node1}{Node2}
\pgfnodeconnline{Node2}{Node3}
```

\pgfnoderect{⟨*node name*⟩}[⟨*drawing type*⟩]{⟨*center*⟩}{⟨*width/height vector*⟩}

Creates a rectangular node with the width and height that is centered at the given position.

Example:

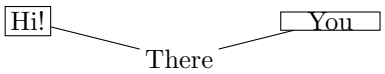


```
\pgfnoderect{Node1}[fill]{\pgfxy(1,1)}{\pgfxy(1,0.5)}
\pgfnodecircle{Node2}[virtual]{\pgfxy(3,0.5)}{0.25cm}
\pgfnoderect{Node3}[stroke]{\pgfxy(5,1)}{\pgfxy(2,1)}

\pgfnodeconnline{Node1}{Node2}
\pgfnodeconnline{Node2}{Node3}
```

\pgfnodebox{⟨*node name*⟩}[⟨*drawing type*⟩]{⟨*center*⟩}{⟨*TEX text*⟩}{⟨*horiz. margin*⟩}{⟨*vert. margin*⟩}

Creates a rectangular node that is centered at ⟨*center*⟩. The size of the node is calculated from the size of the box that is placed inside. The margins can be used to leave a little space around the text.

Example:




```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,1)}{Hi!}{2pt}{2pt}
\pgfnodebox{Node2}[virtual]{\pgfxy(3,0.5)}{There}{2pt}{2pt}
\pgfnodebox{Node3}[stroke]{\pgfxy(5,1)}{You}{10pt}{0pt}

\pgfnodeconnline{Node1}{Node2}
\pgfnodeconnline{Node2}{Node3}
```

3.2 Coordinates Relative to Nodes

\pgfnodecenter{⟨*node name*⟩}

Yields the center of a node. This command is especially useful for placing nodes relative to other nodes.

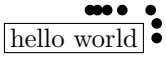
Example:

```
\pgfnodecircle{Node1}[stroke]{\pgfxy(1,0.5)}{0.25cm}
\pgfnodecircle{Node2}[stroke]
  {\pgfrelative{\pgfxy(1,0)}{\pgfnodecenter{Node1}}}{0.25cm}
\pgfnodecircle{Node3}[stroke]
  {\pgfrelative{\pgfxy(1,0)}{\pgfnodecenter{Node2}}}{0.25cm}
\pgfnodecircle{Node4}[stroke]
  {\pgfrelative{\pgfxy(1,0)}{\pgfnodecenter{Node3}}}{0.25cm}
```

\pgfnodeborder{⟨*node name*⟩}{⟨*angle*⟩}{⟨*border offset*⟩}

Returns a position on the border of the node named ⟨*node name*⟩ at an angle of ⟨*angle*⟩ (in degrees). For a positive offset, the position is removed from the border by the amount of the offset.

Example:



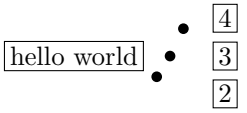

```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello world}{2pt}{2pt}

\pgfcircle[fill]{\pgfnodeborder{Node1}{0}{5pt}}{2pt}
```

```
\pgfcircle[fill]{\pgfnodeborder{Node1}{10}{5pt}}{2pt}
\pgfcircle[fill]{\pgfnodeborder{Node1}{20}{5pt}}{2pt}
\pgfcircle[fill]{\pgfnodeborder{Node1}{30}{5pt}}{2pt}
\pgfcircle[fill]{\pgfnodeborder{Node1}{40}{5pt}}{2pt}
\pgfcircle[fill]{\pgfnodeborder{Node1}{50}{5pt}}{2pt}
\pgfcircle[fill]{\pgfnodeborder{Node1}{60}{5pt}}{2pt}
```

\pgfconnstart[⟨*border offset*⟩]{⟨*start node*⟩}{⟨*end node*⟩}

Returns a position on the border of the first node for a line in the direction of the second node.

Example:



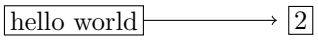

```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello world}{2pt}{2pt}
\pgfnodebox{Node2}[stroke]{\pgfxy(3,0)}{2}{2pt}{2pt}
\pgfnodebox{Node3}[stroke]{\pgfxy(3,0.5)}{3}{2pt}{2pt}
\pgfnodebox{Node4}[stroke]{\pgfxy(3,1)}{4}{2pt}{2pt}

\pgfcircle[fill]{\pgfnodeconnstart[5pt]{Node1}{Node2}}{2pt}
\pgfcircle[fill]{\pgfnodeconnstart[10pt]{Node1}{Node3}}{2pt}
\pgfcircle[fill]{\pgfnodeconnstart[15pt]{Node1}{Node4}}{2pt}
```

3.3 Connecting Nodes

\pgfnodesetsepstart{⟨*offset*⟩}

Sets the offset for the start of lines that are drawn using the below node connection commands. Use \pgfnodesetsepend for setting the end offset.

Example:




```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello world}{2pt}{2pt}
\pgfnodebox{Node2}[stroke]{\pgfxy(4,.5)}{2}{2pt}{2pt}

\pgfnodesetsepstart{0pt}
\pgfnodesetsepend{5pt}
\pgfsetendarrow{\pgfarrowto}

\pgfnodeconnline{Node1}{Node2}
```

\pgfnodeconnline{⟨*start node*⟩}{⟨*end node*⟩}

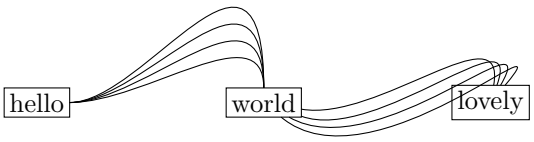
Draws a straight line from the border of the first node to the border of the second node.

Example: `\pgfnodeconnline{A}{B}`

\pgfnodeconncurve{⟨*start node*⟩}{⟨*end node*⟩}{⟨*start angle*⟩}{⟨*end angle*⟩}{⟨d_1⟩}{⟨d_2⟩}

Draws a curve from the ⟨*start node*⟩ to the ⟨*end node*⟩. The curve will start at the ⟨*start angle*⟩ on the border of the ⟨*start node*⟩. It ends at angle ⟨*end angle*⟩ on the border of the {⟨*end node*⟩}. The parameters d_1 and d_2 are the distances of the first, respectively second, support point from the border of the first, respectively second, node.

Example:




```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello}{2pt}{2pt}
```

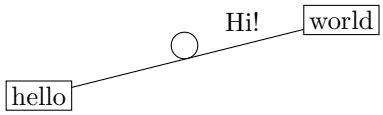
```
\pgfnodebox{Node2}[stroke]{\pgfxy(4,.5)}{world}{2pt}{2pt}
\pgfnodebox{Node3}[stroke]{\pgfxy(7,.5)}{lovely}{2pt}{2pt}

\pgfnodeconncurve{Node1}{Node2}{0}{90}{1cm}{1cm}
\pgfnodeconncurve{Node1}{Node2}{0}{90}{1cm}{1.5cm}
\pgfnodeconncurve{Node1}{Node2}{0}{90}{1cm}{2cm}
\pgfnodeconncurve{Node1}{Node2}{0}{90}{1cm}{2.5cm}

\pgfnodeconncurve{Node2}{Node3}{-10}{80}{1cm}{1cm}
\pgfnodeconncurve{Node2}{Node3}{-20}{70}{1cm}{1cm}
\pgfnodeconncurve{Node2}{Node3}{-30}{60}{1cm}{1cm}
\pgfnodeconncurve{Node2}{Node3}{-40}{50}{1cm}{1cm}
```

3.4 Placing Labels on Node Connections

\pgfnodelabel{⟨*start node*⟩}{⟨*end node*⟩}{⟨*fraction*⟩}[⟨*vertical offset*⟩]{⟨*command*⟩}

This command places a label at the given ⟨*fraction*⟩ of a straight line between two nodes.

Example:

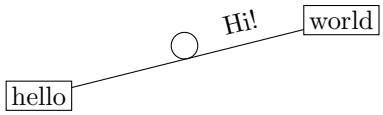



```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello}{2pt}{2pt}
\pgfnodebox{Node2}[stroke]{\pgfxy(5,1.5)}{world}{2pt}{2pt}

\pgfnodeconnline{Node1}{Node2}
\pgfnodelabel{Node1}{Node2}[0.5][5pt]{\pgfcircle[stroke]{\pgforigin}{5pt}}
\pgfnodelabel{Node1}{Node2}[0.75][5pt]{\pgfbox[center,base]{Hi!}}
```

\pgfnodelabelrotated{⟨*start node*⟩}{⟨*end node*⟩}{⟨*fraction*⟩}[⟨*vertical offset*⟩]{⟨*command*⟩}

This command places a rotated label at the given ⟨*fraction*⟩ of a straight line between two nodes. The label is rotated according to the slope of the line.

Example:




```
\pgfnodebox{Node1}[stroke]{\pgfxy(1,0.5)}{hello}{2pt}{2pt}
\pgfnodebox{Node2}[stroke]{\pgfxy(5,1.5)}{world}{2pt}{2pt}

\pgfnodeconnline{Node1}{Node2}
\pgfnodelabelrotated{Node1}{Node2}[0.5][5pt]{\pgfcircle[stroke]{\pgforigin}{5pt}}
\pgfnodelabelrotated{Node1}{Node2}[0.75][5pt]{\pgfbox[center,base]{Hi!}}
```

4 Extended Color Support

This section documents the package `xxcolor`, which is currently distributed as part of PGF. This package extends the `xcolor` package, written by Uwe Kern, which in turn extends the `color` package. I hope that the commands in `xxcolor` will some day migrate to `xcolor`, such that this package becomes superfluous.

The main aim of the `xxcolor` package is to provide an environment inside which all colors are "washed out" or "dimmed." This is useful in numerous situations and must typically be achieved in a roundabout manner if such an environment is not available.

\begin{colormixin}{⟨*mix-in specification*⟩}
⟨*environment contents*⟩

\end{colormixin}

The mix-in specification is applied to all colors inside the environment. At the beginning of the environment, the mix-in is applied to the current color, i. e., the color that was in effect before the environment started. A mix-in specification is a number between 0 and 100 followed by an exclamation mark and a color name. When a `\color` command is encountered inside a mix-in environment, the number states what percentage of the desired color should be used. The rest is "filled up" with the color given in the mix-in specification. Thus, a mix-in specification like `90!blue` will mix in 10% of blue into everything, whereas `25!white` will make everything nearly white.

Example:

```
\color{red}Red text,%
\begin{colormixin}{25!white}
  washed-out red text,
  \color{blue} washed-out blue text,
  \begin{colormixin}{25!black}
    dark washed-out blue text,
    \color{green} dark washed-out green text,%
  \end{colormixin}
  back to washed-out blue text,%
\end{colormixin}
and back to red.
```

Red text, washed-out red text, washed-out blue text, dark washed-out blue text, dark washed-out green text, back to washed-out blue text, and back to red.

Note that the environment only changes colors that have been installed using the standard LaTeX `\color` command. In particular, the colors in images are not changed. There is, however, some support offered by the commands `\pgfuseimage` and `\pgfuseshading`. If the first command is invoked inside a `colormixin` environment with the parameter, say, `50!black` on an image with the name `foo`, the command will first check whether there is also a defined image with the name `foo.!50!black`. If so, this image is used instead. This allows you to provide a different image for this case. If you nest `colormixin` environments, the different mix-ins are appended as a comma-separated list. For example, inside the inner environment of the above example, `\pgfuseimage{foo}` would first check whether there exists an image named `foo.!50!white!25!black`.

\colorcurrentmixin

Expands to the current accumulated mix-in. Each nesting of a `colormixin` adds a mix-in to this list.

Example:

```
\begin{colormixin}{25!white}
  \colorcurrentmixin is now ``25!white''
  \begin{colormixin}{75!black}
    \colorcurrentmixin is now ``75!black!25!white''
    \begin{colormixin}{50!white}
      \colorcurrentmixin is now ``50!white!75!black!25!white''
    \end{colormixin}
  \end{colormixin}
\end{colormixin}
```