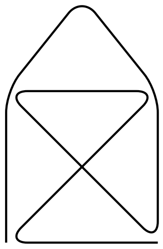
Part I

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);
```

1 教程：给 Karl 的学生的一张图

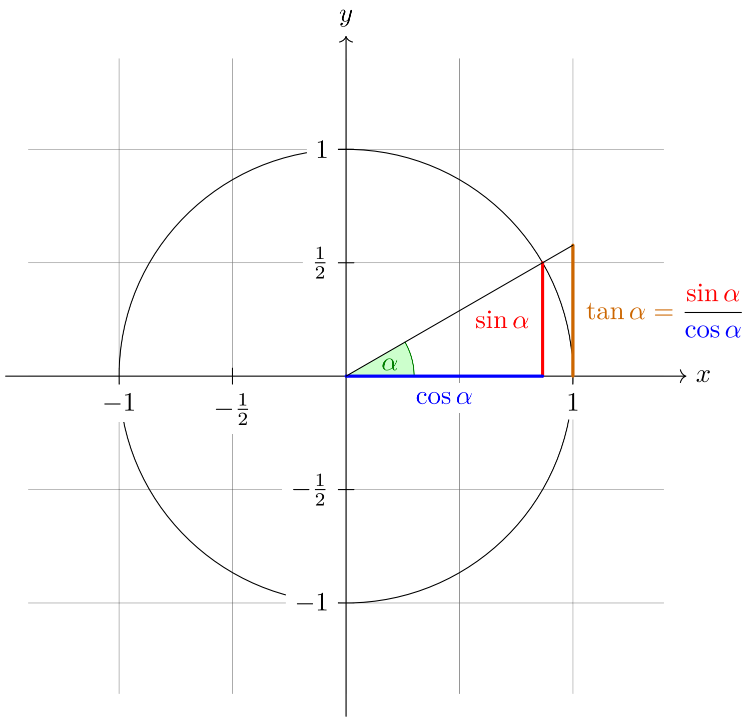
该教程是写给 TikZ 的新手的，它并不详细罗列 TikZ 的所有特性，而是只讲那些你可能立马会用到的。

Karl 是一名高中数学和化学老师，他以前用 LaTeX 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 α 为 30°（等于 $\pi/6$ 弧度），α 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 α 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

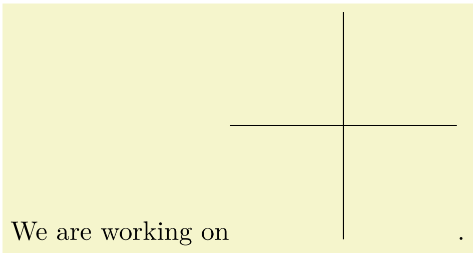
要在 TikZ 中画一张图片，你需要在图片开头告诉 TeX 或者 LaTeX 你想开始画一张图。在 LaTeX 中需要用 `{tikzpicture}` 环境，在 plain TeX 中你只要用 `\tikzpicture` 开始图片，再用 `\endtikzpicture` 结束图片。

1.2.1 设置 LaTeX 中的环境

Karl 是一个 LaTeX 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 TEX 中的环境

Karl 的妻子 Gerda，恰好也是一名数学老师，她不是 LATEX 用户，而是用 plain TEX ，因为她更喜欢“老派的方法”。她也能用 Ti*k*Z，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。Ti*k*Z 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 ConTEXt 中的环境

Karl 的叔叔 Hans 用 ConTEXt，Hans 也能像 Gerda 一样使用 Ti*k*Z。他需要把 \usepackage{tikz} 换成 \usemodule[tikz]，同样，\begin{tikzpicture} 和 \end{tikzpicture} 也要换成 \starttikzpicture 和 \stoptikzpicture。

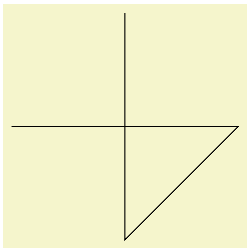
他的样例代码像这样：

```
%% 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

Ti*k*Z 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：



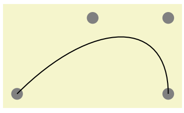
```
\tikz \draw (-1.5,0) -- (1.5,0) -- (0,-1.5) -- (0,1.5);
```

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 Ti*k*Z 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，Ti*k*Z 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，那么曲线开始时会沿着从 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。

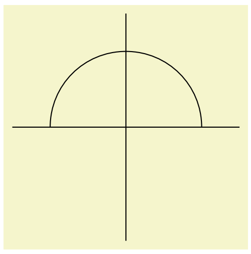
例子如下（为了更清楚地说明，这里画上了控制点）：



```
\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：

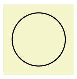


```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (-1,0) .. controls (-1,0.555) and (-0.555,1) .. (0,1)
               .. controls (0.555,1) and (1,0.555) .. (1,0);
\end{tikzpicture}
```

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。

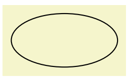
1.5 圆形路径创建

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];
```

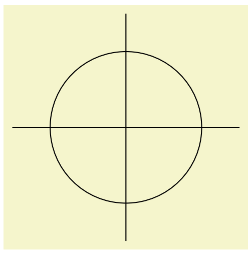
你还可以用 ellipse 画椭圆，它有两个半径：



```
\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” ，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：

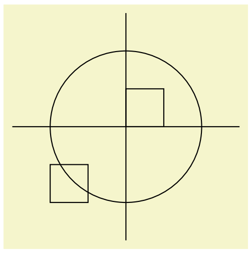


```
\begin{tikzpicture}
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

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，Ti*k*Z 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}
```

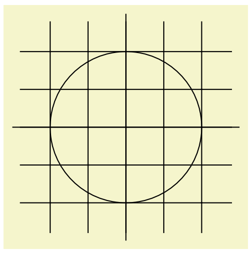
也许在其他情况下这个方法还不错，但是并没有真的解决 Karl 的问题：首先我们需要很多很多矩形，并且边界处是不“闭合”的。

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

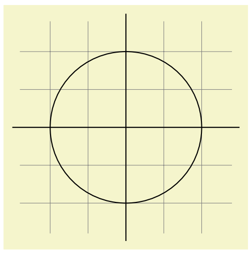
grid 路径操作在当前路径加上网格。它会画出组成网格的线，grid 前后两个点坐标构成了网格矩形的两个对角。比如，代码 `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 生成网格 。注意到 `\draw` 后面的选项，可以指定网格的宽度（还可以用 xstep 和 ystep 来分别设置网格间隔）。正如 Karl 很快会学到的，用这些选项可以影响非常多的事物。

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

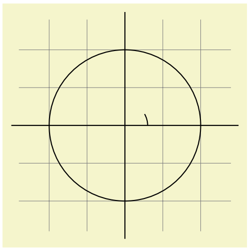
Karl 在想还有什么命令能够影响路径的绘制。他已经注意到 `color=`⟨*color*⟩ 可以设置线的颜色，而 `draw=`⟨*color*⟩ 的作用几乎一样，只不过它只设置线的轮廓颜色，而填充颜色可以设置成另外一种（后面要填充弧线时，Karl 会用到这个）。

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。

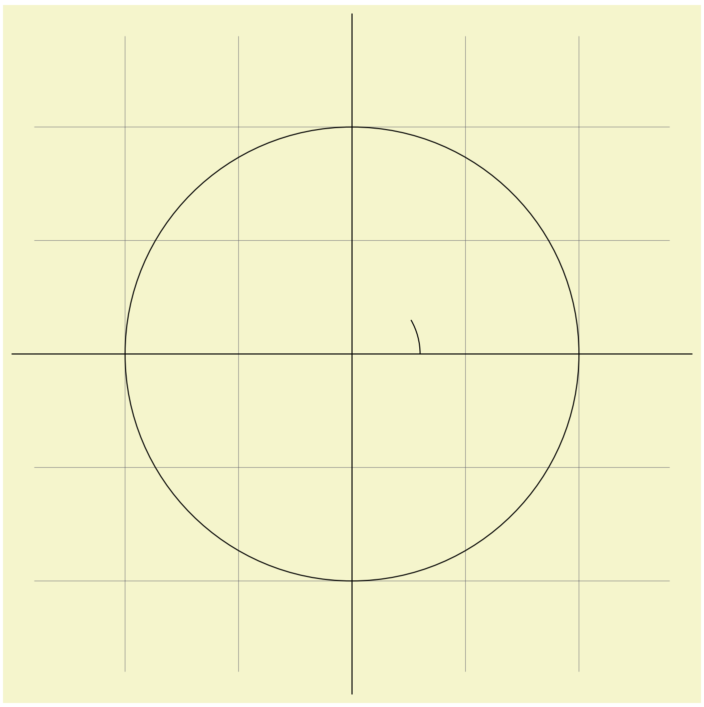
1.10 弧线路径创建

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° 到 30° 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”。



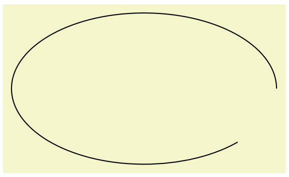
```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

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}
```

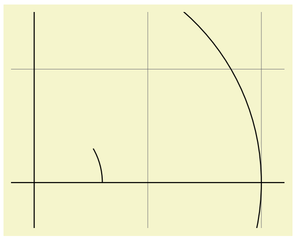
你可以给圆加上“两个”半径，就能得到椭圆弧线。



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

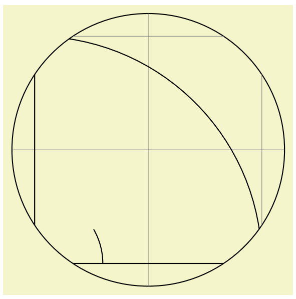
1.11 剪裁路径

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 Ti*k*Z 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}
```

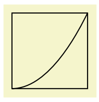
你也可以同时绘制和剪裁路径，我们只需要给 `\draw` 命令加上 `clip` 选项。（这并不是全部，你也可以用 `\clip` 命令加上 `draw` 选项。不过这也还没完。实际上，`\draw` 只是 `\path[draw]` 的简写，`\clip` 是 `\path[clip]` 的简写。因此你也可以用 `\path[draw,clip]`。）例子如下：



```
\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}
```

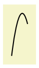
1.12 抛物线和正弦路径创建

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：



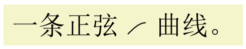
```
\tikz \draw (0,0) rectangle (1,1)  (0,0) parabola (1,1);
```

也可以在某处加上弯曲（`bend`）：

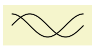


```
\tikz \draw[x=1pt,y=1pt] (0,0) parabola bend (4,16) (6,12);
```

`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：



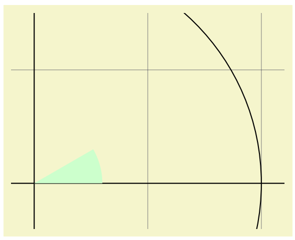
```
一条正弦 \tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1); 曲线。
```



```
\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：



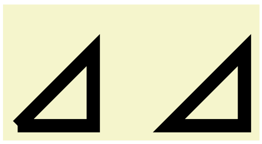
```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,0.75);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \fill[green!20!white] (0,0) -- (3mm,0mm)
    arc [start angle=0, end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

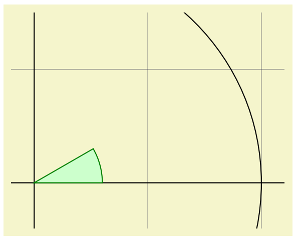
```
  \fill[green!20!white] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
```

`--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}
```

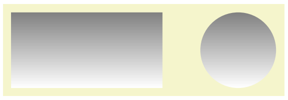
你也可以用 `\filldraw` 命令同时对路径描边和填充。它会先描边，再填充。这个看上去好像没什么用，不过这可以让你在描边和填充时用不同的颜色。这些可以通过参数指定：



```
\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}
```

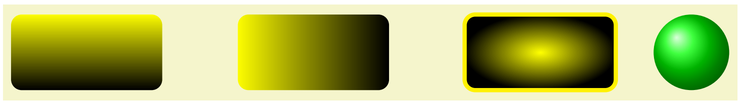
1.14 渐变

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：



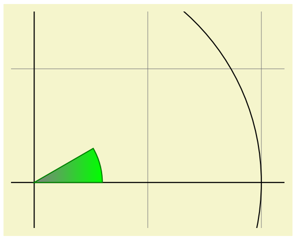
```
  \tikz \shade (0,0) rectangle (2,1)  (3,0.5) circle (.5cm);
```

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。

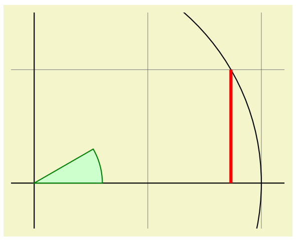
1.15 指定坐标

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 $(\cos 30°, \sin 30°)$”方便多了。

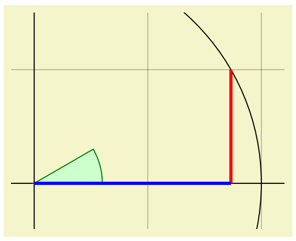
你可以在坐标前加上一个或者两个 `+` 号，比如 `+(0cm,1cm)` 或者 `++(2cm,0cm)`。这时坐标的含义就不一样了，前者表示“在之前指定位置的上方 1 cm”，后者表示“在之前指定位置的右侧 2 cm，并将该点作为新的指定位置”。比如我们可以像下面这样，绘制代表正弦值的线段：



```
\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 $\sin 30° = 1/2$，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，`(`$\langle p\rangle$` |- `$\langle q\rangle$`)` 的意思是“过点 p 的竖直线与过点 q 的水平线的交点”。

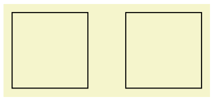
然后我们来画代表余弦值的线段。一种方法是：`(30:1cm |- 0,0) -- (0,0)`。另一种方法如下：（我们接着正弦值线段那部分往下画）



```
\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}
```

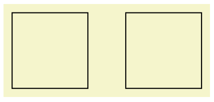
注意，在 `(30:1cm)` 和 `++(0,-0.5)` 之间没有 `--`。具体点说，这条路径应该这样解释：“首先，`(30:1cm)` 告诉我，要把笔移到 $(\cos 30°, 1/2)$；接着，我读到了另一个点坐标，因此将笔移动到新位置，并且什么也不画，新点在旧点下方的半个单位长度，也就是 $(\cos 30°, 0)$；最后，我把笔移到原点，不过这次要画出来（因为有 `--`）。”

下例展示了 `+` 和 `++` 的区别：



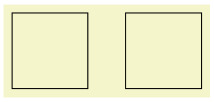
```
\begin{tikzpicture}
  \def\rectanglepath{-- ++(1cm,0cm)  -- ++(0cm,1cm)  -- ++(-1cm,0cm) -- cycle}
  \draw (0,0) \rectanglepath;
  \draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

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 Ti*k*Z 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 Ti*k*Z 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。

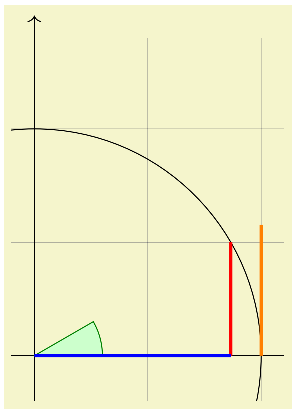
Karl 要做的就是创建两条“隐形”路径，相交于目标位置。创建隐形路径可以用 \path，不加任何 draw 或者 fill 这样的选项。再给路径加上 name path 选项，用于后续的引用。路径创建好后，Karl 就可以用 name intersections 选项来命名交点坐标，以供后续引用。

```
\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。

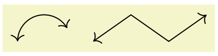
实际上加箭头非常容易：Karl 给画坐标轴的命令加上了一个 -> 选项：



```
\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。

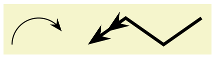
对于可以添加箭头的路径类型，有一定的限制。一般来说，你只能给一条开放的“线”添加箭头。比如，你不能给矩形或者圆加箭头。不过你可以给曲线路径或者是多段路径加箭头，如下：



```
\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 Ti*k*Z 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 TEX 中标准箭头的尾端，用在 $f\colon A \to B$ 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 Ti*k*Z 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。

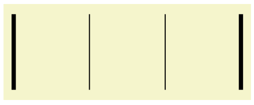
除了 Stealth，Karl 还可以选择另外几种预先定义好的箭头，参见第 **??** 小节。此外，如果他需要新式箭头，也可以自己定义。

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 里或者放在 {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}
```

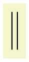
设置作用域还有另外一个有趣的影响：对剪裁区域的任何改动，都局限在作用域内。所以，如果你在作用域内的某个地方用了 \clip，那么剪裁的效果会在作用域外消失。这很有用，因为只有这种方法能“放大”剪裁的区域。

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”。

1.19 Transformations

When you specify a coordinate like (1cm,1cm), where is that coordinate placed on the page? To determine the position, Ti*k*Z, TEX, and PDF or PostScript all apply certain transformations to the given coordinate in order to determine the final position on the page.

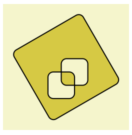
Ti*k*Z provides numerous options that allow you to transform coordinates in Ti*k*Z's private coordinate system. For example, the xshift option allows you to shift all subsequent points by a certain amount:



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

It is important to note that you can change transformation "in the middle of a path," a feature that is not supported by PDF or PostScript. The reason is that Ti*k*Z keeps track of its own transformation matrix.

Here is a more complicated example:



```
\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}
```

The most useful transformations are xshift and yshift for shifting, shift for shifting to a given point as in shift={(1,0)} or shift={+(0,0)} (the braces are necessary so that TEX does not mistake the comma for separating options), rotate for rotating by a certain angle (there is also a rotate around for rotating around a given point), scale for scaling by a certain factor, xscale and yscale for scaling only in the x- or y-direction (xscale=-1 is a flip), and xslant and yslant for slanting. If these transformation and those that I have not mentioned are not sufficient, the cm option allows you to apply an arbitrary transformation matrix. Karl's students, by the way, do not know what a transformation matrix is.

1.20 Repeating Things: For-Loops

Karl's next aim is to add little ticks on the axes at positions −1, −1/2, 1/2, and 1. For this, it would be nice to use some kind of "loop," especially since he wishes to do the same thing at each of these positions. There are different packages for doing this. LATEX has its own internal command for this, pstricks comes along with the powerful \multido command. All of these can be used together with Ti*k*Z, so if you are familiar with them, feel free to use them. Ti*k*Z introduces yet another command, called `\foreach`, which I introduced since I could never remember the syntax of the other packages. `\foreach` is defined in the package `pgffor` and can be used independently Ti*k*Z, but Ti*k*Z includes it automatically.

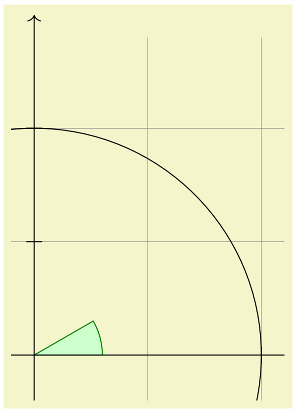
In its basic form, the `\foreach` command is easy to use:

$x = 1$, $x = 2$, $x = 3$,

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

The general syntax is `\foreach` ⟨*variable*⟩ `in {`⟨*list of values*⟩`}` ⟨*commands*⟩. Inside the ⟨*commands*⟩, the ⟨*variable*⟩ will be assigned to the different values. If the ⟨*commands*⟩ do not start with a brace, everything up to the next semicolon is used as ⟨*commands*⟩.

For Karl and the ticks on the axes, he could use the following code:

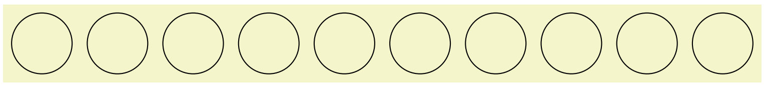


```
\begin{tikzpicture}[scale=3]
  \clip (-0.1,-0.2) rectangle (1.1,1.51);
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
      arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0);
  \draw[->]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

As a matter of fact, there are many different ways of creating the ticks. For example, Karl could have put the `\draw ...;` inside curly braces. He could also have used, say,

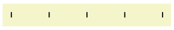
```
\foreach \x in {-1,-0.5,1}
  \draw[xshift=\x cm] (0pt,-1pt) -- (0pt,1pt);
```

Karl is curious what would happen in a more complicated situation where there are, say, 20 ticks. It seems bothersome to explicitly mention all these numbers in the set for `\foreach`. Indeed, it is possible to use `...` inside the `\foreach` statement to iterate over a large number of values (which must, however, be dimensionless real numbers) as in the following example:



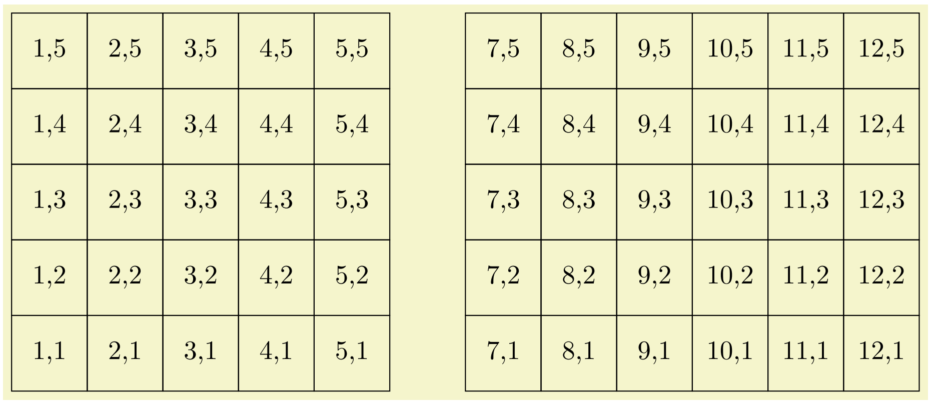
```
\tikz \foreach \x in {1,...,10}
        \draw (\x,0) circle (0.4cm);
```

If you provide *two* numbers before the `...`, the `\foreach` statement will use their difference for the stepping:



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

We can also nest loops to create interesting effects:



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

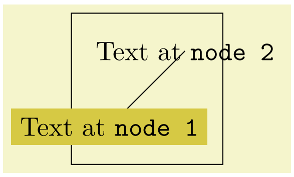
```
\begin{tikzpicture}
  \foreach \x in {1,2,...,5,7,8,...,12}
    \foreach \y in {1,...,5}
    {
      \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
      \draw (\x,\y) node{\x,\y};
    }
\end{tikzpicture}
```

The `\foreach` statement can do even trickier stuff, but the above gives the idea.

1.21 Adding Text

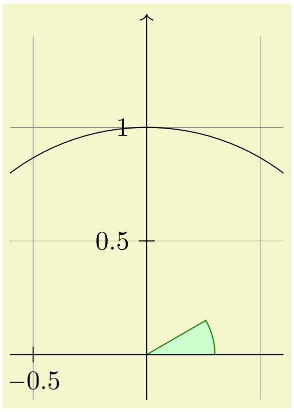
Karl is, by now, quite satisfied with the picture. However, the most important parts, namely the labels, are still missing!

Ti*k*Z offers an easy-to-use and powerful system for adding text and, more generally, complex shapes to a picture at specific positions. The basic idea is the following: When Ti*k*Z is constructing a path and encounters the keyword `node` in the middle of a path, it reads a *node specification*. The keyword `node` is typically followed by some options and then some text between curly braces. This text is put inside a normal TEX box (if the node specification directly follows a coordinate, which is usually the case, Ti*k*Z is able to perform some magic so that it is even possible to use verbatim text inside the boxes) and then placed at the current position, that is, at the last specified position (possibly shifted a bit, according to the given options). However, all nodes are drawn only after the path has been completely drawn/filled/shaded/clipped/whatever.



```
\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}
```

Obviously, Karl would not only like to place nodes *on* the last specified position, but also to the left or the right of these positions. For this, every node object that you put in your picture is equipped with several *anchors*. For example, the `north` anchor is in the middle at the upper end of the shape, the `south` anchor is at the bottom and the `north east` anchor is in the upper right corner. When you give the option `anchor=`*`north`*, the text will be placed such that this northern anchor will lie on the current position and the text is, thus, below the current position. Karl uses this to draw the ticks as follows:

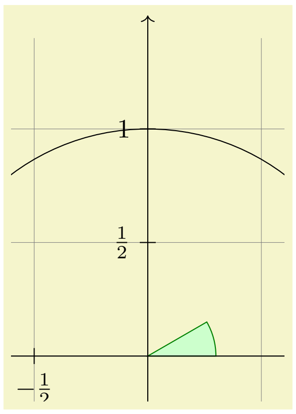


```
\begin{tikzpicture}[scale=3]
  \clip (-0.6,-0.2) rectangle (0.6,1.51);
  \draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
  \filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
    arc [start angle=0, end angle=30, radius=3mm] -- cycle;
  \draw[->] (-1.5,0) -- (1.5,0);   \draw[->]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

This is quite nice, already. Using these anchors, Karl can now add most of the other text elements. However, Karl thinks that, though “correct,” it is quite counter-intuitive that in order to place something *below* a given point, he has to use the *north* anchor. For this reason, there is an option called `below`, which does the same as `anchor=`*`north`*. Similarly, `above right` does the same as `anchor=`*`south west`*. In addition, `below` takes an optional dimension argument. If given, the shape will additionally be shifted downwards by the given amount. So, `below=`*`1pt`* can be used to put a text label below some point and, additionally shift it 1pt downwards.

Karl is not quite satisfied with the ticks. He would like to have 1/2 or $\frac{1}{2}$ shown instead of 0.5, partly to show off the nice capabilities of TEX and Ti*k*Z, partly because for positions like 1/3 or π it is certainly very much preferable to have the “mathematical” tick there instead of just the “numeric” tick. His students, on the other hand, prefer 0.5 over 1/2 since they are not too fond of fractions in general.

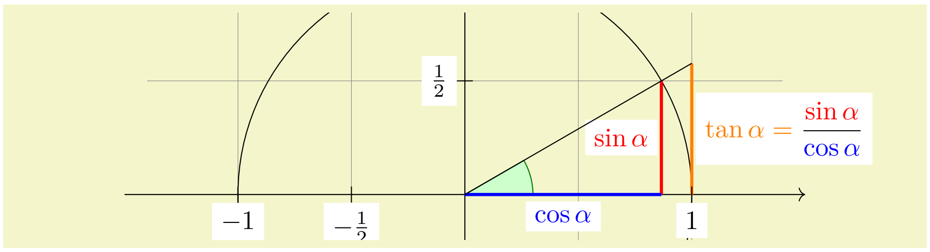
Karl now faces a problem: For the `\foreach` statement, the position `\x` should still be given as `0.5` since Ti*k*Z will not know where `\frac{1}{2}` is supposed to be. On the other hand, the typeset text should really be `\frac{1}{2}`. To solve this problem, `\foreach` offers a special syntax: Instead of having one variable `\x`, Karl can specify two (or even more) variables separated by a slash as in `\x / \xtext`. Then, the elements in the set over which `\foreach` iterates must also be of the form ⟨*first*⟩/⟨*second*⟩. In each iteration, `\x` will be set to ⟨*first*⟩ and `\xtext` will be set to ⟨*second*⟩. If no ⟨*second*⟩ is given, the ⟨*first*⟩ will be used again. So, here is the new code for the ticks:



```
\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=east] {$\ytext$};
\end{tikzpicture}
```

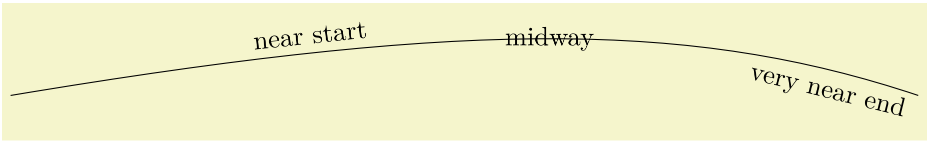
Karl is quite pleased with the result, but his son points out that this is still not perfectly satisfactory: The grid and the circle interfere with the numbers and decrease their legibility. Karl is not very concerned by this (his students do not even notice), but his son insists that there is an easy solution: Karl can add the `[fill=white]` option to fill out the background of the text shape with a white color.

The next thing Karl wants to do is to add the labels like $\sin\alpha$. For this, he would like to place a label "in the middle of the line." To do so, instead of specifying the label `node {$\sin\alpha$}` directly after one of the endpoints of the line (which would place the label at that endpoint), Karl can give the label directly after the `--`, before the coordinate. By default, this places the label in the middle of the line, but the `pos=` options can be used to modify this. Also, options like `near start` and `near end` can be used to modify this position:



```
\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=east,fill=white] {$\ytext$};
\end{tikzpicture}
```

You can also position labels on curves and, by adding the `sloped` option, have them rotated such that they match the line's slope. Here is an example:



```
\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}
```

It remains to draw the explanatory text at the right of the picture. The main difficulty here lies in limiting the width of the text "label," which is quite long, so that line breaking is used. Fortunately, Karl can use the option `text width=6cm` to get the desired effect. So, here is the full code:

```
\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]
    {
      The {\color{anglecolor} angle $\alpha$} is $30^\circ$ in the
      example ($\pi/6$ in radians). The {\color{sincolor}sine of
        $\alpha$}, which is the height of the red line, is
      \[
      {\color{sincolor} \sin \alpha} = 1/2.
      \]
      By the Theorem of Pythagoras ...
    };
\end{tikzpicture}
```

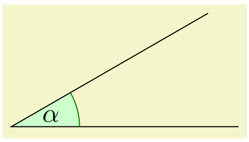
1.22 Pics: The Angle Revisited

Karl expects that the code of certain parts of the picture he created might be so useful that he might wish to reuse them in the future. A natural thing to do is to create TEX macros that store the code he wishes to reuse. However, Ti*k*Z offers another way that is integrated directly into its parser: pics!

A "pic" is "not quite a full picture," hence the short name. The idea is that a pic is simply some code that you can add to a picture at different places using the `pic` command whose syntax is almost identical to the `node` command. The main difference is that instead of specifying some text in curly braces that should be shown, you specify the name of a predefined picture that should be shown.

Defining new pics is easy enough, see Section **??**, but right now we just want to use one such predefined pic: the `angle` pic. As the name suggests, it is a small drawing of an angle consisting of a little wedge and an arc together with some text (Karl needs to load the `angle` library and the `quotes` for the following examples). What makes this pic useful is the fact that the size of the wedge will be computed automatically.

The `angle` pic draws an angle between the two lines BA and BC, where A, B, and C are three coordinates. In our case, B is the origin, A is somewhere on the x-axis and C is somewhere on a line at 30°.



```
\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}
```

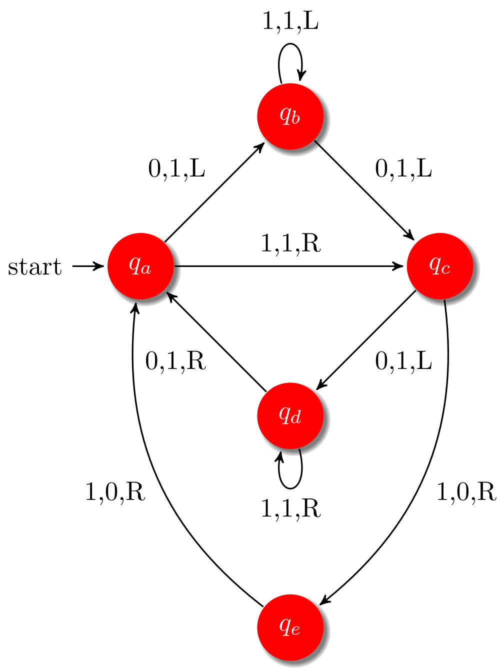
Let us see, what is happening here. First we have specified three *coordinates* using the `\coordinate` command. It allows us to name a specific coordinate in the picture. Then comes something that starts as a normal `\draw`, but then comes the `pic` command. This command gets lots of options and, in curly braces, comes the most important point: We specify that we want to add an `angle` pic and this angle should be between the points we named `A`, `B`, and `C` (we could use other names). Note that the text that we want to be shown in the pic is specified in quotes inside the options of the `pic`, not inside the curly braces.

To learn more about pics, please see Section **??**.

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。



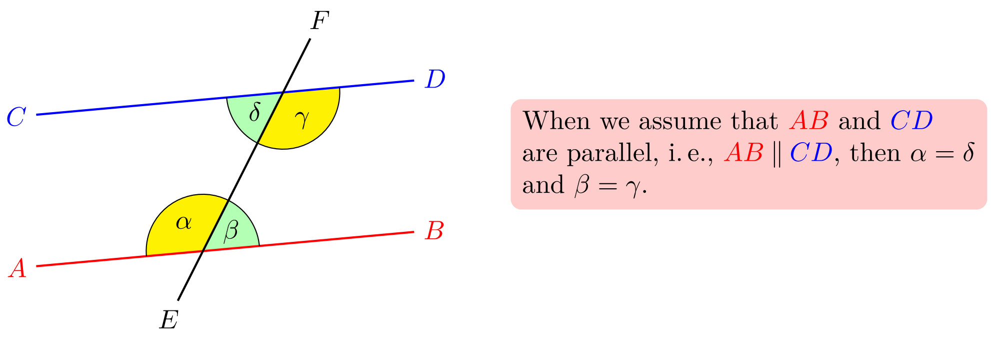
The current candidate for the busy beaver for five states. It is presumed that this Turing machine writes a maximum number of 1's before halting among all Turing machines with five states and the tape alphabet $\{0, 1\}$. Proving this conjecture is an open research problem. 中文测试

```
\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

Ti*k*Z ist *kein* Zeichenprogramm

by Till Tantau



When we assume that AB and CD are parallel, i. e., $AB \mathbin{\|} CD$, 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.

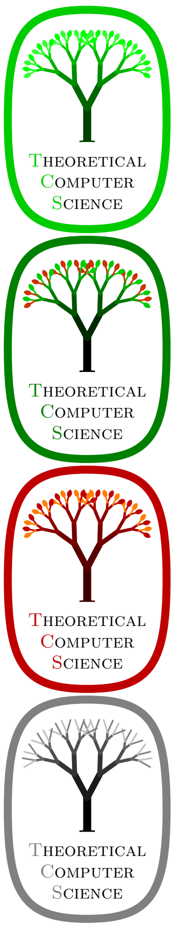
You need to use LuaTeX to typeset this part of the manual (and, also, to use algorithmic graph drawing).

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.



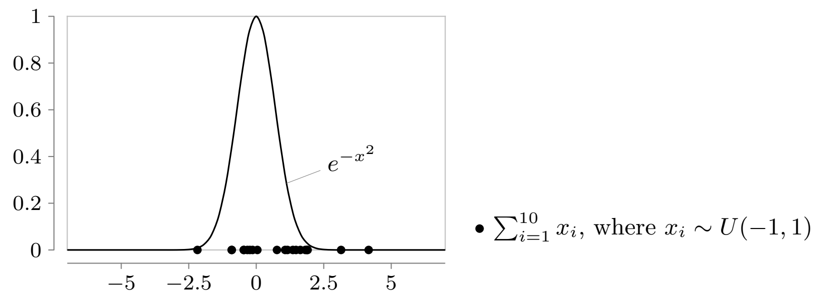
Theoretical Computer Science
Theoretical Computer Science
Theoretical Computer Science
Theoretical Computer Science

```
\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



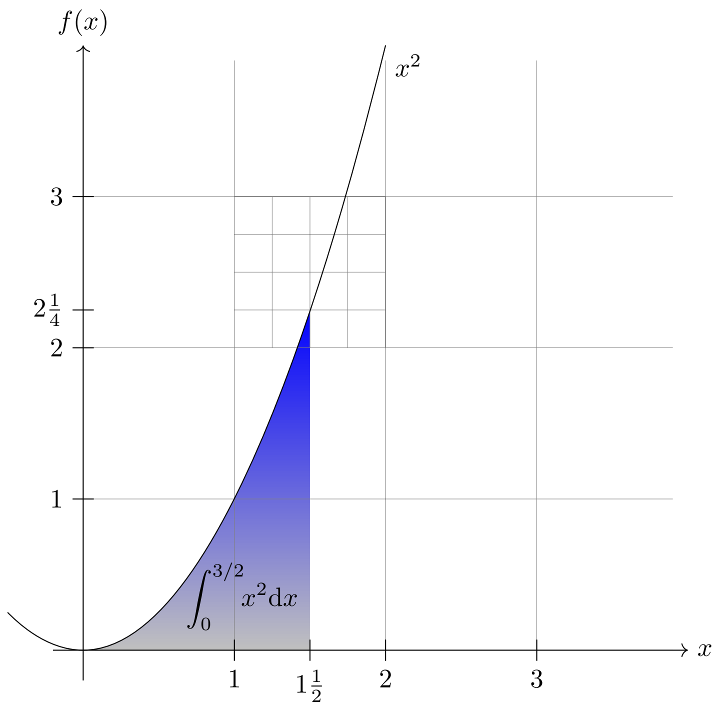
```
\tikz \datavisualization [scientific axes=clean]
[
  visualize as smooth line=Gaussian,
  Gaussian={pin in data={text={$e^{-x^2}$},when=x is 1}}
]
data [format=function] {
  var x : interval [-7:7] samples 51;
  func y = exp(-\value x*\value x);
}
[
  visualize as scatter,
  legend={south east outside},
  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

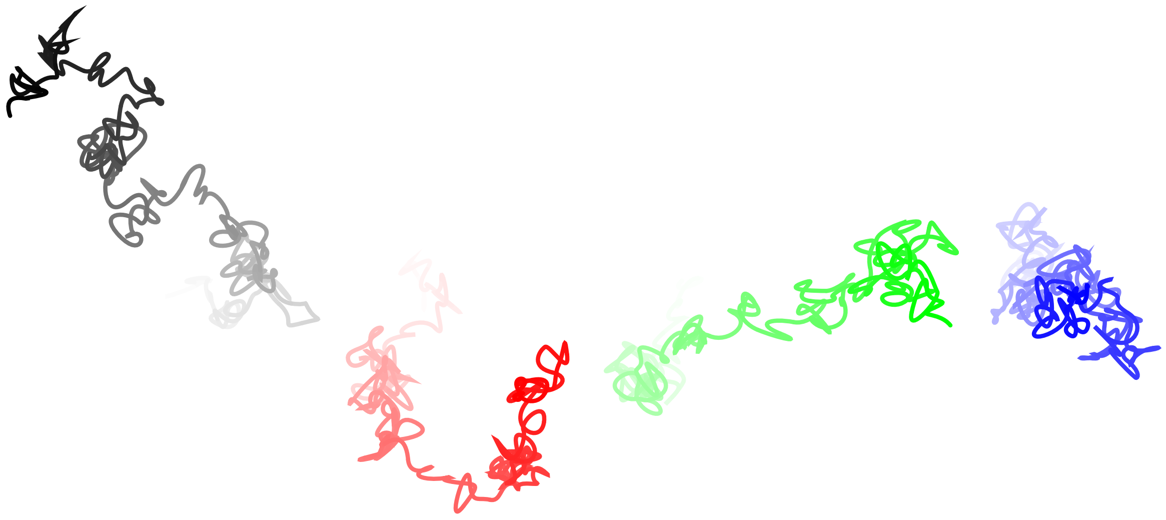
Mathematical and Object-Oriented Engines

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.

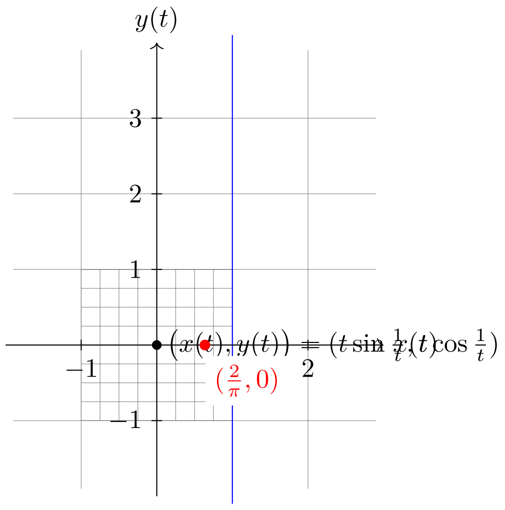
The job of the object-oriented engine is to support simple object-oriented programming in TEX. It allows the definition of *classes* (without inheritance), *methods*, *attributes* and *objects*.



```
\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


$y(t)$
3
2
1
−1
2
−1
$\bigl(x(t),y(t)\bigr) = (t\sin \frac{1}{t}, t\cos \frac{1}{t})$
$x(t)$
$(\frac{2}{\pi},0)$

```
\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}
```

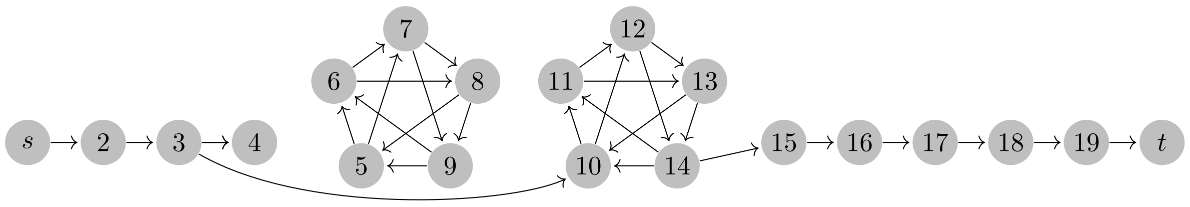
Part X
The System Layer

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.

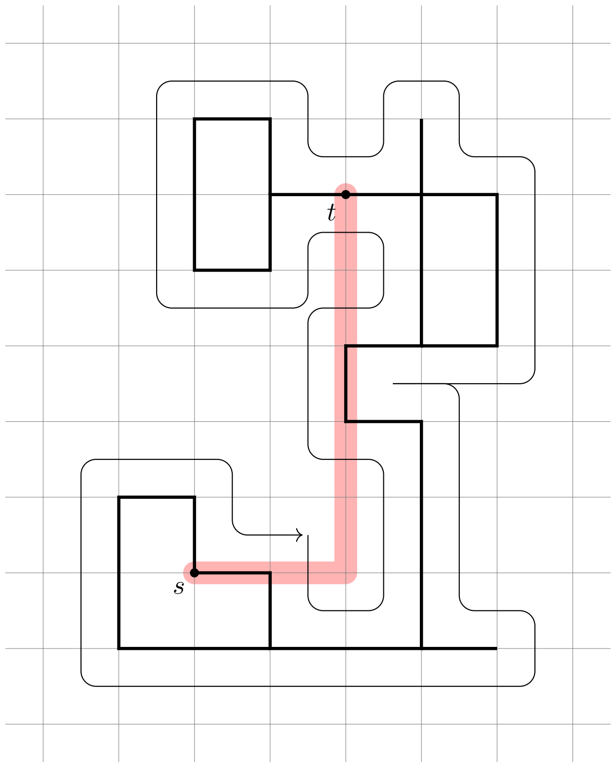
Unless you intend to port PGF to another driver or unless you intend to write your own optimized frontend, you need not read this part.

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}
```