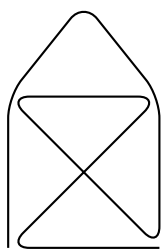


## Part I

# 教程和指导

*by Till Tantau*

为了帮你入门 TikZ，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 TikZ 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);
```

# 1 教程：给 Karl 的学生的一张图

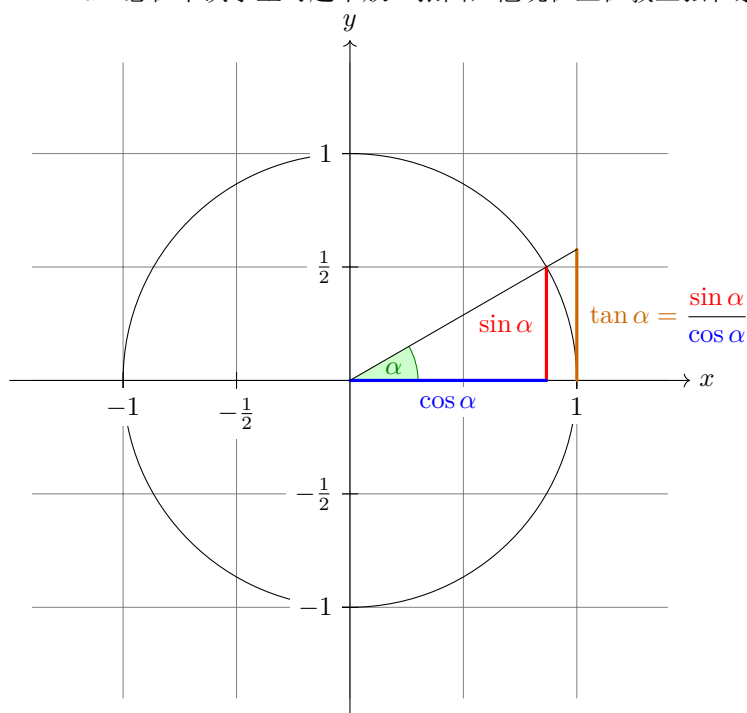
该教程是写给 TikZ 的新手的，它并不详细罗列 TikZ 的所有特性，而是只讲那些你可能立马会用到的。

Karl 是一名高中数学和化学老师，他以前用 L<sup>A</sup>T<sub>E</sub>X 的 `{picture}` 环境，为习题和考试画图。尽管结果可以接受，但是绘图通常要花很久，并且还有一些问题，比如线之间有微小的角度错误，圆形好像也很难画对。当然，他的学生并不关心线之间的角度对不对，因为 Karl 出的试题太难了，画得再好也没用。但 Karl 对绘图的结果从不满意。

Karl 的儿子对绘图结果更不满意（毕竟他不要考试），儿子告诉 Karl 他也许愿意试试一个新的宏包来绘图。令人困惑的是，这个宏包似乎有两个名字：首先 Karl 需要下载和安装一个叫 PGF 的宏包，接着他发现这个宏包里还有另一个宏包，叫 TikZ，应该代表“TikZ ist *kein* Zeichenprogramm”（TikZ 不是一个画图程序）。Karl 觉得有点奇怪，从 TikZ 的名字来看，似乎并不能满足自己的要求。不过，由于他用过一段时间的 GNU 软件，并且知道“GNU's Not Unix!”（GNU 不是 Unix）这句典故，因此似乎有点希望。他儿子告诉他，TikZ 起这个名字，是为了提醒人们，TikZ 不是一个用鼠标和平板画图的程序，相反，它更像是一门“图形语言”。

## 1.1 问题陈述

Karl 想在下次学生习题中放一张图，他现在正在教正弦和余弦。他想得到这样的图（理想情况）：



在本例中，角  $\alpha$  为  $30^\circ$ （等于  $\pi/6$  弧度）， $\alpha$  的正弦，也就是红色线段的高度，为

$$\sin \alpha = 1/2.$$

根据勾股定理，我们有  $\cos^2 \alpha + \sin^2 \alpha = 1$ 。因此蓝色线段的长度，也就是  $\alpha$  的余弦，肯定为

$$\cos \alpha = \sqrt{1 - 1/4} = \sqrt{3}/2.$$

由此可以得到  $\tan \alpha$ ，也就是橙色线段的高度，为

$$\tan \alpha = \frac{\sin \alpha}{\cos \alpha} = 1/\sqrt{3}.$$

## 1.2 设置环境

要在 TikZ 中画一张图片，你需要在图片开头告诉 T<sub>E</sub>X 或者 L<sup>A</sup>T<sub>E</sub>X 你想开始画一张图。在 L<sup>A</sup>T<sub>E</sub>X 中需要用 `{tikzpicture}` 环境，在 plain T<sub>E</sub>X 中你只要用 `\tikzpicture` 开始图片，再用 `\endtikzpicture` 结束图片。

### 1.2.1 设置 L<sup>A</sup>T<sub>E</sub>X 中的环境

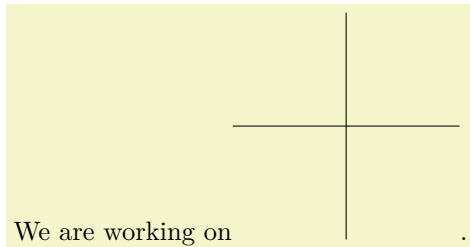
Karl 是一个 L<sup>A</sup>T<sub>E</sub>X 用户，所以他的设置是这样的：

```

\documentclass{article} % say
\usepackage{tikz}
\begin{document}
We are working on
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}.
\end{document}

```

当程序执行时，也就是用 `pdflatex`，或者是 `latex` 加上 `dvips`，输出包含这样的内容：



```

We are working on
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\end{tikzpicture}.

```

诚然，图还算不上完整，不过我们已经建好坐标轴，还画好组成坐标轴的线了。Karl 突然有点沮丧，似乎离画完还有不少距离。

让我们更仔细地看一下代码。首先导入 `tikz` 库，也就是基础 PGF 系统所谓的“前端”。本手册还会提到基本层，更基础也更难用。这个前端语法更简单，让画图变得更容易了。

里面有两句 `\draw` 命令，意思是：“从此处到逗号间指定的路径，需要画出来。”第一条路径指定为 `(-1.5,0) -- (0,1.5)`，意思是：“一条从点  $(-1.5, 0)$  到点  $(-1.5, 0)$  的线”。在这里，位置是用一个特殊的坐标系统指定的，初始的单位长度是 1cm。

Karl 很高兴地注意到，环境中已经预留了足够的空间来容纳图片。

### 1.2.2 设置 Plain T<sub>E</sub>X 中的环境

Karl 的妻子 Gerda，恰好也是一名数学老师，她不是 L<sup>A</sup>T<sub>E</sub>X 用户，而是用 plain T<sub>E</sub>X，因为她更喜欢“老派的方法”。她也能用 TikZ，不过不是 `\usepackage{tikz}`，而是 `\input tikz.tex`，同样，`\begin{tikzpicture}` 和 `\end{tikzpicture}` 也应该换成 `\tikzpicture` 和 `\endtikzpicture`。

因此她应该用下面的语句：

```

%% Plain TeX file
\input tikz.tex
\baselineskip=12pt
\hsize=6.3truein
\vsize=8.7truein
We are working on
\tikzpicture
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
\endtikzpicture.
\bye

```

Gerda 既能用 `pdftex` 也能用 `tex` 加 `dvips` 来排版这个文件。TikZ 会自动识别她用了哪个驱动。如果她想用 `dvipdfm` 结合 `tex`，她既可以选择修改 `pfg.cfg` 文件，也可以选择在导入 `tikz.tex` 或是 `pgf.tex` 之前加上 `\def\pgfsysdriver{pgfsys-dvipdfm.def}`。

### 1.2.3 设置 ConT<sub>E</sub>Xt 中的环境

Karl 的叔叔 Hans 用 ConT<sub>E</sub>Xt，Hans 也能像 Gerda 一样使用 TikZ。他需要把 `\usepackage{tikz}` 换成 `\usemodule{tikz}`，同样，`\begin{tikzpicture}` 和 `\end{tikzpicture}` 也要换成 `\starttikzpicture` 和 `\stoptikzpicture`。

他的样例代码像这样：

```

%% ConTeXt file
\usemodule[tikz]

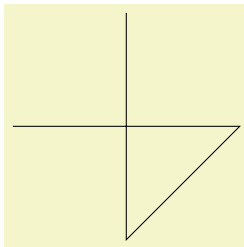
\starttext
  We are working on
  \starttikzpicture
    \draw (-1.5,0) -- (1.5,0);
    \draw (0,-1.5) -- (0,1.5);
  \stoptikzpicture.
\stoptext

```

Hans 现在就能像平常一样，用 `texexec` 或者 `context` 来排版这个文件了。

### 1.3 直线路径创建

TikZ 里所有图片的基本单元是路径 (path)。路径就是相连的直线和曲线 (并不是整个图片都是如此，不过让我们暂时忽略复杂的部分)。你指定一个起点作为路径开始，圆括号中为点的坐标，比如  $(0,0)$ 。紧接着是一系列的“路径扩展操作”，最简单的是 `--`，我们之前用过，它后面必须跟着另一个坐标，它将原来的路径沿直线延伸到新的位置。比如，如果我们打算把坐标轴的两条路径并成一条，那么会得到下面的结果：



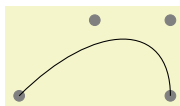
```
\tikz \draw (-1.5,0) -- (1.5,0) -- (0,-1.5) -- (0,1.5);
```

Karl 有点困惑，因为这里没见到 `{tikzpicture}` 环境，而是用了一个小命令 `\tikz`。这个命令既可以接受单个参数 (放在花括号中，比如 `\tikz{\draw (0,0) -- (1.5,0)}` 会生成一条线)，也可以将其放到 `{tikzpicture}` 环境中，每句用以分号结尾。一般来说，所有的 TikZ 绘图命令必须作为 `\tikz` 的参数，或者放在 `{tikzpicture}` 环境里。幸运的是，`\draw` 命令只定义在这种环境下，因此你很少会有出错的机会。

### 1.4 曲线路径创建

Karl 下一步想做的就是画圆，很明显用直线做不到，所以我们需要一些方法绘制曲线。为此，TikZ 提供了一个特殊的语法，需要用一到两个“控制点”。它背后的数学并不简单，但是基本思想是：假设你位于点  $x$ ，第一个控制点是  $y$ ，那么曲线开始时沿着从  $x$  到  $y$  的方向，也就是说，该曲线在点  $x$  处的切线经过  $y$ 。接着，假设曲线的结束点为  $z$ ，第二个控制点是  $w$ ，那么该曲线就会在点  $z$  处结束，并且曲线在点  $z$  处的切线经过  $w$ 。

例子如下 (为了更清楚地说明，这里画上了控制点)：



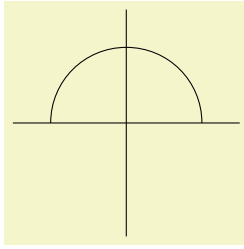
```

\begin{tikzpicture}
  \filldraw [gray] (0,0) circle [radius=2pt]
    (1,1) circle [radius=2pt]
    (2,1) circle [radius=2pt]
    (2,0) circle [radius=2pt];
  \draw (0,0) .. controls (1,1) and (2,1) .. (2,0);
\end{tikzpicture}

```

以曲线方式延伸路径的通用语法是 `.. controls <第一个控制点> and <第二个控制点> .. <结束点>`。你可以略掉 `and <第二个控制点>`，此时第二个控制点和第一个相同。

所以，Karl 现在可以把一个半圆加进图片了：



```
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (-1,0) .. controls (-1,0.555) and (-0.555,1) .. (0,1)
.. controls (0.555,1) and (1,0.555) .. (1,0);
\end{tikzpicture}
```

Karl 看到结果很开心，但是觉得用这种方法指定圆太笨拙了，不过还好有个简单得多的方法。

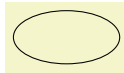
## 1.5 圆形路径创建

路径创建操作 `circle` 可以用来画圆，后面跟着半径，写在方括号里，例子如下：（注意到前面的点作为圆心。）



```
\tikz \draw (0,0) circle [radius=10pt];
```

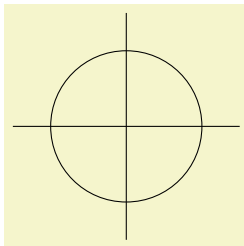
你还可以用 `ellipse` 画椭圆，它有两个半径：



```
\tikz \draw (0,0) ellipse [x radius=20pt, y radius=10pt];
```

要画一个轴线倾斜成任意角度而不是横平竖直的椭圆，比如一个“旋转的椭圆” $\circ$ ，你可以加上变换，这个后面会解释。顺带一提，这个小椭圆的代码是：`\tikz \draw[rotate=30] (0,0) ellipse [x radius=6pt, y radius=3pt];`。

所以，回到 Karl 的问题，他可以用 `\draw (0,0) circle [radius=1cm];` 来画圆：

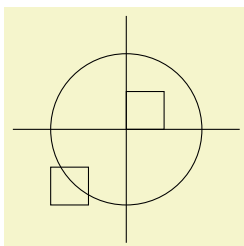


```
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

此时 Karl 有点担心，因为这个圆太小了，而他最后想要的图片要大得多。他很高兴地了解到，TikZ 有强大的变换选项，所以把每样东西都放大三倍非常容易。不过我们暂时先保留这个尺寸，省点地方。

## 1.6 矩形路径创建

我们下一步想画背景中的网格，有好几种方法可以实现。比如你可以画许多矩形。因为矩形很常用，所以有个专门的语法：可以用 `rectangle` 在当前路径中画一个矩形。这个操作后面需要跟另外一个坐标，这样前后两个坐标就构成了矩形对角的两个点，并将矩形绘制出来。所以让我们在图片中加两个矩形：




```
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\draw (0,0) rectangle (0.5,0.5);
\draw (-0.5,-0.5) rectangle (-1,-1);
\end{tikzpicture}
```

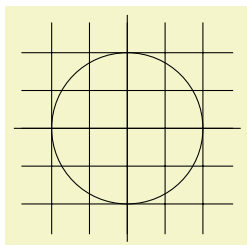
也许在其他情况下这个方法还不错，但是并没有真的解决 Karl 的问题：首先我们需要很多很多矩形，并且边界处是不“闭合”的。

所以，正当 Karl 准备用 `\draw` 命令画四条横线、四条竖线时，他得知有一个 `grid`（网格）路径创建操作。

## 1.7 网格路径创建

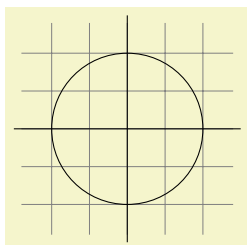
`grid` 路径操作在当前路径加上网格。它会画出组成网格的线，`grid` 前后两个点坐标构成了网格矩形的两个对角。比如，代码 `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 生成网格 。注意到 `\draw` 后面的选项，可以指定网格的宽度（还可以用 `xstep` 和 `ystep` 来分别设置网格间隔）。正如 Karl 很快会学到的，用这些选项可以影响非常多的事物。

Karl 可以用下面的代码：



```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw[step=.5cm] (-1.4,-1.4) grid (1.4,1.4);
\end{tikzpicture}
```

Karl 又看了一眼理想中的图片，他注意到，要是网格更柔和一点就好了。（他儿子告诉他，如果网格不柔和，会让人分心。）为了让网格柔和，他又在 `\draw` 后面加了两个选项。首先他把网格线的颜色设为 `gray`，然后将网格的线宽减到了 `very thin`，最后，他调换了命令的顺序，先画网格，再在上面画其他的。



```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

## 1.8 添加一点样式

除了用 `gray,very thin`，Karl 还可以用 `help lines`。样式就是预定义的选项集合，用来组织图形的绘制。`help lines` 的意思是“使用我（或者其他人）已经设置好的绘制帮助线的样式”。如果 Karl 之后打算用颜色 `blue!50` 而不是 `gray` 画网格，他可以用下面的选项：

```
help lines/.style={color=blue!50,very thin}
```

这个“样式设置”的作用是，在当前作用域或环境下，`help lines` 的效果等同于 `color=blue!50,very thin`。

样式让图形代码更灵活，你可以用统一的方法更容易地修改图形的外观。通常样式在图片开头定义，不过你有时候可能想定义一个全局样式，这样所有图片就可以用这个样式了，然后只改这个样式就能改所有图片的样式。要想这样，你可以在文档开头用 `tikzset` 命令：

```
\tikzset{help lines/.style=very thin}
```

你可以在一个样式中使用另一个样式，建立层级关系。所以要在 `grid` 的基础上定义一个 `Karl's grid` 样式，可以这样写：

```
\tikzset{Karl's grid/.style={help lines,color=blue!50}}
...
\draw[Karl's grid] (0,0) grid (5,5);
```

参数化可以让样式更强大，也就是说，像其他选项一样，样式也是能使用参数的。比如，Karl 可以将他的网格样式参数化，默认蓝色，但也可以用另外一种颜色。

```

\begin{tikzpicture}
[Karl's grid/.style={help lines,color=#1!50},
Karl's grid/.default=blue]

\draw[Karl's grid] (0,0) grid (1.5,2);
\draw[Karl's grid=red] (2,0) grid (3.5,2);
\end{tikzpicture}

```

## 1.9 画图选项

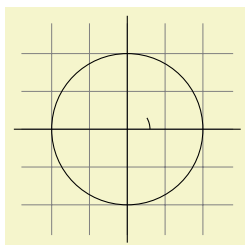
Karl 在想还有什么命令能够影响路径的绘制。他已经注意到 `color=<color>` 可以设置线的颜色，而 `draw=<color>` 的作用几乎一样，只不过它只设置线的轮廓颜色，而填充颜色可以设置成另外一种（后面要填充弧线时，Karl 会用到这个）。

Karl 还注意到 `very thin` 生成了很细的线，对此他并不奇怪，毕竟 `thin` 生成细线，`thick` 生成粗线，`very thick` 生成很粗的线，`ultra thick` 生成超粗的线，`ultra thin` 生成超细的线，细到低分辨率的打印机和显示器都很难显示。他想知道的是，线的“正常”粗细是多少。结果表明 `thin` 是正确的选项，因为它和 TeX 的 `\hrule` 命令生成的线粗细一样。尽管如此，Karl 还是想知道，是否有选项在 `thin` 和 `thick` 中间。事实上是有的，它就是 `semithick`。

另一个有用的线样式是虚线和点线，可以用 `dashed` 生成 `----`，用 `dotted` 生成 `.....`。这两类线都有稀疏和紧密的版本，分别叫 `loosely dashed`，`densely dashed`，`loosely dotted` 和 `densely dotted`。如果 Karl 真需要的话，他还可以用 `dash pattern` 设置更复杂的虚线图案。不过他儿子说，虚线需要尽可能小心使用，并且最容易让人分心，复杂的虚线图案非常糟糕，毕竟 Karl 的学生并不关心虚线的图案。

## 1.10 弧线路径创建

我们的下一个问题是绘制角的弧线，这就要用到 `arc` 路径创建操作，可以画出圆或椭圆的一部分。`arc` 后面方括号中的选项，指定了弧线的样子，一个例子是 `arc[start angle=10, end angle=80, radius=10pt]`，其义如其名。Karl 明显需要一个从  $0^\circ$  到  $30^\circ$  的弧线，半径要小一点，大概是圆的三分之一。在使用弧线路径创建操作时，所画弧线的起始点是当前位置。因此，我们首先要“到达目标位置”。

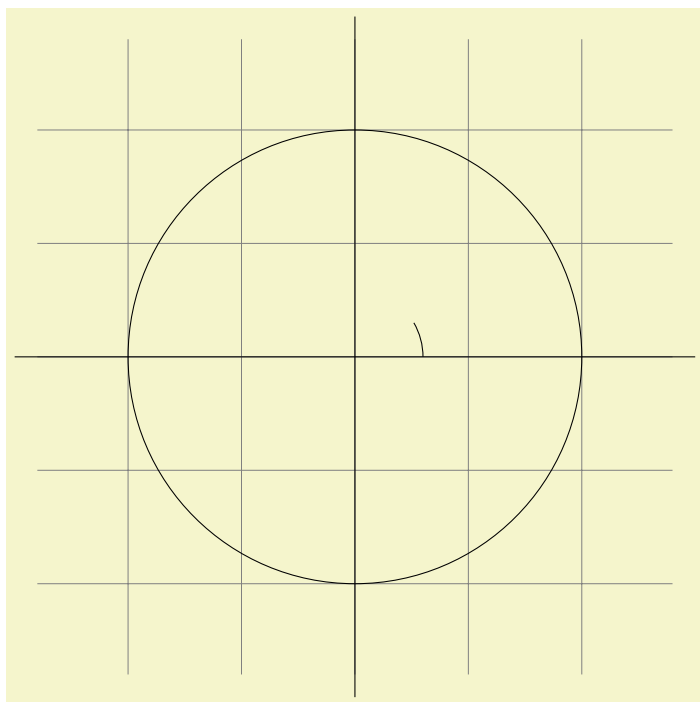


```

\begin{tikzpicture}
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}

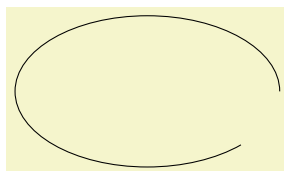
```

Karl 觉得图太小了，如果不学会如何放大，他就不能继续。为此他可以加上 `[scale=3]` 的选项，可以把这个选项挨个加到所有 `\draw` 命令上，但是这太蠢了。所以他把它加到了整个环境上，这样就能将缩放效果应用到里面的所有元素了。



```
\begin{tikzpicture}[scale=3]
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

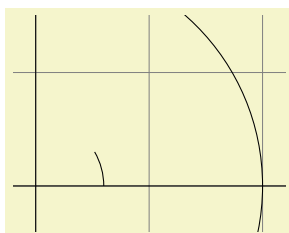
你可以给圆加上“两个”半径，就能得到椭圆弧线。



```
\tikz \draw (0,0)
  arc [start angle=0, end angle=315,
       x radius=1.75cm, y radius=1cm];
```

## 1.11 剪裁路径

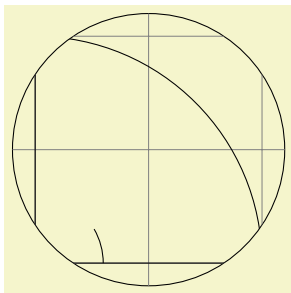
为了给本手册省点地方，最好能剪裁一下 Karl 的图片，这样我们就能只关注“感兴趣的”部分了。在 TikZ 中剪裁很容易，你可以用 `\clip` 命令，剪裁之后画的所有图形。它和 `\draw` 很像，只不过它不画任何东西，而是按给定的路径，剪裁之后画的所有东西。



```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

你也可以同时绘制和剪裁路径，我们只需要给 `\draw` 命令加上 `clip` 选项。（这并不是全部，你也可以用 `\clip` 命令加上 `draw` 选项。不过这还没完。实际上，`\draw` 只是 `\path[draw]` 的简写，`\clip` 是 `\path[clip]` 的简写。因此你也可以用 `\path[draw,clip]`。）例子如下：





```
\begin{tikzpicture}[scale=3]
\clip[draw] (0.5,0.5) circle (.6cm);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

## 1.12 抛物线和正弦路径创建

尽管在这个图片里用不到，Karl 还是很高兴地了解到，路径操作还有 `parabola`、`sin` 和 `cos`，也就是抛物线、正弦和余弦。`parabola` 路径由前后两个点决定，见下例：



```
\tikz \draw (0,0) rectangle (1,1) (0,0) parabola (1,1);
```

也可以在某处加上弯曲 (`bend`):



```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (4,16) (6,12);
```

`sin` 或 `cos` 操作在两点间加上一条正弦或余弦曲线，区间为  $[0, \frac{\pi}{2}]$ 。见如下两例：

一条正弦  $\nearrow$  曲线。

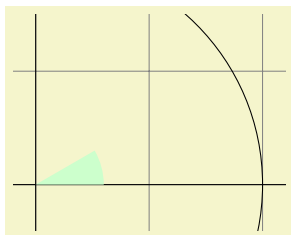
```
一条正弦 \tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1); 曲线。
```



```
\tikz \draw[x=1.57ex,y=1ex] (0,0) sin (1,1) cos (2,0) sin (3,-1) cos (4,0)
(0,1) cos (1,0) sin (2,-1) cos (3,0) sin (4,1);
```

## 1.13 填充和描边

回到之前的图片，Karl 现在想用浅绿色“填充”角。因此他使用 `\fill` 而不是 `\draw`。如下：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\fill[green!20!white] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- (0,0);
\end{tikzpicture}
```

颜色 `green!20!white` 意思是 20% 的绿色混合 80% 的白色。TikZ 中可以用这样的颜色表达式，因为已经导入了 Uwe Kern 的 `xcolor` 宏包，可以参考该宏包的文档，了解更多颜色表达式的细节。

如果 Karl 没有在结尾用 `--(0,0)` “闭合”路径，会怎么样？此时路径会自动“闭合”，因此这句话可以略掉。不过，如果像下面这样写，效果会更好：

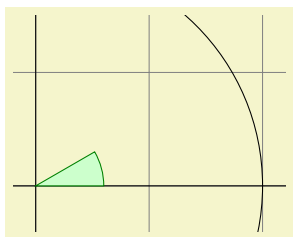
```
\fill[green!20!white] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
```

`--cycle` 将当前路径闭合（实际上是当前路径的当前部分），也就是平滑地连接首尾两点。可以通过下面的例子体会一下差别：



```
\begin{tikzpicture}[line width=5pt]
\draw (0,0) -- (1,0) -- (1,1) -- (0,0);
\draw (2,0) -- (3,0) -- (3,1) -- cycle;
\useasboundingbox (0,1.5); % make bounding box higher
\end{tikzpicture}
```

你也可以用 `\filldraw` 命令同时对路径描边和填充。它会先描边，再填充。这个看上去好像没什么用，不过这可以让你在描边和填充时用不同的颜色。这些可以通过参数指定：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20!white, draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\end{tikzpicture}
```

## 1.14 渐变

Karl 简单想了一下，有没有可能通过渐变让这个角“更精致”。填充时不是用单一的颜色，而是用平滑过渡的不同颜色。`\shade` 或 `\shadedraw` 可以同时实现渐变和描边：



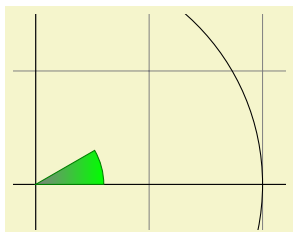
```
\tikz \shade (0,0) rectangle (2,1) (3,0.5) circle (.5cm);
```

默认的渐变是从灰色平滑过渡到白色，要指定不同的颜色，你可以加上选项：



```
\begin{tikzpicture}[rounded corners,ultra thick]
\shade[top color=yellow,bottom color=black] (0,0) rectangle +(2,1);
\shade[left color=yellow,right color=black] (3,0) rectangle +(2,1);
\shadedraw[inner color=yellow,outer color=black,draw=yellow] (6,0) rectangle +(2,1);
\shade[ball color=green] (9,.5) circle (.5cm);
\end{tikzpicture}
```

对 Karl 来说，下面的设置也许更合适：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\shadedraw[left color=gray,right color=green, draw=green!50!black]
(0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\end{tikzpicture}
```

不过，他明智地认为，渐变通常只会使人分心，而不会对图片有任何加成。

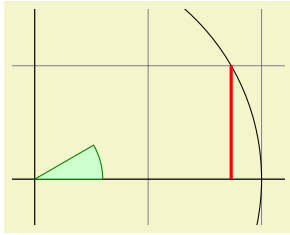
## 1.15 指定坐标

Karl 现在想画上代表正弦值和余弦值的线段。他已经知道可以用 `color=` 选项来设置线的轮廓颜色。那么，用什么方法指定坐标最好呢？

可以用不同的方法指定坐标。最简单的就是 `(10pt,2cm)` 这种，意思是  $x$  轴方向 10pt， $y$  轴方向 2cm。或者你也可以略掉单位，写成 `(1,2)`，意思是“1 倍  $x$  轴单位向量加上 2 倍  $y$  轴单位向量”。单位向量长度默认是 1 cm， $x$  轴和  $y$  轴都是如此。

要用极坐标的方式指定点，可以用记号 `(30:1cm)`，意思是沿 30 度角方向延伸 1 cm。很明显，这个方法比“圆上一点  $(\cos 30^\circ, \sin 30^\circ)$ ”方便多了。

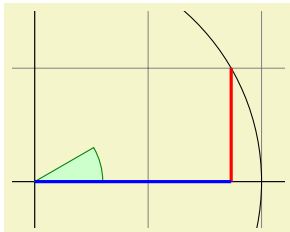
你可以在坐标前加上一个或者两个 `+` 号，比如 `+(0cm,1cm)` 或者 `++(2cm,0cm)`。这时坐标的含义就不一样了，前者表示“在之前指定位置的上方 1 cm”，后者表示“在之前指定位置的右侧 2 cm，并将该点作为新的指定位置”。比如我们可以像下面这样，绘制代表正弦值的线段：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[red,very thick] (30:1cm) -- +(0,-0.5);
\end{tikzpicture}
```

Karl 这里用到了  $\sin 30^\circ = 1/2$ , 不过他觉得学生未必知道这个, 所以他想, 最好有方法能指定“(30:1cm) 到  $x$  轴的垂足”。这确实是可行的, 需要用一个特殊的语法: (30:1cm |- 0,0)。一般来说, ( $p$  |-  $q$ ) 的意思是“过点  $p$  的竖直线与过点  $q$  的水平线的交点”。

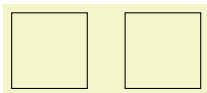
然后我们来画代表余弦值的线段。一种方法是: (30:1cm |- 0,0) -- (0,0)。另一种方法如下: (我们接着正弦值线段那部分往下画)



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,0.75);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[red,very thick] (30:1cm) -- +(0,-0.5);
\draw[blue,very thick] (30:1cm) ++(0,-0.5) -- (0,0);
\end{tikzpicture}
```

注意, 在 (30:1cm) 和 ++(0,-0.5) 之间没有 --。具体点说, 这条路径应该这样解释: “首先, (30:1cm) 告诉我, 要把笔移到  $(\cos 30^\circ, 1/2)$ ; 接着, 我读到了另一个点坐标, 因此将笔移动到新位置, 并且什么也不画, 新点在旧点下方的半个单位长度, 也就是  $(\cos 30^\circ, 0)$ ; 最后, 我把笔移到原点, 不过这次要画出来 (因为 --)。”

下例展示了 + 和 ++ 的区别:



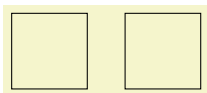
```
\begin{tikzpicture}
\def\rectanglepath{-- ++(1cm,0cm) -- ++(0cm,1cm) -- ++(-1cm,0cm) -- cycle}
\draw (0,0) \rectanglepath;
\draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

和 ++ 相比, 使用 + 时, 路径中的坐标有所不同:



```
\begin{tikzpicture}
\def\rectanglepath{-- +(1cm,0cm) -- +(1cm,1cm) -- +(0cm,1cm) -- cycle}
\draw (0,0) \rectanglepath;
\draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

当然, 这些全都可以采用更加清晰精简的写法 (既不用 + 也不用 ++):



```
\tikz \draw (0,0) rectangle +(1,1) (1.5,0) rectangle +(1,1);
```

## 1.16 交叉路径

现在 Karl 就剩  $\tan \alpha$  对应的线段了, 似乎用变换或者极坐标都不好画。他能做的第一件事——也是最简单的一件事——就是用坐标  $(1, \{\tan(30)\})$ , 因为 TikZ 的数学引擎知道如何计算  $\tan(30)$  这样的式子。注意外面加的花括号 {}, 这是因为 TikZ 的语法分析器在读到第一个圆右括号 ) 时, 会当成是坐标的闭合 (通常, 如果坐标中的元素包含圆括号, 你需要在它外面再套一层花括号)。

Karl 还可以用一种更“精致”也更“几何”的方法, 来计算橙色线段的长度: 他可以指定路径的交点作为坐标。 $\tan \alpha$  值对应的线段起点是  $(1, 0)$ , 然后向“上”, 直到和发自原点、指向 (30:1cm) 的线相交。intersections 库可以支持这类运算。

Karl 要做的就是创建两条“隐形”路径，相交于目标位置。创建隐形路径可以用 `\path`，不加任何 `draw` 或者 `fill` 这样的选项。再给路径加上 `name path` 选项，用于后续的引用。路径创建好后，Karl 就可以用 `name intersections` 选项来命名交点坐标，以供后续引用。

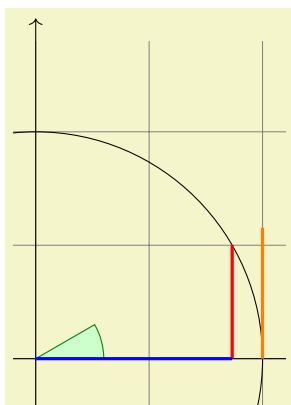
```
\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm); % a bit longer, so that there is an intersection

\draw [name intersections={of=upward line and sloped line, by=x}]
[very thick,orange] (1,0) -- (x);
```

## 1.17 添加箭头

Karl 现在想给坐标轴末尾加个小箭头，他注意到在许多图中，甚至是科学期刊里的，都没有这些箭头，也许是生成的程序做不到这一点。他觉得箭头应该在轴的尾端，他的儿子也同意，而他的学生并不关心箭头。

实际上加箭头非常容易：Karl 给画坐标轴的命令加上了一个 `->` 选项：

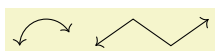


```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,1.51);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[red,very thick] (30:1cm) -- +(0,-0.5);
\draw[blue,very thick] (30:1cm) ++(0,-0.5) -- (0,0);

\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, by=x}]
[very thick,orange] (1,0) -- (x);
\end{tikzpicture}
```

如果 Karl 用 `<-` 而非 `->`，箭头就会加在路径开头，而 `<->` 则会把箭头加在路径两端。

对于可以添加箭头的路径类型，有一定的限制。一般来说，你只能给一条开放的“线”添加箭头。比如，你不能给矩形或者圆加箭头。不过你可以给曲线路径或者是多段路径加箭头，如下：



```
\begin{tikzpicture}
\draw [<->] (0,0) arc [start angle=180, end angle=30, radius=10pt];
\draw [<->] (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl 仔细地观察了一下 TikZ 画在尾端的箭头。放大后像这样： $\rightarrow$ 。形状似乎有点相似，而且事实上，这就是  $\text{T}_{\text{E}}\text{X}$  中标准箭头的尾端，用在  $f: A \rightarrow B$  这样的地方。

Karl 喜欢这个箭头，尤其是它没有很多其他宏包中的“那么粗”。不过他想，有时或许会用到其他类型的箭头。为此，Karl 可以用 `>=<箭头尾端类型>`，其中 `<箭头尾端类型>` 是一个特殊的箭头选项。比方说，如果 Karl 用 `>=Stealth`，那么这是在告诉 TikZ 他喜欢“隐形战斗机式”的箭头：



```
\begin{tikzpicture}[>=Stealth]
\draw [->] (0,0) arc [start angle=180, end angle=30, radius=10pt];
\draw [<-,very thick] (1,0) -- (1.5cm,10pt) -- (2cm,0pt) -- (2.5cm,10pt);
\end{tikzpicture}
```

Karl 在想，箭头类型用这种军事上的名字是否真的有必要。虽然 Karl 的儿子告诉他，微软的 PowerPoint 也用了相同的名字，可是他仍旧不能平静。他决定有朝一日要让学生讨论一下这个话题。

除了 `Stealth`，Karl 还可以选择另外几种预先定义好的箭头，参见第 ?? 小节。此外，如果他需要新式箭头，也可以自己定义。

## 1.18 设置作用域

Karl 已经见到许多图形选项，用于改变路径的渲染方式。他经常想在一套图形命令上用某些特定的图形选项。比如说，他可能想拿一支粗 (`thick`) 笔画三条路径，然后其他的都用“正常”选项。

如果 Karl 想给整张图片设置一个特定的图形选项，那么他只需要把这个选项传给 `\tikz` 或者放在 `{tikzpicture}` 环境中（Gerda 传给 `\tikzpicture`，Hans 则传给 `\starttikzpicture`）。然而，如果 Karl 只想在局部的某组命令中使用这些选项，那么他就需要将这些命令放在 `{scope}` 环境中（Gerda 用 `\scope`，Hans 则用 `\startscope` 和 `\stopscope`）。这个环境把图形选项视为可选参数，只把他们应用在作用域内的命令中，对作用域外的没有影响。

例子如下：



```
\begin{tikzpicture}[ultra thick]
\draw (0,0) -- (0,1);
\begin{scope}[thin]
\draw (1,0) -- (1,1);
\draw (2,0) -- (2,1);
\end{scope}
\draw (3,0) -- (3,1);
\end{tikzpicture}
```

设置作用域还有另外一个有趣的影响：对剪裁区域的任何改动，都局限在作用域内。所以，如果你在作用域内的某个地方用了 `\clip`，那么剪裁的效果会在作用域外消失。这很有用，因为只有这种方法能“放大”剪裁的区域。

Karl 还发现，传给 `\draw` 这类命令的选项，只对当前命令有用。这下情况就有点复杂了。首先，传给 `\draw` 这类命令的选项，并不是真的作用于命令，而是“路径选项”，可以在路径的任何位置给出。所以，除了用 `\draw[thin] (0,0) -- (1,0);`，你还可以写成 `\draw (0,0) [thin] -- (1,0);` 或者 `\draw (0,0) -- (1,0) [thin];`。这些写法的效果是一样的。看上去有点奇怪，按理说在最后一种写法中，`thin` 应该只对线段 `(0,0) -- (1,0)` 之后绘制的元素有影响。事实上，如果你同时在路径中使用 `thin` 和 `thick`，那么最后写的选项会“胜出”。

在读上面的时候，Karl 注意到“大多数”图形选项作用于整条路径。不过，事实上所有变换相关的选项并不对整条路径起作用，他们只对在自己之后出现的元素有效。我们稍后会更加详细地考察这一点。尽管如此，所有的选项都只在该条路径内有效。

## 1.19 变换

当你指定一个坐标时，比如 `(1cm,1cm)`，这个坐标在页面的哪个位置呢？为了确定坐标在页面中的最终位置，TikZ、TeX 以及 PDF 或者 Postscript 都对给出的坐标施加了变换。

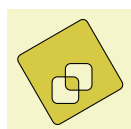
TikZ 中有大量选项，供你在 TikZ 的独立坐标系中变换坐标。比如，`xshift` 选项将之后所有的点移动一定距离：



```
\tikz \draw (0,0) -- (0,0.5) [xshift=2pt] (0,0) -- (0,0.5);
```

值得注意的一点是，你可以在“路径中间”加入变换，这个特性在 PDF 和 PostScript 里是没有的。这是因为 TikZ 一直记录着自己的变换矩阵。

这里有个更复杂的例子：



```
\begin{tikzpicture}[even odd rule,rounded corners=2pt,x=10pt,y=10pt]
\filldraw[fill=yellow!80!black] (0,0) rectangle (1,1)
[xshift=5pt,yshift=5pt] (0,0) rectangle (1,1)
[rotate=30] (-1,-1) rectangle (2,2);
\end{tikzpicture}
```

最有用的变换有：移动选项 `xshift` 和 `yshift`，而 `shift` 则移动到一个给定的点，比如 `shift={(1,0)}` 或者 `shift={+(0,)}`（花括号是必须要加的，这样 TeX 就不会把逗号当成选项之间的分隔符）；旋转选项 `rotate` 用于旋转一定角度（`rotate around` 则围绕一个定点旋转）；缩放选项 `scale` 用于缩放一定比例，`xscale` 和 `yscale` 分别对  $x$  方向和  $y$  方向缩放（`xscale=-1` 则表示水平翻转）；倾斜选项 `xslant` 和 `yslant`。如果这些我提到或者没提到的变换不够用的话，你还可以用 `cm` 选项指定一个任意的变换矩阵。顺便说下，Karl 的学生并不知道什么是变换矩阵。

## 1.20 重复：For 循环

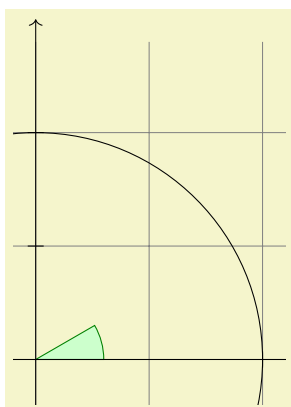
Karl 的下一个目标是在  $-1$ 、 $-1/2$ 、 $1/2$  和  $1$  的位置上标出刻度。因此最好能用“循环”，尤其是他想在每个位置做相同的事情。有几个不同的宏包可以实现这一点。L<sup>A</sup>T<sub>E</sub>X 有相关的内部命令，`pstricks` 有强大的 `\multido` 命令。这些都可以结合 TikZ 使用，所以如果你对他们很熟悉，放心用。TikZ 里有另一个命令，叫 `\foreach`，我加入这个命令是因为我一直记不住其他宏包的语法。`\foreach` 定义在 `pgffor` 宏包中，可以独立于 TikZ 使用，不过用 TikZ 会自动将它包含进来。

`\foreach` 命令的基本形式很简单：

```
x = 1, x = 2, x = 3, \foreach \x in {1,2,3} {\x = \x$, }
```

语法的一般格式为 `\foreach <变量> in {<变量列表>} <命令>`。在 `<命令>` 中，`<变量>` 会被赋予不同的值。如果 `<命令>` 没有以花括号开始，那么在下一个分号之前，所有的语句都认定为 `<命令>`。

Karl 可以用下面的代码在坐标轴上标刻度：



```
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,1.51);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[>-] (-1.5,0) -- (1.5,0);
\draw[>-] (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];

\foreach \x in {-1cm,-0.5cm,1cm}
  \draw (\x,-1pt) -- (\x,1pt);
\foreach \y in {-1cm,-0.5cm,0.5cm,1cm}
  \draw (-1pt,\y) -- (1pt,\y);
\end{tikzpicture}
```

实际上有很多不同的方法创建刻度。比如 Karl 可以把 `\draw ...` 放在花括号里面，又或者像下面这样：

```
\foreach \x in {-1,-0.5,1}
\draw[xshift=\x cm] (0pt,-1pt) -- (0pt,1pt);
```

Karl 很好奇，如果碰到更复杂的情况，比如要画 20 个刻度，会怎么样。要在 `\foreach` 里挨个写上所有数字，真的很麻烦。事实上，可以在 `\foreach` 语句里用 `...` 来遍历大量数值（不过必须是没有量纲的实数），例子如下：



```
\tikz \foreach \x in {1,...,10}
\draw (\x,0) circle (0.4cm);
```

如果在 `...` 之前有两个数，那么 `\foreach` 语句则会将这两个数之差作为步长：

```
\tikz \foreach \x in {-1,-0.5,...,1}
\draw (\x cm,-1pt) -- (\x cm,1pt);
```

我们也可以嵌套循环，创造出有趣的效果：



1,5	2,5	3,5	4,5	5,5	7,5	8,5	9,5	10,5	11,5	12,5
1,4	2,4	3,4	4,4	5,4	7,4	8,4	9,4	10,4	11,4	12,4
1,3	2,3	3,3	4,3	5,3	7,3	8,3	9,3	10,3	11,3	12,3
1,2	2,2	3,2	4,2	5,2	7,2	8,2	9,2	10,2	11,2	12,2
1,1	2,1	3,1	4,1	5,1	7,1	8,1	9,1	10,1	11,1	12,1

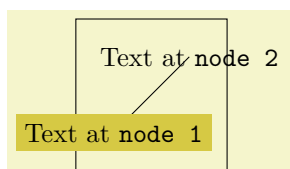
```
\begin{tikzpicture}
\foreach \x in {1,2,...,5,7,8,...,12}
\foreach \y in {1,...,5}
{
\draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
\draw (\x,\y) node{\x,\y};
}
\end{tikzpicture}
```

`\foreach` 语句还能完成更巧妙的任务，不过上面的例子已经足够传达精神了。

## 1.21 添加文本

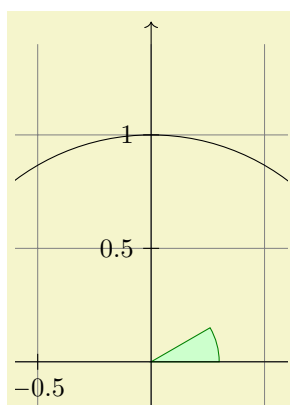
Karl 现在已经对图片很满意了，不过最重要的部分，也就是标签，还没有加！

TikZ 提供了一个好用且强大的系统，用于在图片上的特定位置添加文本以及——更一般地——复杂的形状。其基本思想是：如果 TikZ 正在创建一条路径，并且在路径中遇到关键词 `node`，它会将之后的内容作为节点说明。关键词 `node` 后面通常跟着一些选项，然后是文本（写在花括号里）。这些文本放在一个普通的 TeX 盒子中（如果节点说明后面直接跟着一个坐标——这很常见——那么 TikZ 就能变些魔术，这样甚至可以在盒子中按照原样呈现文本（verbatim text）），并置于当前位置，也就是之前指定的位置（也可能有些许移动，取决于给定的选项）。不过，只有在整条路径完成了描边/填充/渐变/剪裁/其他之后，才会绘制所有节点。



```
\begin{tikzpicture}
\draw (0,0) rectangle (2,2);
\draw (0.5,0.5) node [fill=yellow!80!black]
{Text at \verb!node 1!};
-- (1.5,1.5) node {Text at \verb!node 2!};
\end{tikzpicture}
```

很明显，Karl 未必一定是将节点放在指定位置上，也有可能要放到左边或者右边。所以，每个节点对象都有几个锚点。比如说，`north` 锚点在形状的正上方，`south` 锚点在正下方，`north east` 锚点在右上角。当你加上选项 `anchor=north` 时，锚点会放在当前位置，文本则位于当前位置的下方。Karl 就用它来标刻度，如下：



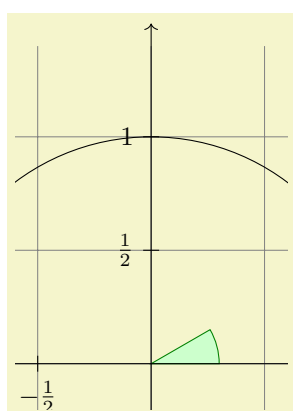
```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[>-] (-1.5,0) -- (1.5,0); \draw[>-] (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];

\foreach \x in {-1,-0.5,1}
\draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {\x};
\foreach \y in {-1,-0.5,0.5,1}
\draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=west] {\y};
\end{tikzpicture}
```

这已经很不错了，利用锚点，Karl 现在可以加上大多数其他的文本元素了。然而 Karl 在想，尽管“正确”，但是很反直觉，因为要把元素放在某个点的下面，需要用锚点 `north`（而北是在元素正上方的）。因此，有一个选项叫 `below`，效果等同于 `anchor=north`。类似地，`above right` 等同于 `anchor=south west`。此外，`below` 还有一个可选参数，用于指定形状向下偏移的距离。因此，`below=1pt` 可以把一个文本标签放在某点下面，并且向下偏移 1pt。

Karl 对刻度还是有点不满意，他想显示  $\frac{1}{2}$  或者  $\frac{1}{3}$  而不是 0.5。一部分原因是，他想展示一下 T<sub>E</sub>X 和 TikZ 的出色能力，另一部分原因是，对于  $\frac{1}{3}$  或者  $\pi$  这样的位置，刻度用“数学”形式比“数值”形式要好得多。不过他的学生倒是更喜欢 0.5 而不是  $\frac{1}{2}$ ，因为他们一般不是很喜欢分数。

Karl 现在碰到了一个在 `\foreach` 语句中，位置 `\x` 还是需要用 0.5，因为 TikZ 不知道 `\frac{1}{2}` 是什么意思，而打印出来的文本应该是 `\frac{1}{2}`。为了解决这个问题，`\foreach` 提供了一个特殊的语法：不单是 `\x` 这样的单个变量，还可以指定两个（甚至更多个）变量，之间用斜杠分隔，比如 `\x / \xtext`。在 `\foreach` 要遍历的集合里，元素也必须是 `<第一个值>/<第二个值>` 这种形式。在每次迭代中，`\x` 会被赋为 `<第一个值>`，`\xtext` 会被赋为 `<第二个值>`。如果没有给定 `<第二个值>`，那么其值等于 `<第一个值>`。因此，下面是新的画刻度的代码：

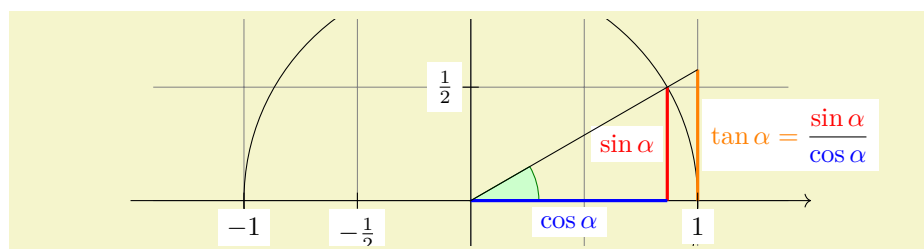


```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[>-] (-1.5,0) -- (1.5,0); \draw[>-] (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];

\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
\draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {\xtext};
\foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
\draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=west] {\ytext};
\end{tikzpicture}
```

Karl 对结果相当满意，但是他的儿子指出，这幅图并不完美：网格和圆影响了数字的显示。Karl 对这个不是很关心（他的学生甚至都没注意到），但是他的儿子坚持说有一个简单的解决方法：可以用 `[fill=white]` 选项给文本加个白色背景。

Karl 下一步打算加上  $\sin \alpha$  这样的标签，他想把标签放在“线段正中间”。Karl 不用在线段的端点指定标签 `node {\sin \alpha}`（这会将标签放在端点处），他可以直接把标签放在 `--` 之后、坐标之前。默认情况下，这会将标签放在线段的正中间。不过 `pos=` 选项可以修改标签的位置，`near start` 和 `near end` 也是类似的作用。





```

\begin{tikzpicture}[scale=3]
\clip (-2,-0.2) rectangle (2,0.8);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[->] (-1.5,0) -- (1.5,0) coordinate (x axis);
\draw[->] (0,-1.5) -- (0,1.5) coordinate (y axis);
\draw (0,0) circle [radius=1cm];

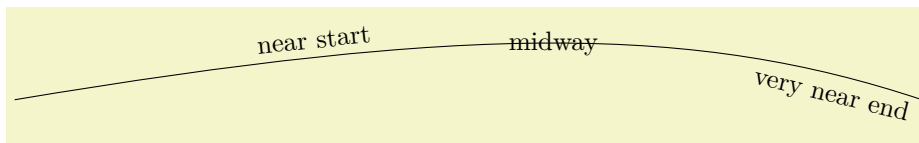
\draw[very thick,red]
  (30:1cm) -- node[left=1pt,fill=white] {\sin \alpha} (30:1cm |- x axis);
\draw[very thick,blue]
  (30:1cm |- x axis) -- node[below=2pt,fill=white] {\cos \alpha} (0,0);
\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, by=t}]
  [very thick,orange] (1,0) -- node [right=1pt,fill=white]
  {\displaystyle \tan \alpha \color{black}=
   \frac{\color{red}\sin \alpha}{\color{blue}\cos \alpha}} (t);

\draw (0,0) -- (t);

\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
  \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north,fill=white] {\xtext};
\foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
  \draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=west,fill=white] {\ytext};
\end{tikzpicture}

```

你也可以把标签放在曲线上，并且加上 `sloped` 选项，这会旋转标签，保持和该点切线的斜率一致。如下：



```

\begin{tikzpicture}
\draw (0,0) .. controls (6,1) and (9,1) ..
  node[near start,sloped,above] {near start}
  node {midway}
  node[very near end,sloped,below] {very near end} (12,0);
\end{tikzpicture}

```

现在只剩图片右边的说明文字没加了。这里的主要问题是限制文本“标签”的宽度，因为文字很多，所以需要换行。好在 Karl 可以用 `text width=6cm` 得到想要的结果。下面是完整的代码：

```

\begin{tikzpicture}
  [scale=3,line cap=round,
  % Styles
  axes/.style=,
  important line/.style={very thick},
  information text/.style={rounded corners,fill=red!10,inner sep=1ex}]

  % Colors
  \colorlet{anglecolor}{green!50!black}
  \colorlet{sincolor}{red}
  \colorlet{tancolor}{orange!80!black}
  \colorlet{coscolor}{blue}

  % The graphic
  \draw[help lines,step=0.5cm] (-1.4,-1.4) grid (1.4,1.4);

  \draw (0,0) circle [radius=1cm];

  \begin{scope}[axes]
    \draw[->] (-1.5,0) -- (1.5,0) node[right] {$x$} coordinate(x axis);
    \draw[->] (0,-1.5) -- (0,1.5) node[above] {$y$} coordinate(y axis);

    \foreach \x/\xtext in {-1, -.5/-\frac{1}{2}, 1}
      \draw[xshift=\x cm] (0pt,1pt) -- (0pt,-1pt) node[below,fill=white] {$\xtext$};

    \foreach \y/\ytext in {-1, -.5/-\frac{1}{2}, .5/\frac{1}{2}, 1}
      \draw[yshift=\y cm] (1pt,0pt) -- (-1pt,0pt) node[left,fill=white] {$\ytext$};
  \end{scope}

  \filldraw[fill=green!20,draw=anglecolor] (0,0) -- (3mm,0pt)
    arc [start angle=0, end angle=30, radius=3mm];
  \draw (15:2mm) node[anglecolor] {$\alpha$};

  \draw[important line,sincolor]
    (30:1cm) -- node[left=1pt,fill=white] {$\sin \alpha$} (30:1cm |- x axis);

  \draw[important line,coscolor]
    (30:1cm |- x axis) -- node[below=2pt,fill=white] {$\cos \alpha$} (0,0);

  \path [name path=upward line] (1,0) -- (1,1);
  \path [name path=sloped line] (0,0) -- (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, by=t}]
    [very thick,orange] (1,0) -- node [right=1pt,fill=white]
    {$\displaystyle \tan \alpha \color{black} = \frac{\color{red}\sin \alpha}{\color{blue}\cos \alpha}$} (t);

  \draw (0,0) -- (t);

  \draw[xshift=1.85cm]
    node[right,text width=6cm,information text]
    {
      在本例中, {\color{anglecolor} 角 $\alpha$} 为  $30^\circ$  (等于  $\pi / 6$  弧度),
      {\color{sincolor} $\sin \alpha$} 的正弦}, 也就是红色线段的高度, 为
      \[
        {\color{sincolor} \sin \alpha} = 1/2.
      \]
      根据勾股定理……
    };
\end{tikzpicture}

```

## 1.22 片图：再访角形

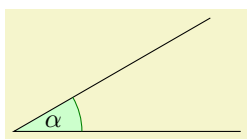
Karl 在想, 图片中的部分代码可能很有用, 将来也许还会用到。一个很自然的做法, 就是把他想重用的代码写成  $\text{T}_\text{E}_\text{X}$  宏。不过  $\text{TikZ}$  提供了另外一种做法, 并且直接集成到了分析器里, 那就是: 片图 (pics)!

一个片图 (pic) 并不是一张完整的图片 (picture), 因此得到了这个改编的名称。其中的思想是, 一个片图就是一段简单的代码, 你可以用 `pic` 命令把它加到图片中的不同位置, 语法和 `node` 几乎一样。主要的不同在于, 在花括号中不是写上要显示的文本, 而是写上想要显示的预先定义的图片的名字。

定义新的片图很简单, 参考第 ?? 节, 不过我们现在只是想用这样一个预先定义好的片图: `angle` 片图。

正如其名所示，它是一个小小的角，包含一个小楔形和一条弧线，以及一些文本（在下面的例子中，Karl 需要导入 `angles` 和 `quotes` 库）。这个片图的有用之处在于，它会自动计算楔形的尺寸。

`angle` 片图在线  $BA$  和  $BC$  之间画一个角，其中  $A$ 、 $B$ 、 $C$  是三个点的坐标。在我们的例子中， $B$  点是原点， $A$  点在  $x$  轴上， $C$  点在  $30^\circ$  角对应的线上。



```
\begin{tikzpicture}[scale=3]
  \coordinate (A) at (1,0);
  \coordinate (B) at (0,0);
  \coordinate (C) at (30:1cm);

  \draw (A) -- (B) -- (C)
    pic [draw=green!50!black, fill=green!20, angle radius=9mm,
         "$\alpha$"] {angle = A--B--C};
\end{tikzpicture}
```

让我们看看这里发生了什么。首先我们用 `\coordinate` 命令指定了三个坐标，它可以为图片中的特定坐标命名。然后是普通的 `\draw` 接一个 `pic` 命令，这个命令有很多选项，再后面接了一对花括号，其中包含了最重要的部分：我们给出了说明，要加入一个 `angle` 片图，并且这个角在点  $A$ 、 $B$  和  $C$ （我们也可以用其他名字）之间。注意，我们想在片图中呈现的文本，是作为 `pic` 选项给出的，并用双引号指定，而不是放在花括号里。

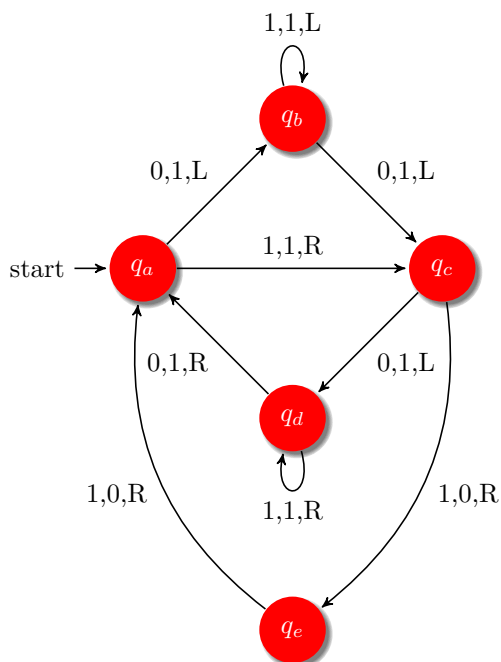
要了解更多关于片图的内容，请参考第 ?? 节。

## 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);

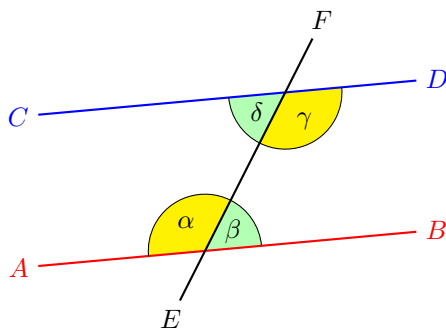
\node [right=1cm,text width=8cm] at (C)
{
The current candidate for the busy beaver for five states. It is
presumed that this Turing machine writes a maximum number of
1's before halting among all Turing machines with five states
and the tape alphabet  $\{0, 1\}$ . Proving this conjecture is an
open 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)!1.5!(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., $\color{red}AB \parallel \color{blue}CD$,
  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. *TikZ* comes with powerful such algorithms, but you can also implement new algorithms in the Lua programming language.

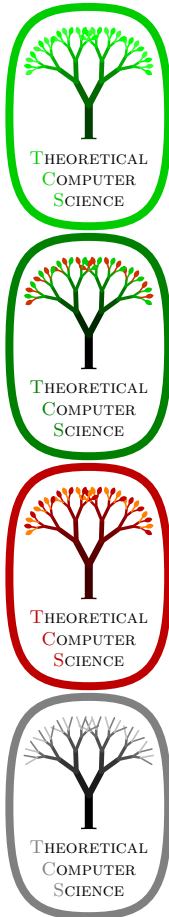
You need to use `LuaTeX` to typeset this part of the manual (and, also, to use algorithmic graph drawing).

# 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 TikZ 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}
{\pgfarrowslefttextend{-2pt} \pgfarrowsrighttextend{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;

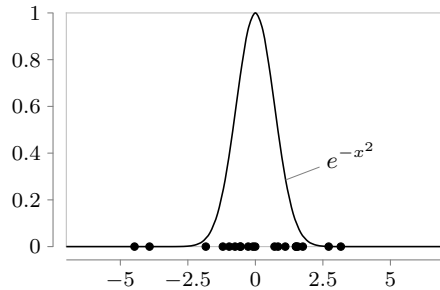
    \coordinate (root) [grow cyclic,rotate=90]
    child {
      child [line cap=round] foreach \a in {0,1} {
        child foreach \b in {0,1} {
          child foreach \c in {0,1} {
            child foreach \d in {0,1} {
              child foreach \leafcolor in {leaf a,leaf b}
                { edge 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={\mathit{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);
};
```

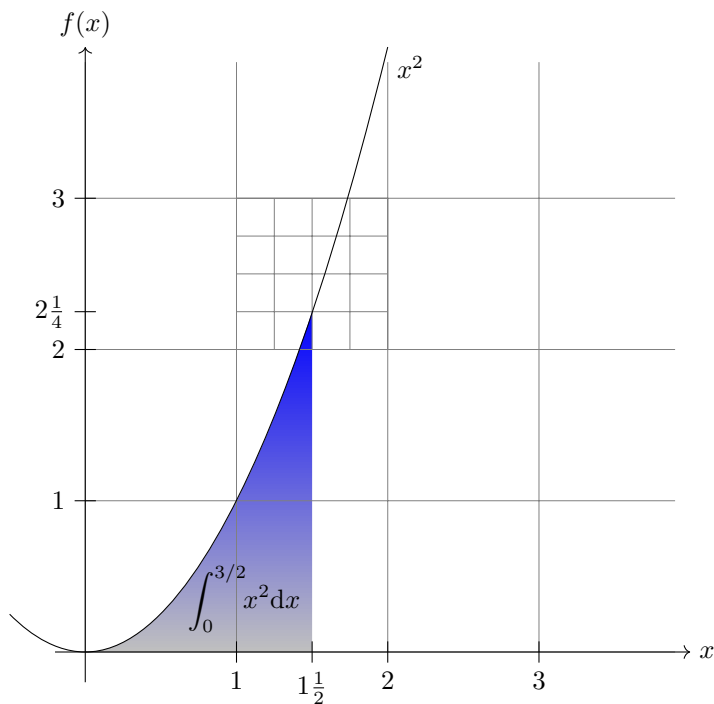


# Part VII

## 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

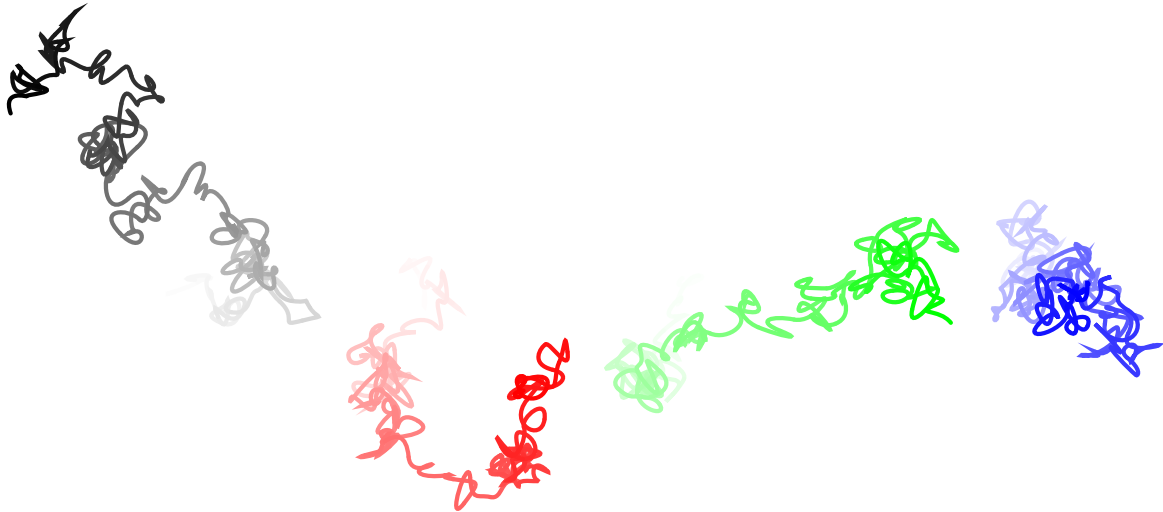
# Mathematical and Object-Oriented Engines

*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  $(5\text{cm}*3, 6\text{cm}/4)$ , but the mathematical engine can also be used independently of PGF and TikZ.

The job of the object-oriented engine is to support simple object-oriented programming in T<sub>E</sub>X. It allows the definition of *classes* (without inheritance), *methods*, *attributes* and *objects*.



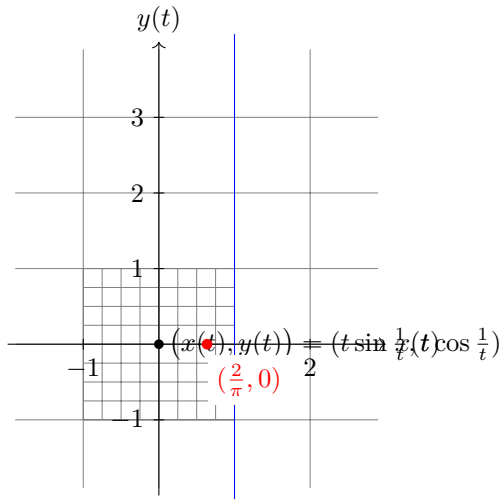
```
\pgfmathsetseed{1}
\foreach \col in {black,red,green,blue}
{
  \begin{tikzpicture}[x=10pt,y=10pt,ultra thick,baseline,line cap=round]
    \coordinate (current point) at (0,0);
    \coordinate (old velocity) at (0,0);
    \coordinate (new velocity) at (rand,rand);

    \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

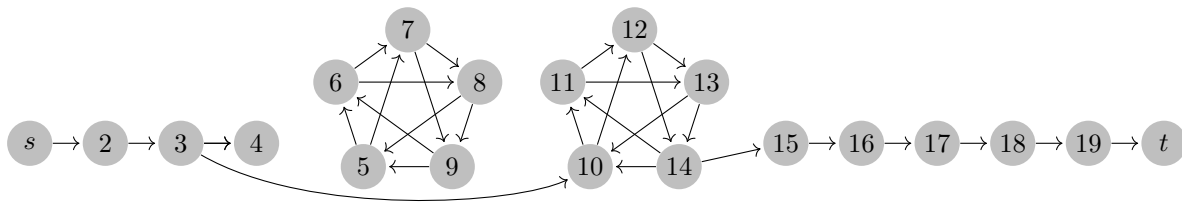
# The System Layer

*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  $\text{\TeX}$ -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) {$\text$};

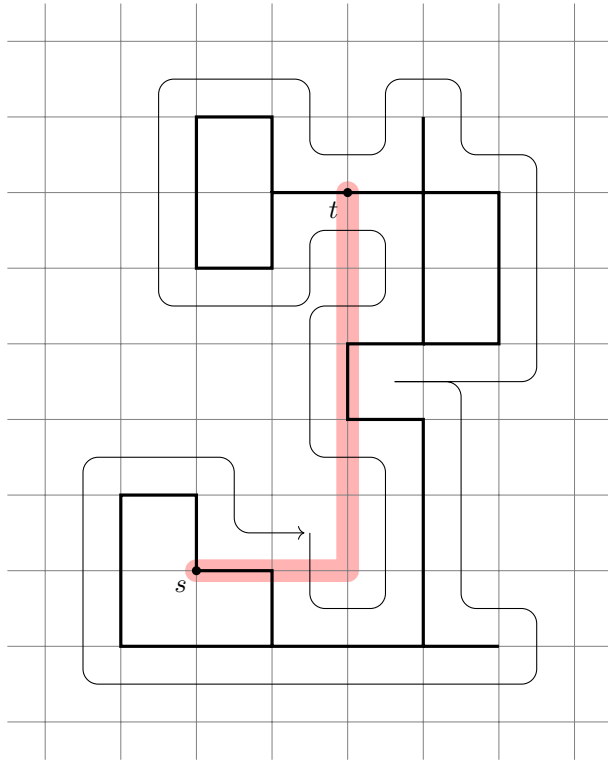
  \foreach \from/\to in {s/2,2/3,3/4,3/4,15/16,16/17,17/18,18/19,19/t}
    \draw (G-\from) -- (G-\to);

  \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}
```