TikZ & PGF

TikZ 3.0.1a 中文手册

```
\begin{tikzpicture}
  \coordinate (front) at (0,0);
  \coordinate (horizon) at (0,.31\paperheight);
  \coordinate (bottom) at (0,-.6\paperheight);
  \coordinate (sky) at (0,.57\paperheight);
  \coordinate (left) at (-.51\paperwidth,0);
  \coordinate (right) at (.51\paperwidth,0);

  \shade [bottom color=white,
          top color=blue!30!black!50]
              ([yshift=-5mm]horizon -|  left)
    rectangle (sky -| right);

  \shade [bottom color=black!70!green!25,
          top color=black!70!green!10]
    (front -| left) -- (horizon -| left)
    decorate [decoration=random steps] {
      -- (horizon -| right)  }
    -- (front -| right) -- cycle;

  \shade [top color=black!70!green!25,
         bottom color=black!25]
              ([yshift=-5mm-1pt]front -| left)
    rectangle ([yshift=1pt]front -| right);

  \fill [black!25]
              (bottom -| left)
    rectangle ([yshift=-5mm]front -| right);

  \def\nodeshadowed[#1]#2;{
    \node[scale=2,above,#1]{
      \global\setbox\mybox=\hbox{#2}
      \copy\mybox};
    \node[scale=2,above,#1,yscale=-1,
          scope fading=south,opacity=0.4]{\box\mybox};
  }

  \nodeshadowed [at={(-5,8  )},yslant=0.05]
    {\Huge Ti\textcolor{orange}{\emph{k}}Z};
  \nodeshadowed [at={( 0,8.3)}]
    {\huge \textcolor{green!50!black!50}{\&}};
  \nodeshadowed [at={( 5,8  )},yslant=-0.05]
    {\Huge \textsc{PGF}};
  \nodeshadowed [at={( 0,5  )}]
    {Manual for Version \pgftypesetversion};

  \foreach \where in {-9cm,9cm} {
    \nodeshadowed [at={(\where,5cm)}] { \tikz
      \draw [green!20!black, rotate=90,
             l-system={rule set={F -> FF-[-F+F]+[+F-F]},
               axiom=F, order=4,step=2pt,
               randomize step percent=50, angle=30,
               randomize angle percent=5}] l-system; }}

  \foreach \i in {0.5,0.6,...,2}
    \fill
      [white,opacity=\i/2,
       decoration=Koch snowflake,
       shift=(horizon),shift={(rand*11,rnd*7)},
       scale=\i,double copy shadow={
         opacity=0.2,shadow xshift=0pt,
         shadow yshift=3*\i pt,fill=white,draw=none}]
      decorate {
        decorate {
          decorate {
            (0,0)- ++(60:1) -- ++(-60:1) -- cycle
          } } };

  \node (left text) ...
  \node (right text) ...

  \fill [decorate,decoration={footprints,foot of=gnome},
         opacity=.5,brown]          (rand*8,-rnd*10)
    to [out=rand*180,in=rand*180] (rand*8,-rnd*10);
\end{tikzpicture}
```

Für meinen Vater, damit er noch viele schöne TEX-Graphiken erschaffen kann.

Till

TikZ 和 PGF 宏包

TikZ 3.0.1a 中文手册*

http://sourceforge.net/projects/pgf

Till Tantau†
Institut für Theoretische Informatik
Universität zu Lübeck

August 29, 2015 ‡

Contents

*翻译者：Hansimov。此项目开源在 GitHub，欢迎提出建议或参与翻译：https://github.com/Hansimov/pgfmanual-zh
†该手册的编者。手册的部分内容由另外一些作者写成，详见 1.5 小节。
‡中文手册更新时间：2018 年 7 月 3 日

1 引言

欢迎阅读 Ti*k*Z 和底层 PGF 系统的文档。一开始这只是一个小小的 LaTeX 样式，用来在我（Till Tantau）的博士论文里画图，如今它已经变成了飞速发展的图形语言，手册有一千多页。Ti*k*Z 的大量选项常常吓到新手，不过好消息是，这个文档还有一些慢节奏的教程，你不必看其他部分，就能学到几乎所有你需要知道的关于 Ti*k*Z 的内容。

我想从“什么是 Ti*k*Z？”这个问题开始。大体上，它只是定义一些 TeX 中的绘图命令。比如说，代码 `\tikz \draw (0pt,0pt) --(20pt,6pt);` 会生成一条线 ／，代码 `\tikz \fill[orange] (1ex,1ex) circle (1ex);` 会生成 ●。在某种程度上，当你用 Ti*k*Z 的时候，你是在**编写**图形**程序**，就像你用 TeX **编写**文档**程序**一样。这也解释了 Ti*k*Z 这个名字的由来：它是一个递归缩写，沿袭了“GNU is not unix”的风格。它的德文含义是“Ti*k*Z ist *kein* Zeichenprogramm”，翻译成英文就是“Ti*k*Z is not a drawing program”，中文意思是“Ti*k*Z 不是一个绘图程序”，它提醒读者注意这个宏包的真正意图。Ti*k*Z 采用“TeX 式的排版”，因此有这样一些优点：迅速创建简单图形，精准定位，可以使用宏，以及一流的排版。同样，它也继承了 TeX 系统的缺点：学习曲线陡峭，不是“所见即所得”[1]，微小的改变就需要长时间的重新编译，并且代码并不能真正“展示出”事物将有的样子。

现在我们知道什么是 Ti*k*Z 了，那么 PGF 呢？

1.1 The Layers Below Ti*k*Z

It turns out that there are actually *two* layers below Ti*k*Z:

System layer: This layer provides a complete abstraction of what is going on “in the driver.” The driver is a program like `dvips` or `dvipdfm` that takes a `.dvi` file as input and generates a `.ps` or a `.pdf` file. (The `pdftex` program also counts as a driver, even though it does not take a `.dvi` file as input. Never mind.) Each driver has its own syntax for the generation of graphics, causing headaches to everyone who wants to create graphics in a portable way. PGF's system layer “abstracts away” these differences. For example, the system command `\pgfsys@lineto{10pt}{10pt}` extends the current path to the coordinate $(10\text{pt}, 10\text{pt})$ of the current `{pgfpicture}`. Depending on whether `dvips`, `dvipdfm`, or `pdftex` is used to process the document, the system command will be converted to different `\special` commands. The system layer is as “minimalistic” as possible since each additional command makes it more work to port PGF to a new driver.

As a user, you will not use the system layer directly.

Basic layer: The basic layer provides a set of basic commands that allow you to produce complex graphics in a much easier manner than by using the system layer directly. For example, the system layer provides no commands for creating circles since circles can be composed from the more basic Bézier curves (well, almost). However, as a user you will want to have a simple command to create circles (at least I do) instead of having to write down half a page of Bézier curve support coordinates. Thus, the basic layer provides a command `\pgfpathcircle` that generates the necessary curve coordinates for you.

The basic layer consists of a *core*, which consists of several interdependent packages that can only be loaded *en bloc,* and additional *modules* that extend the core by more special-purpose commands like node management or a plotting interface. For instance, the BEAMER package uses only the core and not, say, the `shapes` modules.

In theory, Ti*k*Zitself is just one of several possible “frontends,” which are sets of commands or a special syntax that makes using the basic layer easier. A problem with directly using the basic layer is that code written for this layer is often too “verbose.” For example, to draw a simple triangle, you may need as many as five commands when using the basic layer: One for beginning a path at the first corner of the triangle, one for extending the path to the second corner, one for going to the third, one for closing the path, and one for actually painting the triangle (as opposed to filling it). With the Ti*k*Z frontend all this boils down to a single simple METAFONT-like command:

```
\draw (0,0) -- (1,0) -- (1,1) -- cycle;
```

[1] WYSIWYG，What You See Is What You Get。

In practice, Ti*k*Z is the only "serious" frontend for PGF. It gives you access to all features of PGF, but it is intended to be easy to use. The syntax is a mixture of METAFONT and PSTRICKS and some ideas of myself. There are other frontends besides Ti*k*Z, but they are more intended as "technology studies" and less as serious alternatives to Ti*k*Z. In particular, the `pgfpict2e` frontend reimplements the standard LaTeX `{picture}` environment and commands like `\line` or `\vector` using the PGF basic layer. This layer is not really "necessary" since the `pict2e.sty` package does at least as good a job at reimplementing the `{picture}` environment. Rather, the idea behind this package is to have a simple demonstration of how a frontend can be implemented.

Since most users will only use Ti*k*Z and almost no one will use the system layer directly, this manual is mainly about Ti*k*Z in the first parts; the basic layer and the system layer are explained at the end.

1.2 Comparison with Other Graphics Packages

Ti*k*Z is not the only graphics package for TeX. In the following, I try to give a reasonably fair comparison of Ti*k*Z and other packages.

1. The standard LaTeX `{picture}` environment allows you to create simple graphics, but little more. This is certainly not due to a lack of knowledge or imagination on the part of LaTeX's designer(s). Rather, this is the price paid for the `{picture}` environment's portability: It works together with all backend drivers.

2. The `pstricks` package is certainly powerful enough to create any conceivable kind of graphic, but it is not really portable. Most importantly, it does not work with `pdftex` nor with any other driver that produces anything but PostScript code.

   Compared to Ti*k*Z, `pstricks` has a similar support base. There are many nice extra packages for special purpose situations that have been contributed by users over the last decade. The Ti*k*Z syntax is more consistent than the `pstricks` syntax as Ti*k*Z was developed "in a more centralized manner" and also "with the shortcomings on `pstricks` in mind."

3. The `xypic` package is an older package for creating graphics. However, it is more difficult to use and to learn because the syntax and the documentation are a bit cryptic.

4. The `dratex` package is a small graphic package for creating a graphics. Compared to the other package, including Ti*k*Z, it is very small, which may or may not be an advantage.

5. The `metapost` program is a powerful alternative to Ti*k*Z. It used to be an external program, which entailed a bunch of problems, but in LuaTeX it is now build in. An obstacle with `metapost` is the inclusion of labels. This is *much* easier to achieve using PGF.

6. The `xfig` program is an important alternative to Ti*k*Z for users who do not wish to "program" their graphics as is necessary with Ti*k*Z and the other packages above. There is a conversion program that will convert `xfig` graphics to Ti*k*Z.

1.3 Utility Packages

The PGF package comes along with a number of utility package that are not really about creating graphics and which can be used independently of PGF. However, they are bundled with PGF, partly out of convenience, partly because their functionality is closely intertwined with PGF. These utility packages are:

1. The `pgfkeys` package defines a powerful key management facility. It can be used completely independently of PGF.

2. The `pgffor` package defines a useful `\foreach` statement.

3. The `pgfcalendar` package defines macros for creating calendars. Typically, these calendars will be rendered using PGF's graphic engine, but you can use `pgfcalendar` also typeset calendars using normal text. The package also defines commands for "working" with dates.

4. The `pgfpages` package is used to assemble several pages into a single page. It provides commands for assembling several "virtual pages" into a single "physical page." The idea is that whenever TeX has a page ready for "shipout," `pgfpages` interrupts this shipout and instead stores the page to be shipped out in a special box. When enough "virtual pages" have been accumulated in this way, they are scaled

down and arranged on a "physical page," which then *really* shipped out. This mechanism allows you to create "two page on one page" versions of a document directly inside LaTeX without the use of any external programs. However, `pgfpages` can do quite a lot more than that. You can use it to put logos and watermark on pages, print up to 16 pages on one page, add borders to pages, and more.

1.4 How to Read This Manual

This manual describes both the design of Ti*k*Z and its usage. The organization is very roughly according to "user-friendliness." The commands and subpackages that are easiest and most frequently used are described first, more low-level and esoteric features are discussed later.

If you have not yet installed Ti*k*Z, please read the installation first. Second, it might be a good idea to read the tutorial. Finally, you might wish to skim through the description of Ti*k*Z. Typically, you will not need to read the sections on the basic layer. You will only need to read the part on the system layer if you intend to write your own frontend or if you wish to port PGF to a new driver.

The "public" commands and environments provided by the system are described throughout the text. 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.

1.5 Authors and Acknowledgements

The bulk of the PGF system and its documentation was written by Till Tantau. A further member of the main team is Mark Wibrow, who is responsible, for example, for the PGF mathematical engine, many shapes, the decoration engine, and matrices. The third member is Christian Feuersänger who contributed the floating point library, image externalization, extended key processing, and automatic hyperlinks in the manual.

Furthermore, occasional contributions have been made by Christophe Jorssen, Jin-Hwan Cho, Olivier Binda, Matthias Schulz, Renée Ahrens, Stephan Schuster, and Thomas Neumann.

Additionally, numerous people have contributed to the PGF system by writing emails, spotting bugs, or sending libraries and patches. Many thanks to all these people, who are too numerous to name them all!

1.6 Getting Help

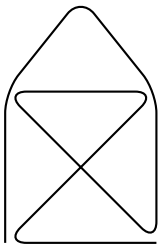
When you need help with PGF and Ti*k*Z, please do the following:

1. Read the manual, at least the part that has to do with your problem.
2. If that does not solve the problem, try having a look at the sourceforge development page for PGF and Ti*k*Z (see the title of this document). Perhaps someone has already reported a similar problem and someone has found a solution.
3. On the website you will find numerous forums for getting help. There, you can write to help forums, file bug reports, join mailing lists, and so on.
4. Before you file a bug report, especially a bug report concerning the installation, make sure that this is really a bug. In particular, have a look at the `.log` file that results when you TeX your files. This `.log` file should show that all the right files are loaded from the right directories. Nearly all installation problems can be resolved by looking at the `.log` file.
5. *As a last resort* you can try to email me (Till Tantau) or, if the problem concerns the mathematical engine, Mark Wibrow. I do not mind getting emails, I simply get way too many of them. Because of this, I cannot guarantee that your emails will be answered timely or even at all. Your chances that your problem will be fixed are somewhat higher if you mail to the PGF mailing list (naturally, I read this list and answer questions when I have the time).

Part I
教程和指导

by Till Tantau

为了帮你入门 Ti*k*Z，本手册没有立刻给出长长的安装和配置过程，而是直接从教程开始。这些教程解释了该系统所有基本特性和部分高级特性，并不深入所有细节。这部分还指导你在用 Ti*k*Z 绘图时，如何继续前进。

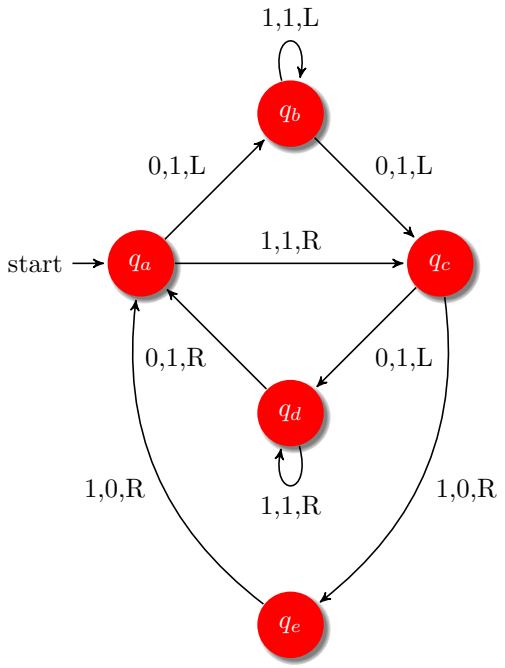


```
\tikz \draw[thick,rounded corners=8pt]
  (0,0) -- (0,2) -- (1,3.25) -- (2,2) -- (2,0) -- (0,2) -- (2,2) -- (0,0) -- (2,0);
```

Part II
安装和配置

by Till Tantau

这部分介绍如何安装该系统。通常已经有人帮你装好了，所以你可以跳过这部分；但是如果事与愿违，你是那个不得不自己安装的可怜的家伙，那么请阅读这一部分。



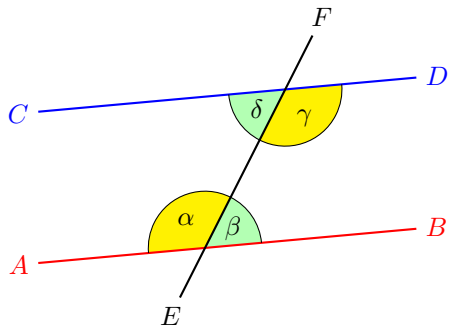
The current candidate for the busy beaver for five states. It is presumed that this Turing machine writes a maximum number of 1's before halting among all Turing machines with five states and the tape alphabet $\{0, 1\}$. Proving this conjecture is an open research problem. 中文测试

```
\begin{tikzpicture}[->,>=stealth',shorten >=1pt,auto,node distance=2.8cm,on grid,semithick,
                    every state/.style={fill=red,draw=none,circular drop shadow,text=white}]

  \node[initial,state] (A)                    {$q_a$};
  \node[state]         (B) [above right=of A] {$q_b$};
  \node[state]         (D) [below right=of A] {$q_d$};
  \node[state]         (C) [below right=of B] {$q_c$};
  \node[state]         (E) [below=of D]       {$q_e$};

  \path (A) edge              node {0,1,L} (B)
            edge              node {1,1,R} (C)
        (B) edge [loop above] node {1,1,L} (B)
            edge              node {0,1,L} (C)
        (C) edge              node {0,1,L} (D)
            edge [bend left]  node {1,0,R} (E)
        (D) edge [loop below] node {1,1,R} (D)
            edge              node {0,1,R} (A)
        (E) edge [bend left]  node {1,0,R} (A);

   \node [right=1cm,text width=8cm] at (C)
   {
     The current candidate for the busy beaver for five states. It is
     presumed that this Turing machine writes a maximum number of
     $1$'s before halting among all Turing machines with five states
     and the tape alphabet $\{0, 1\}$. Proving this conjecture is an
     open research problem. 中文测试
   };
\end{tikzpicture}
```

Part III

TikZ ist *kein* Zeichenprogramm

by Till Tantau



When we assume that AB and CD are parallel, i. e., $AB \parallel CD$, then $\alpha = \delta$ and $\beta = \gamma$.

```
\begin{tikzpicture}[angle radius=.75cm]

  \node (A) at (-2,0)    [red,left]   {$A$};
  \node (B) at ( 3,.5)    [red,right]  {$B$};
  \node (C) at (-2,2)  [blue,left]  {$C$};
  \node (D) at ( 3,2.5)  [blue,right] {$D$};
  \node (E) at (60:-5mm) [below]      {$E$};
  \node (F) at (60:3.5cm)  [above]      {$F$};

  \coordinate (X) at (intersection cs:first line={(A)--(B)}, second line={(E)--(F)});
  \coordinate (Y) at (intersection cs:first line={(C)--(D)}, second line={(E)--(F)});

  \path
    (A) edge [red, thick]  (B)
    (C) edge [blue, thick] (D)
    (E) edge [thick]       (F)
      pic ["$\alpha$",  draw, fill=yellow]   {angle = F--X--A}
      pic ["$\beta$",   draw, fill=green!30] {angle = B--X--F}
      pic ["$\gamma$",  draw, fill=yellow]   {angle = E--Y--D}
      pic ["$\delta$",  draw, fill=green!30] {angle = C--Y--E};

  \node at ($ (D)!.5!(B) $) [right=1cm,text width=6cm,rounded corners,fill=red!20,inner sep=1ex]
    {
      When we assume that $\color{red}AB$ and $\color{blue}CD$ are
      parallel, i.\,e., ${\color{red}AB} \mathbin{\|} \color{blue}CD$,
      then $\alpha = \delta$ and $\beta = \gamma$.
    };
\end{tikzpicture}
```

Part IV
Graph Drawing

by Till Tantau et al.

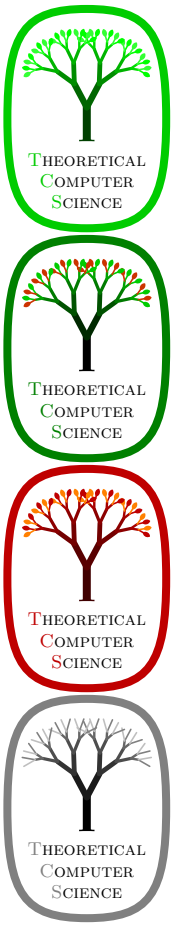
Graph drawing algorithms do the tough work of computing a layout of a graph for you. Ti*k*Z comes with powerful such algorithms, but you can also implement new algorithms in the Lua programming language.

You need to use LuaTEX to typeset this part of the manual (and, also, to use algorithmic graph drawing).

Part V
Libraries

by Till Tantau

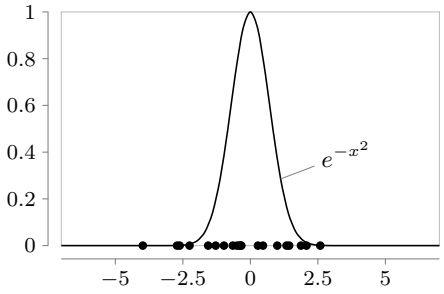
In this part the library packages are documented. They provide additional predefined graphic objects like new arrow heads or new plot marks, but sometimes also extensions of the basic PGF or Ti*k*Z system. The libraries are not loaded by default since many users will not need them.




```
\tikzset{
  ld/.style={level distance=#1},lw/.style={line width=#1},
  level 1/.style={ld=4.5mm, trunk,          lw=1ex ,sibling angle=60},
  level 2/.style={ld=3.5mm, trunk!80!leaf a,lw=.8ex,sibling angle=56},
  level 3/.style={ld=2.75mm,trunk!60!leaf a,lw=.6ex,sibling angle=52},
  level 4/.style={ld=2mm,   trunk!40!leaf a,lw=.4ex,sibling angle=48},
  level 5/.style={ld=1mm,   trunk!20!leaf a,lw=.3ex,sibling angle=44},
  level 6/.style={ld=1.75mm,leaf a,         lw=.2ex,sibling angle=40},
}
\pgfarrowsdeclare{leaf}{leaf}
  {\pgfarrowsleftextend{-2pt} \pgfarrowsrightextend{1pt}}
{
  \pgfpathmoveto{\pgfpoint{-2pt}{0pt}}
  \pgfpatharc{150}{30}{1.8pt}
  \pgfpatharc{-30}{-150}{1.8pt}
  \pgfusepathqfill
}

\newcommand{\logo}[5]
{
  \colorlet{border}{#1}
  \colorlet{trunk}{#2}
  \colorlet{leaf a}{#3}
  \colorlet{leaf b}{#4}
  \begin{tikzpicture}
    \scriptsize\scshape
    \draw[border,line width=1ex,yshift=.3cm,
          yscale=1.45,xscale=1.05,looseness=1.42]
      (1,0) to [out=90, in=0]    (0,1)  to [out=180,in=90]  (-1,0)
            to [out=-90,in=-180] (0,-1) to [out=0,  in=-90] (1,0) -- cycle;

    \coordinate (root) [grow cyclic,rotate=90]
    child {
      child [line cap=round] foreach \a in {0,1} {
        child foreach \b in {0,1} {
          child foreach \c in {0,1} {
            child foreach \d in {0,1} {
              child foreach \leafcolor in {leaf a,leaf b}
                { edge from parent [color=\leafcolor,-#5] }
        } } }
      } edge from parent [shorten >=-1pt,serif cm-,line cap=butt]
    };

    \node [align=center,below] at (0pt,-.5ex)
    { \textcolor{border}{T}heoretical \\ \textcolor{border}{C}omputer \\
      \textcolor{border}{S}cience };
  \end{tikzpicture}
}
\begin{minipage}{3cm}
  \logo{green!80!black}{green!25!black}{green}{green!80}{leaf}\\
  \logo{green!50!black}{black}{green!80!black}{red!80!green}{leaf}\\
  \logo{red!75!black}{red!25!black}{red!75!black}{orange}{leaf}\\
  \logo{black!50}{black}{black!50}{black!25}{}
\end{minipage}
```

Part VI
Data Visualization

by Till Tantau



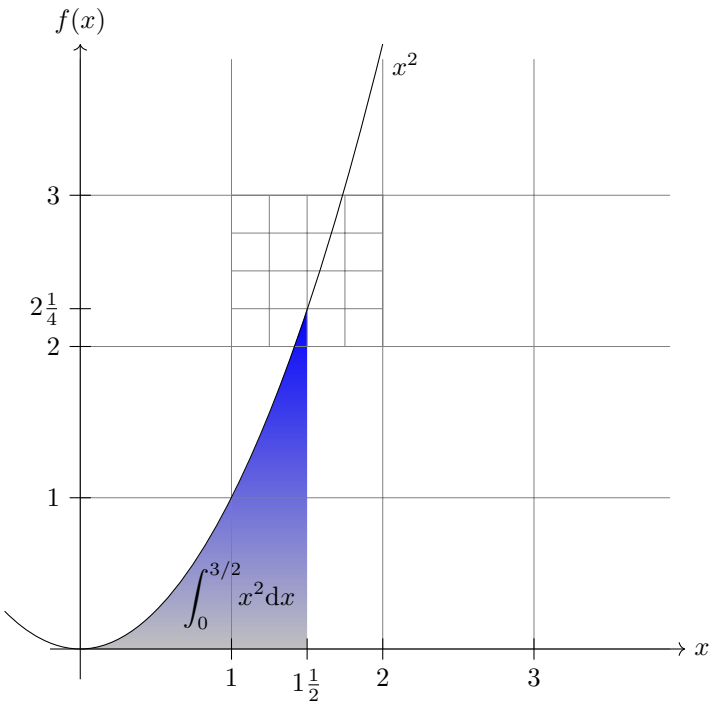
- $\sum_{i=1}^{10} x_i$, where $x_i \sim U(-1, 1)$

```
\tikz \datavisualization [scientific axes=clean]
[
  visualize as smooth line=Gaussian,
  Gaussian={pin in data={text={$e^{-x^2}$},when=x is 1}}
]
data [format=function] {
  var x : interval [-7:7] samples 51;
  func y = exp(-\value x*\value x);
}
[
  visualize as scatter,
  legend={south east outside},
  scatter={
    style={mark=*,mark size=1.4pt},
    label in legend={text={
        $\sum_{i=1}^{10} x_i$, where $x_i \sim U(-1,1) $}}}
]
data [format=function] {
  var i : interval [0:1] samples 20;
  func y = 0;
  func x = (rand + rand + rand + rand + rand +
            rand + rand + rand + rand + rand);
};
```

Part VII
Utilities

by Till Tantau

The utility packages are not directly involved in creating graphics, but you may find them useful nonetheless. All of them either directly depend on PGF or they are designed to work well together with PGF even though they can be used in a stand-alone way.




```
\begin{tikzpicture}[scale=2]
  \shade[top color=blue,bottom color=gray!50] (0,0) parabola (1.5,2.25) |- (0,0);
  \draw (1.05cm,2pt) node[above] {$\displaystyle\int_0^{3/2} \!\!x^2\mathrm{d}x$};

  \draw[help lines] (0,0) grid (3.9,3.9)
       [step=0.25cm]      (1,2) grid +(1,1);

  \draw[->] (-0.2,0) -- (4,0) node[right] {$x$};
  \draw[->] (0,-0.2) -- (0,4) node[above] {$f(x)$};

  \foreach \x/\xtext in {1/1, 1.5/1\frac{1}{2}, 2/2, 3/3}
    \draw[shift={(\x,0)}] (0pt,2pt) -- (0pt,-2pt) node[below] {$\xtext$};

  \foreach \y/\ytext in {1/1, 2/2, 2.25/2\frac{1}{4}, 3/3}
    \draw[shift={(0,\y)}] (2pt,0pt) -- (-2pt,0pt) node[left] {$\ytext$};

  \draw (-.5,.25) parabola bend (0,0) (2,4) node[below right] {$x^2$};
\end{tikzpicture}
```

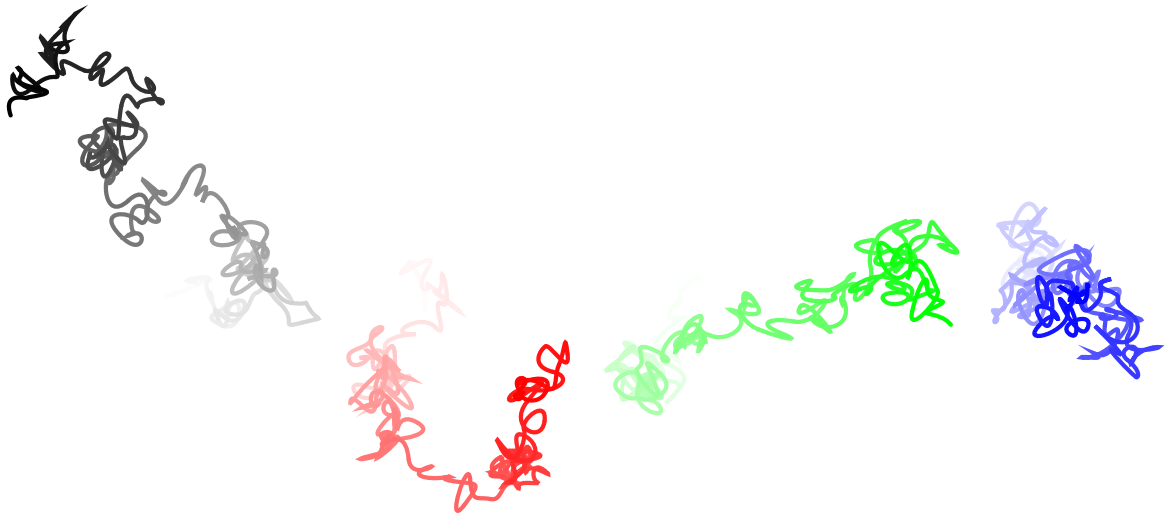
Part VIII
Mathematical and Object-Oriented Engines

by Mark Wibrow and Till Tantau

PGF comes with two useful engines: One for doing mathematics, one for doing object-oriented programming. Both engines can be used independently of the main PGF.

The job of the mathematical engine is to support mathematical operations like addition, subtraction, multiplication and division, using both integers and non-integers, but also functions such as square-roots, sine, cosine, and generate pseudo-random numbers. Mostly, you will use the mathematical facilities of PGF indirectly, namely when you write a coordinate like `(5cm*3,6cm/4)`, but the mathematical engine can also be used independently of PGF and Ti*k*Z.

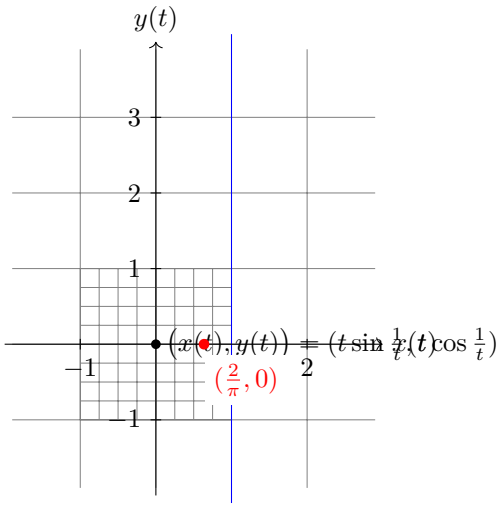
The job of the object-oriented engine is to support simple object-oriented programming in TEX. It allows the definition of *classes* (without inheritance), *methods*, *attributes* and *objects*.



```
\pgfmathsetseed{1}
\foreach \col in {black,red,green,blue}
{
  \begin{tikzpicture}[x=10pt,y=10pt,ultra thick,baseline,line cap=round]
    \coordinate (current point) at (0,0);
    \coordinate (old velocity) at (0,0);
    \coordinate (new velocity) at (rand,rand);

    \foreach \i in {0,1,...,100}
    {
      \draw[\col!\i] (current point)
      .. controls ++([scale=-1]old velocity) and
                  ++(new velocity) .. ++(rand,rand)
         coordinate (current point);
      \coordinate (old velocity) at (new velocity);
      \coordinate (new velocity) at (rand,rand);
    }
  \end{tikzpicture}
}
```

Part IX
The Basic Layer

by Till Tantau




```
\begin{tikzpicture}
  \draw[gray,very thin] (-1.9,-1.9) grid (2.9,3.9)
          [step=0.25cm] (-1,-1) grid (1,1);
  \draw[blue] (1,-2.1) -- (1,4.1); % asymptote

  \draw[->] (-2,0) -- (3,0) node[right] {$x(t)$};
  \draw[->] (0,-2) -- (0,4) node[above] {$y(t)$};

  \foreach \pos in {-1,2}
    \draw[shift={(\pos,0)}] (0pt,2pt) -- (0pt,-2pt) node[below] {$\pos$};

  \foreach \pos in {-1,1,2,3}
    \draw[shift={(0,\pos)}] (2pt,0pt) -- (-2pt,0pt) node[left] {$\pos$};

  \fill (0,0) circle (0.064cm);
  \draw[thick,parametric,domain=0.4:1.5,samples=200]
    % The plot is reparameterised such that there are more samples
    % near the center.
    plot[id=asymptotic-example] function{(t*t*t)*sin(1/(t*t*t)),(t*t*t)*cos(1/(t*t*t))}
    node[right] {$\bigl(x(t),y(t)\bigr) = (t\sin \frac{1}{t}, t\cos \frac{1}{t})$};

  \fill[red] (0.63662,0) circle (2pt)
    node [below right,fill=white,yshift=-4pt] {$(\frac{2}{\pi},0)$};
\end{tikzpicture}
```

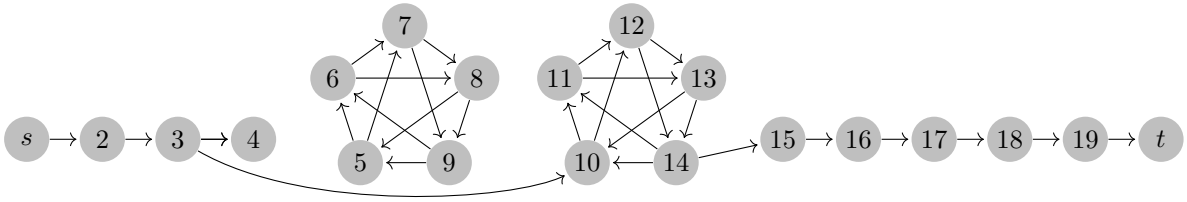
Part X
The System Layer

by Till Tantau

This part describes the low-level interface of PGF, called the *system layer*. This interface provides a complete abstraction of the internals of the underlying drivers.

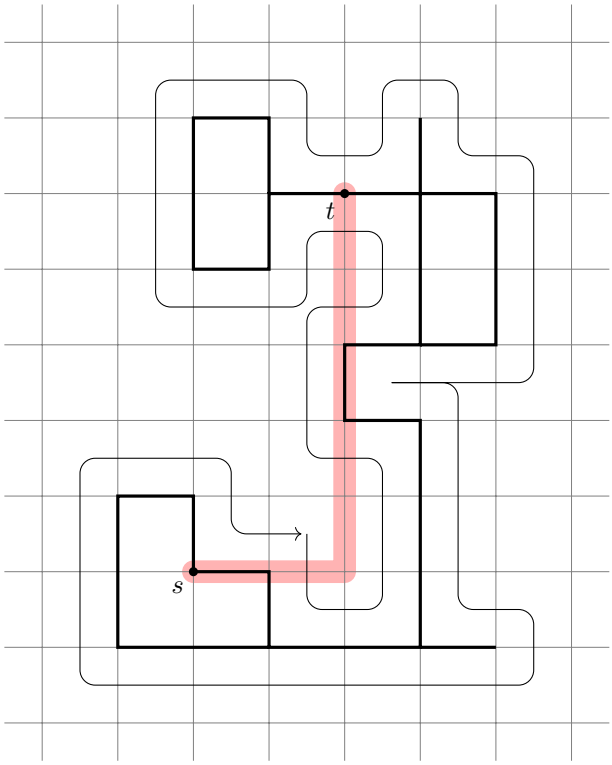
Unless you intend to port PGF to another driver or unless you intend to write your own optimized frontend, you need not read this part.

In the following it is assumed that you are familiar with the basic workings of the `graphics` package and that you know what TEX-drivers are and how they work.




```
\begin{tikzpicture}
  [shorten >=1pt,->,
   vertex/.style={circle,fill=black!25,minimum size=17pt,inner sep=0pt}]

  \foreach \name/\x in {s/1, 2/2, 3/3, 4/4, 15/11, 16/12, 17/13, 18/14, 19/15, t/16}
    \node[vertex] (G-\name) at (\x,0) {$\name$};

  \foreach \name/\angle/\text in {P-1/234/5, P-2/162/6, P-3/90/7, P-4/18/8, P-5/-54/9}
    \node[vertex,xshift=6cm,yshift=.5cm] (\name) at (\angle:1cm) {$\text$};

  \foreach \name/\angle/\text in {Q-1/234/10, Q-2/162/11, Q-3/90/12, Q-4/18/13, Q-5/-54/14}
    \node[vertex,xshift=9cm,yshift=.5cm] (\name) at (\angle:1cm) {$\text$};

  \foreach \from/\to in {s/2,2/3,3/4,3/4,15/16,16/17,17/18,18/19,19/t}
    \draw (G-\from) -- (G-\to);

  \foreach \from/\to in {1/2,2/3,3/4,4/5,5/1,1/3,2/4,3/5,4/1,5/2}
    { \draw (P-\from) -- (P-\to); \draw (Q-\from) -- (Q-\to); }

  \draw (G-3) .. controls +(-30:2cm) and +(-150:1cm) .. (Q-1);
  \draw (Q-5) -- (G-15);
\end{tikzpicture}
```

Part XI
References and Index




```
\begin{tikzpicture}
  \draw[line width=0.3cm,color=red!30,line cap=round,line join=round] (0,0)--(2,0)--(2,5);
  \draw[help lines] (-2.5,-2.5) grid (5.5,7.5);
  \draw[very thick] (1,-1)--(-1,-1)--(-1,1)--(0,1)--(0,0)--
    (1,0)--(1,-1)--(3,-1)--(3,2)--(2,2)--(2,3)--(3,3)--
    (3,5)--(1,5)--(1,4)--(0,4)--(0,6)--(1,6)--(1,5)
    (3,3)--(4,3)--(4,5)--(3,5)--(3,6)
    (3,-1)--(4,-1);
  \draw[below left] (0,0) node(s){$s$};
  \draw[below left] (2,5) node(t){$t$};
  \fill (0,0) circle (0.06cm) (2,5) circle (0.06cm);
  \draw[->,rounded corners=0.2cm,shorten >=2pt]
    (1.5,0.5)-- ++(0,-1)-- ++(1,0)-- ++(0,2)-- ++(-1,0)-- ++(0,2)-- ++(1,0)--
    ++(0,1)-- ++(-1,0)-- ++(0,-1)-- ++(-2,0)-- ++(0,3)-- ++(2,0)-- ++(0,-1)--
    ++(1,0)-- ++(0,1)-- ++(1,0)-- ++(0,-1)-- ++(1,0)-- ++(0,-3)-- ++(-2,0)--
    ++(1,0)-- ++(0,-3)-- ++(1,0)-- ++(0,-1)-- ++(-6,0)-- ++(0,3)-- ++(2,0)--
    ++(0,-1)-- ++(1,0);
\end{tikzpicture}
```