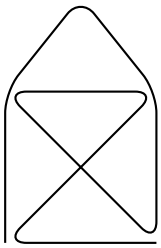
Part I
教程和指导

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 Tutorial: Euclid's Amber Version of the *Elements*

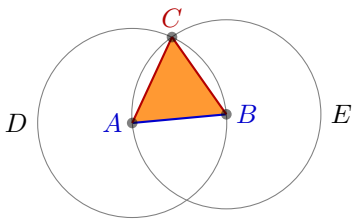
在这第三篇教程中，我们看看如何用 Ti*k*Z 实现几何作图。

Euclid 最近在忙着写一套新书，书名暂定为“几何原本”（Euclid 不是很确定这个书名是否能把主旨传达给后人，但是他打算在交付给出版商之前修改一下书名）。到目前为止，他用莎草纸写字画图，但是他的出版商突然要求他必须提交电子版。Euclid 跟出版商争辩道，电子产品要在几千年后才会发明出来，然而出版商告诉他，莎草纸不再是尖端技术了，他必须得跟上现代工具的步伐。

Euclid 略带不满，准备把他在莎草纸写的东西转到琥珀上，该部分题为“第 I 卷，命题 1”。

1.1 第 I 卷，命题 1

他在莎草纸上的图是这样的：[1] [2] [3]



命题 1
在一个已知有限直线上做一个等边三角形.

设 AB 是已知有限直线. 那么要求在线段 AB 上作一个等边三角形.
以 A 为心，且以 AB 为距离画圆 BCD；再以 B 为心，且以 BA 为距离画圆 ACE；由两圆的交点 C 到 A、B 连线 CA、CB.
因为，点 A 是圆 CDB 的圆心，AC 等于 AB. 又点 B 是圆 CAE 的圆心，BC 等于 BA. 但是，已经证明了 CA 等于 AB；所以线段 CA、CB 都等于 AB. 而且等于同量的量彼此相等. 三条线段 CA、AB、BC 彼此相等. 所以三角形 ABC 是等边的，即在已知有限直线 AB 上作出了这个三角形.

让我们看看，Euclid 怎么把这张图变成 Ti*k*Z 代码。

1.1.1 设置环境

如之前教程所述，Euclid 需要载入 Ti*k*Z 以及一些库，这些库包括 `calc`、`intersections`、`through` 和 `backgrounds`。根据格式不同，Euclid 要在序言区加上下列语句之一：

```
% 对于 LaTeX:
\usepackage{tikz}
\usetikzlibrary{calc,intersections,through,backgrounds}
```

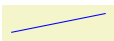
```
% 对于 plain TeX:
\input tikz.tex
\usetikzlibrary{calc,intersections,through,backgrounds}
```

```
% 对于 ConTeXt:
\usemodule[tikz]
\usetikzlibrary[calc,intersections,through,backgrounds]
```

1.1.2 线 *AB*

图中 Euclid 想最先画的部分是线 AB。很简单，`\draw (0,0) -- (2,1);` 就能搞定。但是，Euclid 不想在 A 和 B 后面加上 $(0,0)$ 和 $(2,1)$，他想只写上 `A` 和 `B`。事实上，他的书的主旨是，点 A 和 B 可以是任意的，其他的点（比如 C）可以根据它们的位置构造出来，Euclid 不需要直接写出 C 的坐标。

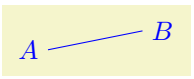
所以 Euclid 用 `\coordinate` 命令定义两个坐标：



```
\begin{tikzpicture}
  \coordinate (A) at (0,0);
  \coordinate (B) at (1.25,0.25);

  \draw[blue] (A) -- (B);
\end{tikzpicture}
```

[1] 该段文本摘自 David E. Joyce 漂亮的交互式欧几里得几何原本，可以在他 Clark 大学的网站上找到。
[2] 该段文本原文链接：https://mathcs.clarku.edu/~djoyce/java/elements/bookI/propI1.html
[3] 该段译文参考：欧几里得 著，兰纪正、朱恩宽 译. 欧几里得 • 几何原本. 陕西科学技术出版社, 2003. ISBN 9787536903579.

非常简单。这里少了坐标的标签，Euclid 不想标在点上面，而是在点旁边。他决定用 label 选项：




```
\begin{tikzpicture}
  \coordinate [label=left:\textcolor{blue}{$A$}]  (A) at (0,0);
  \coordinate [label=right:\textcolor{blue}{$B$}] (B) at (1.25,0.25);

  \draw[blue] (A) -- (B);
\end{tikzpicture}
```

Euclid 这时在想，要是点 A 和 B 可以“随机”一点就更好了。关于这些点的位置，Euclid 和读者都不能犯下“想当然”的错误。Euclid 很高兴地了解到，Ti*k*Z 里有一个 rand 函数可以满足要求：它能生成一个 -1 到 1 之间的数。既然 Ti*k*Z 能做些数学运算，Euclid 就可以把点坐标写成下面这样：

```
\coordinate [...] (A) at (0+0.1*rand,0+0.1*rand);
\coordinate [...] (B) at (1.25+0.1*rand,0.25+0.1*rand);
```

这个效果不错，但是 Euclid 并不是很满意，因为他想让“主坐标” $(0,0)$ 和 $(1.25,0.25)$ 同扰动 $0.1(rand, rand)$ “分离开来”。也就是说，他希望将坐标 A 指定成这样，“点 $(0,0)$ 加上向量 $(rand, rand)$ 的十分之一”。

实际上，calc 库来能让他做这类运算。载入该库后，你可以用特殊的坐标，始于 ($ 止于 $)，而不只是 (和)。你可以对这些特殊的坐标进行线性组合。（注意到 $ 符号只是表示开始“运算”，而不是输入数学公式。）

关于坐标的新代码如下：

```
\coordinate [...] (A) at ($ (0,0) + .1*(rand,rand) $);
\coordinate [...] (B) at ($ (1.25,0.25) + .1*(rand,rand) $);
```

注意，如果在这类计算中，一个坐标包含小数（比如 .1），那么你必须在坐标的左圆括号之前加上 *。这些运算可以嵌套。

1.1.3 围绕点 *A* 的圆

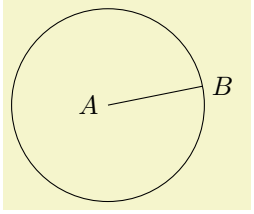
第一个棘手的构造是围绕点 A 的圆。我们之后会介绍一个非常简单的方法，但是首先让我们用“困难的”方式实现它。

思路如下：我们画一个围绕点 A 的圆，其半径由线段 AB 的长度决定。困难在于计算这条线段的长度。

可以通过两个点子解决这个问题：第一个，我们可以用 ($ (A) - (B) $) 表示 A 和 B 相差的向量。我们需要的是这个向量的长度。第二个，给定两个数 x 和 y，我们可以在数学表达式中写 veclen(x,y)，这会返回数值 $\sqrt{x^2+y^2}$，也就是想要的长度。

剩下的唯一问题就是如何获得向量 AB 的 x 轴和 y 轴坐标。为此我们需要引入一个新概念：let 操作。let 操作可以在路径中这样一些地方插入，比如直线指向的位置或者移动的目标位置。let 操作的效果是计算一些坐标并将其结果赋给特定的宏，这些宏方便了对坐标的 x 和 y 值的访问。

Euclid 可以这样写：




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \draw (A) let
              \p1 = ($ (B) - (A) $)
            in
              circle ({veclen(\x1,\y1)});
\end{tikzpicture}
```

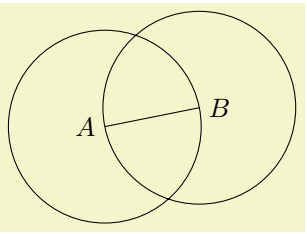
let 操作的每一次赋值都从 \p 开始，通常跟着一个 ⟨数字⟩，接着是一个等号和一个坐标。计算好坐标后，结果保存在内部，之后你就可以用下面的表达式：

1. \x⟨数字⟩ 得到对应点的 x 坐标。
2. \y⟨数字⟩ 得到对应点的 y 坐标。
3. \p⟨数字⟩ 得到 \x⟨数字⟩,\y⟨数字⟩。

You can have multiple assignments in a let operation, just separate them with commas. In later assignments you can already use the results of earlier assignments.

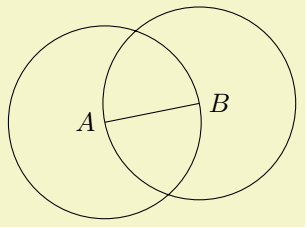
Note that `\p1` is not a coordinate in the usual sense. Rather, it just expands to a string like `10pt,20pt`. So, you cannot write, for instance, `(\p1.center)` since this would just expand to `(10pt,20pt.center)`, which makes no sense.

Next, we want to draw both circles at the same time. Each time the radius is `veclen(\x1,\y1)`. It seems natural to compute this radius only once. For this, we can also use a let operation: Instead of writing `\p1 = ...`, we write `\n2 = ...`. Here, "n" stands for "number" (while "p" stands for "point"). The assignment of a number should be followed by a number in curly braces.




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \draw let \p1 = ($ (B) - (A) $),
            \n2 = {veclen(\x1,\y1)}
        in
          (A) circle (\n2)
          (B) circle (\n2);
\end{tikzpicture}
```

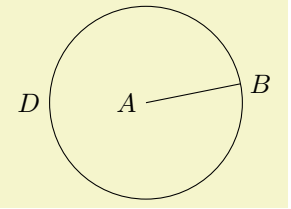
In the above example, you may wonder, what `\n1` would yield? The answer is that it would be undefined – the `\p`, `\x`, and `\y` macros refer to the same logical point, while the `\n` macro has "its own namespace." We could even have replaced `\n2` in the example by `\n1` and it would still work. Indeed, the digits following these macros are just normal TEX parameters. We could also use a longer name, but then we have to use curly braces:




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \draw let \p1        = ($ (B) - (A) $),
            \n{radius} = {veclen(\x1,\y1)}
        in
          (A) circle (\n{radius})
          (B) circle (\n{radius});
\end{tikzpicture}
```

At the beginning of this section it was promised that there is an easier way to create the desired circle. The trick is to use the `through` library. As the name suggests, it contains code for creating shapes that go through a given point.

The option that we are looking for is `circle through`. This option is given to a *node* and has the following effects: First, it causes the node's inner and outer separations to be set to zero. Then it sets the shape of the node to `circle`. Finally, it sets the radius of the node such that it goes through the parameter given to `circle through`. This radius is computed in essentially the same way as above.



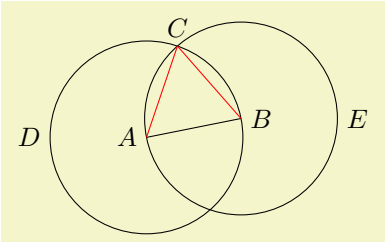

```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \node [draw,circle through=(B),label=left:$D$] at (A) {};
\end{tikzpicture}
```

1.1.4 The Intersection of the Circles

Euclid can now draw the line and the circles. The final problem is to compute the intersection of the two circles. This computation is a bit involved if you want to do it "by hand." Fortunately, the intersection library allows us to compute the intersection of arbitrary paths.

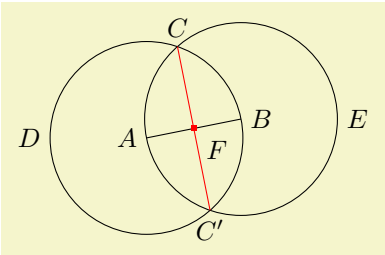
The idea is simple: First, you "name" two paths using the `name path` option. Then, at some later point, you can use the option `name intersections`, which creates coordinates called `intersection-1`,

`intersection-2`, and so on at all intersections of the paths. Euclid assigns the names `D` and `E` to the paths of the two circles (which happen to be the same names as the nodes themselves, but nodes and their paths live in different "namespaces").




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \node (D) [name path=D,draw,circle through=(B),label=left:$D$]  at (A) {};
  \node (E) [name path=E,draw,circle through=(A),label=right:$E$] at (B) {};

  % Name the coordinates, but do not draw anything:
  \path [name intersections={of=D and E}];

  \coordinate [label=above:$C$] (C) at (intersection-1);

  \draw [red] (A) -- (C);
  \draw [red] (B) -- (C);
\end{tikzpicture}
```

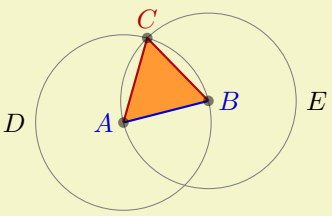
It turns out that this can be further shortened: The `name intersections` takes an optional argument `by`, which lets you specify names for the coordinates and options for them. This creates more compact code. Although Euclid does not need it for the current picture, it is just a small step to computing the bisection of the line AB:




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw [name path=A--B] (A) -- (B);

  \node (D) [name path=D,draw,circle through=(B),label=left:$D$]  at (A) {};
  \node (E) [name path=E,draw,circle through=(A),label=right:$E$] at (B) {};

  \path [name intersections={of=D and E, by={[label=above:$C$]C, [label=below:$C'$]C'}}];

  \draw [name path=C--C',red] (C) -- (C');

  \path [name intersections={of=A--B and C--C',by=F}];
  \node [fill=red,inner sep=1pt,label=-45:$F$] at (F) {};
\end{tikzpicture}
```

1.1.5 The Complete Code

Back to Euclid's code. He introduces a few macros to make life simpler, like a `\A` macro for typesetting a blue A. He also uses the `background` layer for drawing the triangle behind everything at the end.



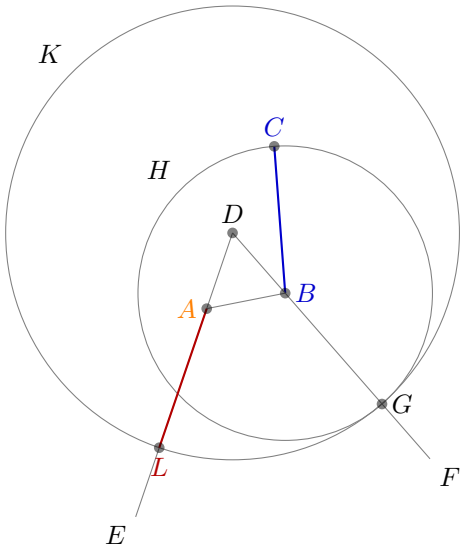
Proposition I
To construct an equilateral triangle on a given finite straight line.

Let AB be the given finite straight line. . . .

```
\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
  \def\A{\textcolor{input}{$A$}}     \def\B{\textcolor{input}{$B$}}
  \def\C{\textcolor{output}{$C$}}    \def\D{$D$}
  \def\E{$E$}

  \colorlet{input}{blue!80!black}    \colorlet{output}{red!70!black}
  \colorlet{triangle}{orange}

  \coordinate [label=left:\A]  (A) at ($ (0,0) + .1*(rand,rand) $);
  \coordinate [label=right:\B] (B) at ($ (1.25,0.25) + .1*(rand,rand) $);

  \draw [input] (A) -- (B);

  \node [name path=D,help lines,draw,label=left:\D]   (D) at (A) [circle through=(B)] {};
  \node [name path=E,help lines,draw,label=right:\E]  (E) at (B) [circle through=(A)] {};

  \path [name intersections={of=D and E,by={[label=above:\C]C}}];

  \draw [output] (A) -- (C) -- (B);

  \foreach \point in {A,B,C}
    \fill [black,opacity=.5] (\point) circle (2pt);

  \begin{pgfonlayer}{background}
    \fill[triangle!80] (A) -- (C) -- (B) -- cycle;
  \end{pgfonlayer}

  \node [below right, text width=10cm,align=justify] at (4,3) {
    \small\textbf{Proposition I}\par
    \emph{To construct an \textcolor{triangle}{equilateral triangle}
      on a given \textcolor{input}{finite straight line}.}
    \par\vskip1em
    Let \A\B\ be the given \textcolor{input}{finite straight line}.  \dots
  };
\end{tikzpicture}
```

1.2 Book I, Proposition II

The second proposition in the Elements is the following:



Proposition II

To place a straight line equal to a given straight line with one end at a given point.

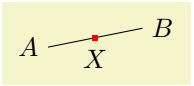
Let A be the given point, and BC the given straight line. It is required to place a straight line equal to the given straight line BC with one end at the point A.
Join the straight line AB from the point A to the point B, and construct the equilateral triangle DAB on it.
Produce the straight lines AE and BF in a straight line with DA and DB. Describe the circle CGH with center B and radius BC, and again, describe the circle GKL with center D and radius DG.
Since the point B is the center of the circle CGH, therefore BC equals BG. Again, since the point D is the center of the circle GKL, therefore DL equals DG. And in these DA equals DB, therefore the remainder AL equals the remainder BG. But BC was also proved equal to BG, therefore each of the straight lines AL and BC equals BG. And things which equal the same thing also equal one another, therefore AL also equals BC.
Therefore the straight line AL equal to the given straight line BC has been placed with one end at the given point A.

1.2.1 Using Partway Calculations for the Construction of D

Euclid's construction starts with "referencing" Proposition I for the construction of the point D. Now, while we could simply repeat the construction, it seems a bit bothersome that one has to draw all these circles and do all these complicated constructions.

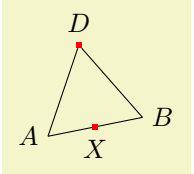
For this reason, Ti*k*Z supports some simplifications. First, there is a simple syntax for computing a point that is "partway" on a line from p to q: You place these two points in a coordinate calculation – remember, they start with `($` and end with `$)` – and then combine them using `!`⟨*part*⟩`!`. A ⟨*part*⟩ of `0` refers to the *first* coordinate, a ⟨*part*⟩ of `1` refers to the second coordinate, and a value in between refers to a point on the line from p to q. Thus, the syntax is similar to the `xcolor` syntax for mixing colors.

Here is the computation of the point in the middle of the line AB:



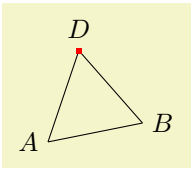

```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);
  \node [fill=red,inner sep=1pt,label=below:$X$] (X) at ($ (A)!.5!(B) $) {};
\end{tikzpicture}
```

The computation of the point D in Euclid's second proposition is a bit more complicated. It can be expressed as follows: Consider the line from X to B. Suppose we rotate this line around X for 90° and then stretch it by a factor of $\sin(60°) \cdot 2$. This yields the desired point D. We can do the stretching using the partway modifier above, for the rotation we need a new modifier: the rotation modifier. The idea is that the second coordinate in a partway computation can be prefixed by an angle. Then the partway point is computed normally (as if no angle were given), but the resulting point is rotated by this angle around the first point.




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);
  \node [fill=red,inner sep=1pt,label=below:$X$] (X) at ($ (A)!.5!(B) $) {};
  \node [fill=red,inner sep=1pt,label=above:$D$] (D) at
    ($ (X) ! {sin(60)*2} ! 90:(B) $) {};
  \draw (A) -- (D) -- (B);
\end{tikzpicture}
```

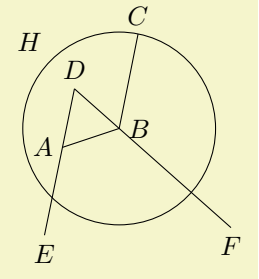
Finally, it is not necessary to explicitly name the point X. Rather, again like in the `xcolor` package, it is possible to chain partway modifiers:




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);
  \node [fill=red,inner sep=1pt,label=above:$D$] (D) at
    ($ (A) ! .5 ! (B) ! {sin(60)*2} ! 90:(B) $) {};
  \draw (A) -- (D) -- (B);
\end{tikzpicture}
```

1.2.2 Intersecting a Line and a Circle

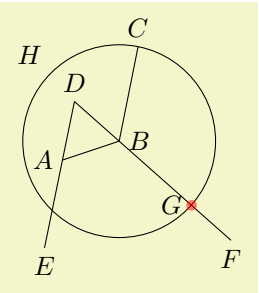
The next step in the construction is to draw a circle around B through C, which is easy enough to do using the `circle through` option. Extending the lines DA and DB can be done using partway calculations, but this time with a part value outside the range $[0, 1]$:




```
\begin{tikzpicture}
  \coordinate [label=left:$A$]  (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (0.75,0.25);
  \coordinate [label=above:$C$] (C) at (1,1.5);
  \draw (A) -- (B) -- (C);
  \coordinate [label=above:$D$] (D) at
    ($ (A) ! .5 ! (B) ! {sin(60)*2} ! 90:(B) $) {};
  \node (H) [label=135:$H$,draw,circle through=(C)] at (B) {};
  \draw (D) -- ($ (D) ! 3.5 ! (B) $) coordinate [label=below:$F$] (F);
  \draw (D) -- ($ (D) ! 2.5 ! (A) $) coordinate [label=below:$E$] (E);
\end{tikzpicture}
```

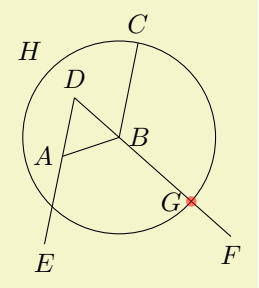
We now face the problem of finding the point G, which is the intersection of the line BF and the circle H. One way is to use yet another variant of the partway computation: Normally, a partway computation has the form $\langle p\rangle\,!\,\langle factor\rangle\,!\,\langle q\rangle$, resulting in the point $(1-\langle factor\rangle)\langle p\rangle + \langle factor\rangle\langle q\rangle$. Alternatively, instead of $\langle factor\rangle$ you can also use a $\langle dimension\rangle$ between the points. In this case, you get the point that is $\langle dimension\rangle$ away from $\langle p\rangle$ on the straight line to $\langle q\rangle$.

We know that the point G is on the way from B to F. The distance is given by the radius of the circle H. Here is the code for computing H:



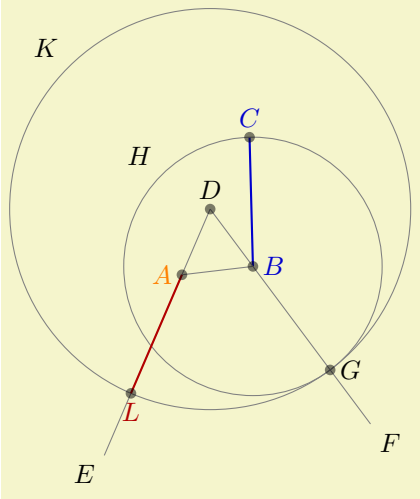

```
  \node (H) [label=135:$H$,draw,circle through=(C)] at (B) {};
  \path let \p1 = ($ (B) - (C) $) in
    coordinate [label=left:$G$] (G) at ($ (B) ! veclen(\x1,\y1) ! (F) $);
  \fill[red,opacity=.5] (G) circle (2pt);
```

However, there is a simpler way: We can simply name the path of the circle and of the line in question and then use `name intersections` to compute the intersections.




```
  \node (H) [name path=H,label=135:$H$,draw,circle through=(C)] at (B) {};
  \path [name path=B--F] (B) -- (F);
  \path [name intersections={of=H and B--F,by={[label=left:$G$]G}}];
  \fill[red,opacity=.5] (G) circle (2pt);
```

1.2.3 The Complete Code

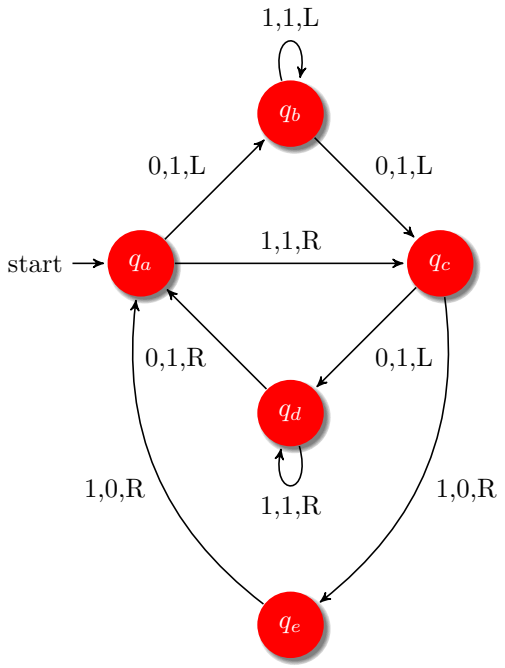



```
\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
  \def\A{\textcolor{orange}{$A$}}   \def\B{\textcolor{input}{$B$}}
  \def\C{\textcolor{input}{$C$}}    \def\D{$D$}
  \def\E{$E$}                       \def\F{$F$}
  \def\G{$G$}                       \def\H{$H$}
  \def\K{$K$}                       \def\L{\textcolor{output}{$L$}}

  \colorlet{input}{blue!80!black}    \colorlet{output}{red!70!black}

  \coordinate [label=left:\A]  (A) at ($ (0,0) + .1*(rand,rand) $);
  \coordinate [label=right:\B] (B) at ($ (1,0.2) + .1*(rand,rand) $);
  \coordinate [label=above:\C] (C) at ($ (1,2) + .1*(rand,rand) $);

  \draw [input] (B) -- (C);
  \draw [help lines] (A) -- (B);

  \coordinate [label=above:\D] (D) at ($ (A)!.5!(B) ! {sin(60)*2} ! 90:(B) $);

  \draw [help lines] (D) -- ($ (D)!3.75!(A) $) coordinate [label=-135:\E] (E);
  \draw [help lines] (D) -- ($ (D)!3.75!(B) $) coordinate [label=-45:\F] (F);

  \node (H) at (B) [name path=H,help lines,circle through=(C),draw,label=135:\H] {};
  \path [name path=B--F] (B) -- (F);
  \path [name intersections={of=H and B--F,by={[label=right:\G]G}}];

  \node (K) at (D) [name path=K,help lines,circle through=(G),draw,label=135:\K] {};
  \path [name path=A--E] (A) -- (E);
  \path [name intersections={of=K and A--E,by={[label=below:\L]L}}];

  \draw [output] (A) -- (L);

  \foreach \point in {A,B,C,D,G,L}
    \fill [black,opacity=.5] (\point) circle (2pt);

  % \node ...
\end{tikzpicture}
```

Part II
安装和配置

by Till Tantau

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

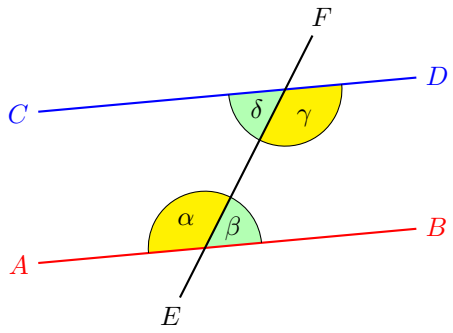


The current candidate for the busy beaver for five states. It is presumed that this Turing machine writes a maximum number of 1's before halting among all Turing machines with five states and the tape alphabet $\{0, 1\}$. Proving this conjecture is an open research problem. 中文测试

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

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

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

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

Part III
TikZ ist *kein* Zeichenprogramm

by Till Tantau



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

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

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

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

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

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

Part IV
Graph Drawing

by Till Tantau et al.

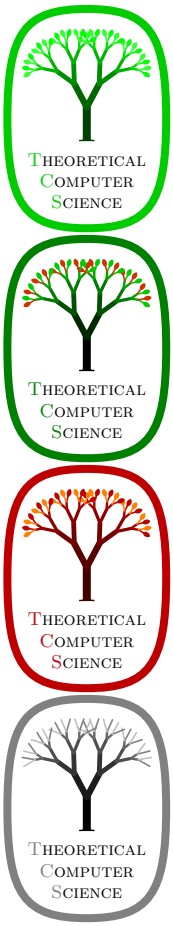
Graph drawing algorithms do the tough work of computing a layout of a graph for you. Ti*k*Z comes with powerful such algorithms, but you can also implement new algorithms in the Lua programming language.

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

Part V
Libraries

by Till Tantau

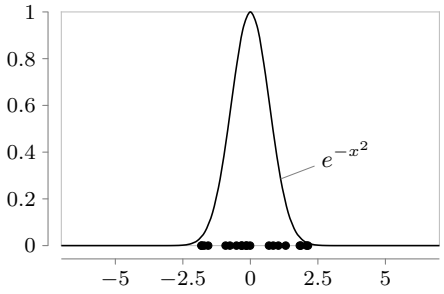
In this part the library packages are documented. They provide additional predefined graphic objects like new arrow heads or new plot marks, but sometimes also extensions of the basic PGF or Ti*k*Z system. The libraries are not loaded by default since many users will not need them.




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

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

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

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

Part VI
Data Visualization

by Till Tantau



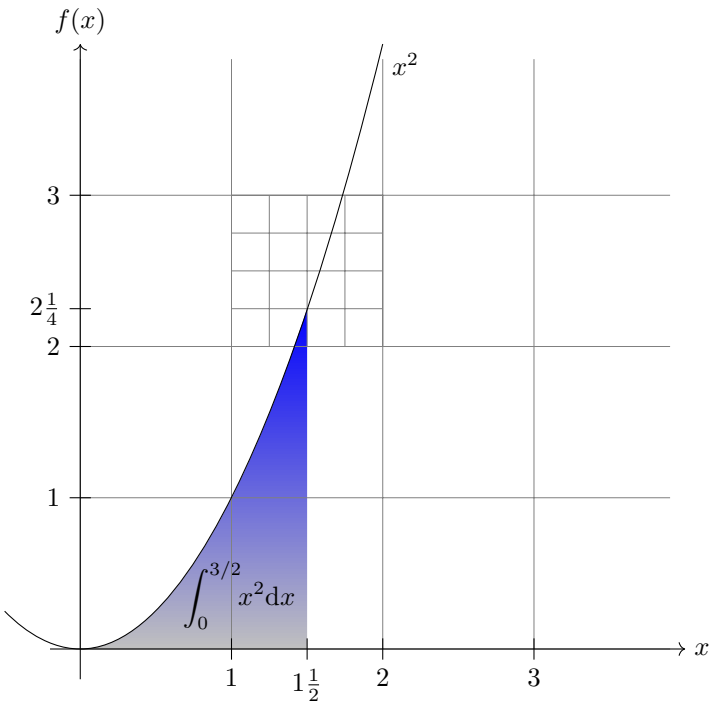
- $\sum_{i=1}^{10} x_i$, where $x_i \sim U(-1, 1)$

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

Part VII
Utilities

by Till Tantau

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




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

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

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

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

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

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

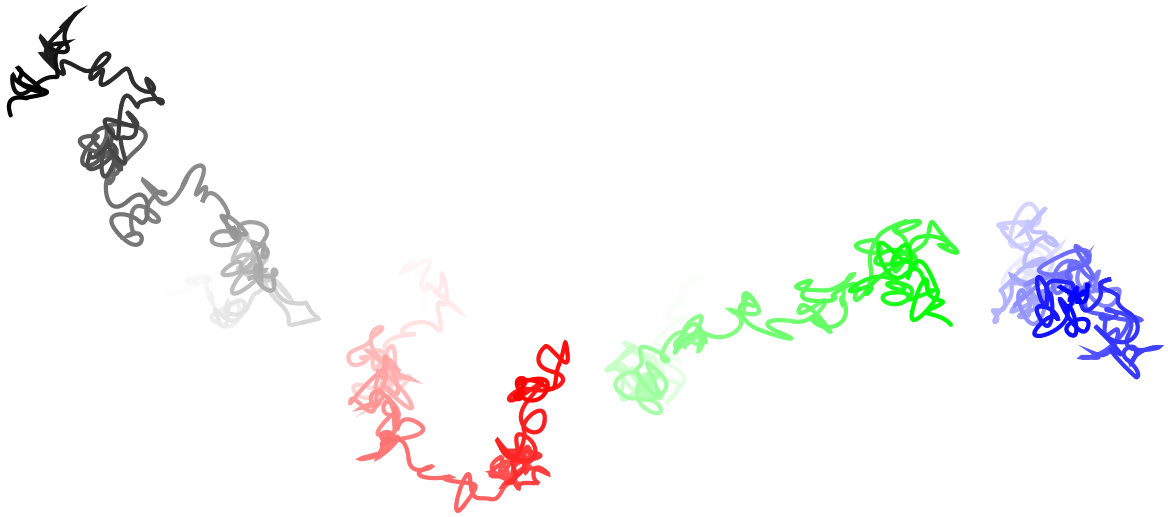
Part VIII
Mathematical and Object-Oriented Engines

by Mark Wibrow and Till Tantau

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

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

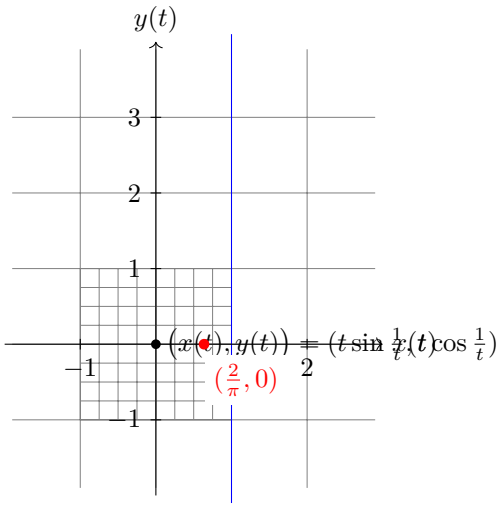
The job of the object-oriented engine is to support simple object-oriented programming in TEX. It allows the definition of *classes* (without inheritance), *methods*, *attributes* and *objects*.



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

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

Part IX
The Basic Layer

by Till Tantau




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

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

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

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

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

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

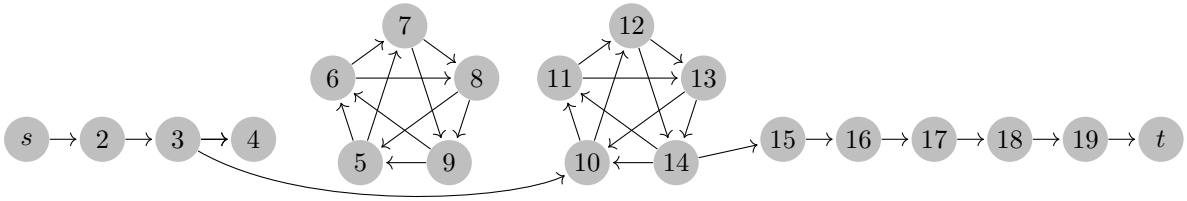
Part X
The System Layer

by Till Tantau

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

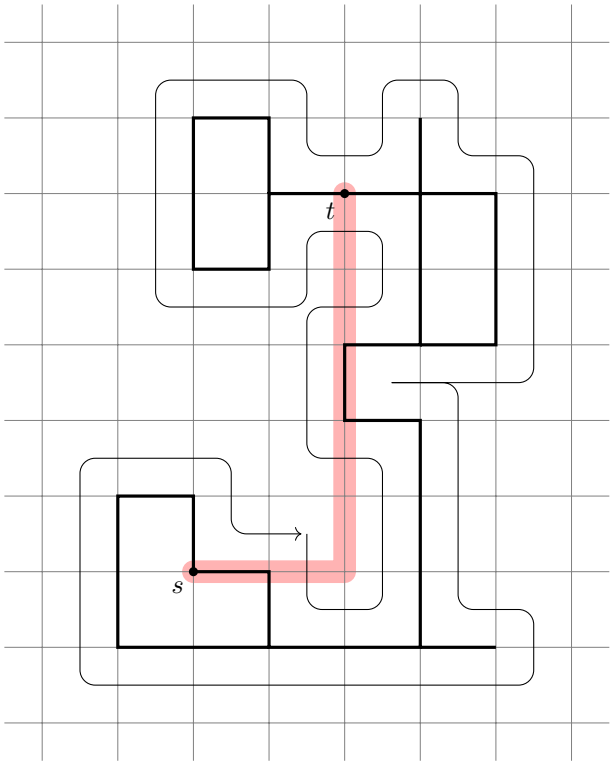
Unless you intend to port PGF to another driver or unless you intend to write your own optimized frontend, you need not read this part.

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




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

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

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

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

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

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

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

Part XI
References and Index




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