

Ti*k*Z  
L!KΣ

&  
8

PGF  
BCE

## 3.1.5b 版 中文手册

3.1.5b 版 中文手册

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

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

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

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

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

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

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

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

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

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

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

Für meinen Vater, damit er noch viele schöne T<sub>E</sub>X-Graphiken erschaffen kann.

*Till*

Copyright 2007 to 2013 by Till Tantau

Permission is granted to copy, distribute and/or modify *the documentation* under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

Permission is granted to copy, distribute and/or modify *the code of the package* under the terms of the GNU Public License, Version 2 or any later version published by the Free Software Foundation. A copy of the license is included in the section entitled GNU Public License.

Permission is also granted to distribute and/or modify *both the documentation and the code* under the conditions of the LaTeX Project Public License, either version 1.3 of this license or (at your option) any later version. A copy of the license is included in the section entitled L<sup>A</sup>T<sub>E</sub>X Project Public License.

# TikZ 和 PGF 宏包

3.1.5b 版 中文手册<sup>\*†</sup>

<http://sourceforge.net/projects/pgf>

Till Tantau<sup>‡</sup>

Institut für Theoretische Informatik

Universität zu Lübeck

August 29, 2015 <sup>§</sup>

## Contents

<b>1</b>	<b>引言</b>	<b>5</b>
1.1	TikZ 底下的层 . . . . .	5
1.2	和其他图形宏包的对比 . . . . .	6
1.3	实用宏包 . . . . .	6
1.4	如何阅读本手册 . . . . .	7
1.5	作者与致谢 . . . . .	7
1.6	获取帮助 . . . . .	7
<b>I</b>	<b>教程和指导</b>	<b>8</b>
<b>2</b>	<b>绘图指导</b>	<b>9</b>
2.1	规划绘图用时 . . . . .	9
2.2	绘图的工作流程 . . . . .	9
2.3	关联图文内容 . . . . .	10
2.4	统一图文风格 . . . . .	10
2.5	图的标签 . . . . .	11
2.6	Plots 和 Charts . . . . .	11
2.7	注意力的集中与分散 . . . . .	14
<b>II</b>	<b>安装和配置</b>	<b>15</b>
<b>III</b>	<b>绘何物为</b>	<b>16</b>

---

<sup>\*</sup>翻译者: Hansimov. 此项目开源在 GitHub, 欢迎提出建议或参与翻译: <https://github.com/Hansimov/pgfmanual-zh>

<sup>†</sup>本手册中译者所加注释均为蓝色, 与原来的注释区分开来。



<sup>‡</sup>该手册的编者。手册的部分内容由另外一些作者写成, 详见 1.5 小节。

<sup>§</sup>本中文手册更新时间: 2020 年 6 月 25 日

<b>3</b>	<b>设计原则</b>	<b>17</b>
3.1	Special Syntax For Specifying Points . . . . .	17
3.2	Special Syntax For Path Specifications . . . . .	17
3.3	Actions on Paths . . . . .	17
3.4	Key-Value Syntax for Graphic Parameters . . . . .	18
3.5	Special Syntax for Specifying Nodes . . . . .	18
3.6	Special Syntax for Specifying Trees . . . . .	18
3.7	Special Syntax for Graphs . . . . .	19
3.8	Grouping of Graphic Parameters . . . . .	19
3.9	Coordinate Transformation System . . . . .	19
<b>IV</b>	<b>Graph Drawing</b>	<b>21</b>
<b>V</b>	<b>Libraries</b>	<b>22</b>
<b>VI</b>	<b>Data Visualization</b>	<b>23</b>
<b>VII</b>	<b>Utilities</b>	<b>24</b>
<b>VIII</b>	<b>Mathematical and Object-Oriented Engines</b>	<b>25</b>
<b>IX</b>	<b>The Basic Layer</b>	<b>26</b>
<b>X</b>	<b>The System Layer</b>	<b>27</b>
<b>XI</b>	<b>References and Index</b>	<b>28</b>

# 1 引言

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

我想从“什么是 TikZ?”这个问题开始。大体上，它只是定义一些 TeX 中的绘图命令。比如说，代码 `\tikz \draw (0pt,0pt) --(20pt,6pt);` 会生成一条线段 ，代码 `\tikz \fill[orange] (1ex,1ex) circle (1ex);` 会生成一个橙色实心圆 。从某种意义上说，当你用 TikZ 的时候，你是在编写图形程序，就像你用 TeX 编写文档程序一样。这也解释了 TikZ 这个名字的由来：它是一个递归缩写，追随了“GNU is not unix”的传统。它的德文含义是“TikZ ist *kein* Zeichenprogramm”，翻译成英文就是“TikZ is not a drawing program”，中文意思是“TikZ 不是一个绘图程序”，它在提醒读者注意这个宏包的真正意图。TikZ 采用“TeX 式的排版”，因此在绘图时沿袭了这样一些优点：迅速创建简单图形，精准定位，可以使用宏，以及一流的排版。同样，它也继承了 TeX 系统的缺点：学习曲线陡峭，不是“所见即所得” (WYSIWYG, What You See Is What You Get)，微小的改变就需要长时间的重新编译，并且代码并不能真正“展示出”事物将有的样子。

现在我们知道什么是 TikZ 了，那么 PGF 呢？我们之前提过，TikZ 一开始是 TeX 中一些绘图的宏，并且能同时用在 pdfLaTeX 和传统的（基于 PostScript 的）LaTeX。换句话说，我当时想的是为 TeX 实现一个“便携式图形格式” (portable graphics format)，这也是为什么叫 PGF。这些早期的宏仍然在用，并且组成了该手册所述的“基本层” (basic layer)。不过，如今一名写作者几乎都在和 TikZ 打交道——它已经自成一套完整的语言了。

## 1.1 TikZ 底下的层

事实上，在 TikZ 底下有两层 (layer)：

**系统层：** 这一层提供了对驱动的完整抽象。驱动就是一个程序，比如 dvips 或者 dvipdfm，输入 .dvi 文件，输出 .ps 或者 .pdf 文件。（pdfTeX 也可以算作是驱动，尽管它的输入并不是 .dvi 文件。这没有影响。）每个驱动都有一套自己的生成图形的语法，每个想用统一语法绘图的人，都对此感到很头疼。而 PGF 的系统层则将这些差异抽象掉了。比如，系统命令 `\pgfsys@lineto{10pt}{10pt}` 延伸了从当前图片 `{pgfpicture}` 到相对坐标 (10pt, 10pt) 的路径。对于不同的驱动，比如 dvips、dvipdfm 或者是 pdfTeX，系统命令会将其转换成不同的 `\special` 命令。系统层尽可能做到简约，毕竟每加一个额外的命令，都意味着将 PGF 对接到新驱动时，要做更多的工作。

作为一个使用者，你不会直接接触系统层。

**基本层：** 基本层提供了一套基本的命令，在创建复杂图形时，用基本层要比系统层简单的多。比如说，系统层没有画圆的命令，因为圆形几乎可以由更加基础的贝塞尔曲线 (Bézier curves) 组成。然而作为一个使用者，在画圆时，你只想用一条简单的命令（至少我是这样），而不是写上半页纸的贝塞尔曲线的控制点坐标。因此，基本层有一个 `\pgfpathcircle` 的命令，帮你生成必要的曲线坐标。

基本层拥有一个核心 (core)，它包含了几个相互依赖的包，并且只能一起导入，而其它的模块 (modules) 则用于扩展核心，提供更多特定用途的命令，比如操纵节点 (node) 或者作图。例如，BEAMER 宏包只用了基本层的核心，而没有用到 shapes 这样的模块。

理论上，TikZ 本身只是几个可能的“前端”之一，集成了一些命令或者特定的语法，使基本层更容易使用。直接使用基本层有一个问题，就是代码通常都很“冗长”。比如要画一个简单的三角形，用基本层你可能需要多达 5 条命令：第 1 条命令从三角形的一个角开始一条路径，第 2 条命令将这条路径延伸到第二个角，第 3 条命令连接第三个角，第 4 条命令闭合路径，第 5 条命令将三角形“绘画” (paint) 出来（不同于“填充” (fill)）。如果用 TikZ 这个前端，这些可以归为一句 METAFONT 式的简单命令：

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

实际上, `TikZ` 是 PGF 唯一“正式”的前端。`TikZ` 可以访问 PGF 的所有特性, 不过更容易使用。它的语法结合了 `MEATFONT` 和 `PSTRICKS`, 外加一些我自己的想法。除了 `TikZ`, 还有一些其他的前端, 但它们大多更倾向于“技术研究”, 并且没有 `TikZ` 正式。特别是 `pgfpict2e` 这个前端, 使用 PGF 的基本层, 重新实现了标准的  $\text{\LaTeX}$  `{picture}` 环境, 以及 `\line` 和 `\vector` 这样的命令。这一层其实并没有必要, 因为 `pict2e.sty` 宏包在重新实现 `{picture}` 环境上同样出色。当然, `pgfpict2e` 宏包背后的出发点是, 简单展示一下如何实现一个前端。

由于大多数使用者只会用到 `TikZ`, 几乎不会有人直接用到系统层, 因此这个手册的前面部分主要涉及 `TikZ`, 基本层和系统层将在最后介绍。

## 1.2 和其他图形宏包的对比

`TikZ` 并不是  $\text{\TeX}$  中唯一的图形宏包。接下来, 我会将 `TikZ` 和其他宏包作一个非常合理的对比。

1. 标准的  $\text{\LaTeX}$  `{picture}` 环境可以创建基本图形, 但只是一点点。这并不是  $\text{\LaTeX}$  的设计者缺乏知识或者想象力, 相反, 这是 `{picture}` 环境的可移植性带来的代价: 它得和所有后端驱动打交道。
2. `pstricks` 宏包确实很强大, 可以创建任何你能想到的图形, 但是它不够方便。更重要的是, 它既不兼容 `pdfTeX`, 也不兼容任何其他驱动, 除了 PostScript 代码, 别的什么也不能产生。  
`pstricks` 的基础支持和 `TikZ` 类似, 在过去十年里, 使用者们出于特定用途, 贡献了许多出色的额外宏包。`TikZ` 的语法比 `pstricks` 更一致, 因为 `TikZ` 的开发“更加中心化”, 并且“牢记 `pstricks` 的缺点”。
3. `xypic` 是一个更老的绘图宏包, 它更难学习和使用, 因为它的语法和文档有点晦涩。
4. `dratex` 是一个小的绘图宏包, 相比其他宏包 (包括 `TikZ`), 它真的很小, 也许是个优点, 也许不是。
5. `metapost` 是 `TikZ` 一个强大的替代物, 它一般作为外部程序使用, 不过伴随着一堆问题, 但是  $\text{\LuaTeX}$  现在已经将它集成进去了。`metapost` 嵌入标签有点麻烦, 而用 PGF 则非常容易。
6. 如果使用者不想编写图形程序 (而这在 `TikZ` 和其他宏包中是必要的), 那么可以用 `xfig` 替代 `TikZ`。有程序可以将 `xfig` 图形转换成 `TikZ`。

## 1.3 实用宏包

PGF 宏包伴随着一些实用宏包 (utility package), 它们并不和绘图相关, 并且可以独立于 PGF 使用。不过它们和 PGF 打包在一起, 一部分是为了方便, 另一部分是因为它们的功能和 PGF 紧密相关。这些实用宏包有:

1. `pgfkeys` 宏包提供了强大的关键字 (key) 管理功能, 可以完全独立于 PGF 使用。
2. `pgffor` 宏包定义了一个有用的 `\foreach` 语句。
3. `pgfcalendar` 宏包定义了创建日历的宏, 通常这些日历可以用 PGF 的图形引擎渲染, 但是你也可以用 `pgfcalender` 在普通文本中排版日历。该宏包还定义了一些命令用于处理日期。
4. `pgfpages` 宏包用于将多页合并成一页, 它提供了将多张“虚拟页”合并成单张“物理页”的命令。思路是每当  $\text{\TeX}$  准备好将一页“输出” (shipout) 时, `pgfpages` 打断这个输出, 并且将该页存到一个特殊的盒子 (box) 里。等到积累了足够的“虚拟页”, 会将它们按比例缩小, 并且排列到一张“物理页”中, 然后将其真正地输出来。这个机制可以让你把文档的每两页并成一页, 而且直接在  $\text{\LaTeX}$  中操作, 不需要借助任何外部程序。`pgfpages` 还能做得更多, 你可以在页面中加上徽标 (logo)、水印 (watermark) 和边框 (border), 或者是将 16 页合成 1 页, 等等。

## 1.4 如何阅读本手册

本手册讲述了 TikZ 的设计和使用。文档几乎是遵循“用户友好”的原则组织的，最简单和最常用的命令和子宏包最先介绍，底层的和深奥的特性则在后面介绍。

第一，如果你还没装 TikZ，请先阅读安装步骤。第二，建议你通读教程部分。第三，你可能想浏览一遍 TikZ 的描述。如果你想写自己的前端，或者将 PGF 对接到新的驱动，那么通常情况下，你不必阅读“基本层”那章，只需要阅读“系统层”这章就行了。

系统中“公开”(public)的命令和环境会在正文中介绍，命令、环境和选项用红色，绿色表示该部分是可选的，可以不加。

## 1.5 作者与致谢

大部分 PGF 系统及其文档是 Till Tantau 写的。主要团队的另一位成员是 Mark Wibrow，他负责 PGF 的数学引擎、许多形状、装饰引擎以及矩阵库。第三位成员是 Christian Feuersänger，他贡献了浮点库、图像外化 (image externalization)、扩展的关键字处理以及手册中的自动超链接。

还有一些人作出了临时的贡献，包括：Christophe Jorssen、Jin-Hwan Cho、Olivier Binda、Matthias Schulz、Renée Ahrens、Stephan Schuster 以及 Thomas Neumann。

此外，还有许多人也为 PGF 作出了贡献，比如写邮件、发现错误 (spot bug)、或者是提交库和补丁。非常感谢他们中的所有人，只是人数太多，无法一一陈列姓名。

## 1.6 获取帮助

如果你在使用 PGF 和 TikZ 时需要帮助，请照下面这样做：

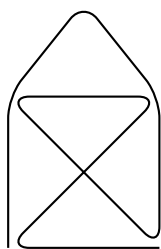
1. 阅读手册，至少是和你的问题相关的那部分。
2. 如果阅读手册没有解决问题，试着看一下 sourceforge 上 PGF 和 TikZ 的开发页面（见本手册标题页）。也许有人已经报告了一个相同的问题，并且有人已经找到了一个解决方法。
3. 在网站上你能找到很多求助的论坛，你可以在论坛发帖，提交错误报告，加入邮件列表，等等。
4. 在你提交错误报告前，尤其是和安装有关的，请确保这真的是个错误。特别是要看一下  $\text{\TeX}$  处理后得到的 .log 文件，它应该显示所有正确的文件都从正确的文件夹导入了。几乎所有的安装问题，都能通过查看 .log 文件解决。
5. 如果万不得已，你可以给我 (Till Tantau) 发邮件，如果问题和数学引擎相关，那么找 Mark Wibrow。我不介意收到邮件，我收到太多了。因此，我无法保证及时甚至是能够回复你的邮件。如果你发到 PGF 的邮件列表，也许问题解决的可能性要更高一些（当然了，如果有时间，我会查看这个列表并且回答问题）。

## Part I

# 教程和指导

*by Till Tantau*

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



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



## 2 绘图指导

这一节不是讲 PGF 和 TikZ 的，而是讲在科学报告、论文和书籍中绘制图形时，通用的指导和原则。

本节的指导源自不同的地方，我想声明的内容，大多都只是“常识”，一些基于我的个人经历（当然，我希望并不只是我个人的偏好），一些来自图形设计和排版的书籍（还没写参考文献，见谅）。

当有人给你列出一堆指导时，你首先得问自己：我真的应该遵循这些指导吗？这是个重要的问题，因为有很多好理由不去遵循这些通用的指导。给出这些指导的人，目标可能和你的并不一致。比如说，某条指导可能写“用红色来强调”，这可能非常适合用投影仪做的报告，但是对于黑白打印的内容来说，红色可能就起了相反的效果。指导几乎总是用于处理特定的情况，在错误的情况下遵守它只会弊大于利。

关于排版的基本规则，你要知道的第二件事是：“每一条规则都可以打破，只要你确实“意识到”你在打破某条规则。”这条规则也适用于绘图。上面那句话换个说法，就是：“排版时唯一的错误，就是对发生的事一无所知。”如果你想打破一条规则并且清楚后果，那么打破它。

### 2.1 规划绘图用时

如果你要写一篇图很多的文章，那么一个重要的因素是，画这些图要花多久。你怎样计算绘图所需的时间呢？

我们假设，画一张图花费的时间，等于写同样篇幅的文字。比如我写文章，初稿可能一页一小时，到后面修改时，每页可能需要两到四小时。那么，要画半页左右的图，初稿我预计需要半个小时，后面还需要一到两小时，完成最终的图。

在许多出版物甚至是优秀的期刊中，作者和编辑明显在文字上花了很多工夫，但是似乎只花了五分钟就画好了所有图。通常这些图好像就是“后加上的”，或者只是统计软件的截图。正如后面会讨论的，像 GNPLOT 这种软件默认画出来的图，质量并不高。

结合文字绘制信息图，从而帮助读者理解，是一个困难而漫长的过程。

- 把图形作为你文章的一等公民。图形值得花费同文字相等的时间和精力。事实上，相比文字，绘图可能值得投入更多的时间，因为人们第一眼看到的就是图形，也更关注图形。
- 给图形的绘制和修改规划尽可能多的时间，就像对待同等篇幅的文字一样。
- 信息量大的困难的图形，可能需要更多的时间。
- 简单的图形需要更少的时间，但是无论如何，很可能你并不想在文章里放“非常简单的图”，就像你不想在文章中写同等篇幅“非常简单的文字”一样。

### 2.2 绘图的工作流程

你写一篇（科学）文章，通常会遵循下面的模式：你有一些结果或者想法要阐述。写文章时一般会先列一个粗糙的大纲，然后分别写各个章节，得到初稿。在成稿写好前，一般要不停地大量地修改。一篇好的期刊文章，初稿里几乎没有一句到最后还没改过的。

绘图也遵循相同的模式：

- 决定图形想要表达什么。一定要有意识地思考，“图形应该告诉读者什么？”
- 列一个“大纲”，也就是图形整体的大致“轮廓”，包含最重要的元素。在这一步，笔纸一般很有帮助。
- 补充和完善图形的细节，得到初稿。
- 根据文章内容，不断修改图形。

## 2.3 关联图文内容

图形可以置于文本中的不同位置。既可以插在行内，也就是“文字中间”，也可以放在单独的“图片”中。由于印刷者（也就是人们）喜欢将页面“填满”（同时出于美学和经济的考虑），因此独立的图片通常会放到离相关文字很远的地方。基于技术原因， $\text{\LaTeX}$  和  $\text{\TeX}$  倾向于鼓励图形的这种“游离”形式。

插在行内的图形，或多或少和正文有些关联，因为周围的文字间接地解释了图形的标签，并且通常正文也会阐明这个图形和什么相关，展示了什么。

独立的图片则大不相同，读者在看到它们的时候，也许还没读到与之相关的文字，或者已经读过很久了。因此，如果你要绘制独立的图片，应该遵循如下指导：

- 独立的图片应当有一个标题，并且“顾名思义”。
  - 比方说，假设一张图展示了快排算法的不同阶段，那么图片的标题至少应当告诉读者，“该图展示了快排算法的不同阶段，如 xyz 页所述”，而不仅仅是“快排算法”。
  - 好的标题会加上尽可能多的上下文信息。比如你会写：“该图展示了快排算法的不同阶段，如 xyz 页所述。在第一行中，选择基准元素 5，这会造成 ...” 这些信息当然也可以在正文中写出，但是放在标题里保证了上下文。不要害怕写一个长达五行的标题。（你的编辑可能会因此讨厌你，你可以把讨厌反弹回去。）
  - 在正文中，你可以这样引用图片：“有一个快排的‘实际’例子，见第 xyz 页的图 2.1。”
  - 很多讲样式和排版的书，会建议你不要缩写成“Fig. 2.1”，而是应该写“Figure 2.1”。
- 反对缩写的主要论据是，“不应在缩写这种地方浪费句点的价值”。意思是，句点会让读者以为句子结束于“Fig”，并且需要“有意识的回溯”，才能发现句子根本没有结束。

支持缩写的论据是，节省空间。

我个人并不信服任何一种论据。一方面，我还没见过有力的证据，表明缩写会降低阅读速度；另一方面，在大多数文章中，如果将所有“Figure”缩写成“Fig.”，省下的空间几乎连一行都不到。我会避免使用缩写。

## 2.4 统一图文风格

人们在绘图时犯得最多的一个“错误”（记住，设计中的“错误”通常只是“无知”），也许就是图形和文字的式样不搭。

一个很常见的情况是，文章的配图来自几个不同的程序。作者可能用  $\text{GNUPLOT}$  作图， $\text{XFIG}$  画图表，以及插入一张  $\text{.eps}$  格式的图片（它是合作者画的，不知道用的什么程序）。这些图很可能用了不同的线宽、字体和尺寸。此外，作者在插入图片的时候，还经常用 `[height=5cm]` 这样的选项将图片缩放到“漂亮的尺寸”。

如果写文字也用这种方法的话，那么就相当于：不同的章节用不同的字体和字号。书写定理时，在某些章节中都加了下划线，在另一个章节则全用大写字母，再在另外一个章节用红色字体。此外，不同页面的边距还不一样。读者和编辑一定不能忍受这样的文字，但是对于图形他们常常不得不忍受。

为了保持图文风格一致，请忠于如下指导：

- 不要缩放图形。
- 意思是说，如果用别的程序画图，那么要输出“正确的尺寸”。
- 在图文中使用相同的字体。
  - 在图文中使用相同的线宽。
- 普通文本的“线宽”是指，T 这样的字母中竖直笔画的宽度。在  $\text{\TeX}$  中，这个值通常是 0.4pt。不过有些期刊不接受线宽低于 0.5pt 的图形。
- 图文中使用统一的颜色。比方说，如果正文用红色表示警告，那么在图形中的重要部分也应该用红色；如果正文用蓝色表示结构元素，比如大标题和章节标题，那么在图形中也应该用蓝色表示结构元素。
- 不过，图形也可以用符合内在逻辑的（logical intrinsic）颜色。比如无论你正常用什么颜色，读者一般会总是认为，绿色表示“正面的、前进、好”（positive, go, ok），而红色表示“警醒、警告、行动”（alert, warning, action）。

用不同的图形程序绘图，几乎不可能保持风格一致。因此，你应该考虑忠于一种图形程序。

## 2.5 图的标签

几乎所有图形中都有标签，也就是文字片段，用于解释图形中的部分内容。加标签时，遵照如下指导：

- 放置标签时请保持一致。需要做到两点：一，标签与正文一致，也就是说，标签应当使用与正文相同的字体。二，标签之间一致，也就是说，如果某个标签采用了特定的格式，那么其他标签也应使用同一格式。
- 图文除了字体一致外，还要符号一致。比如，如果你在正文中用了  $1/2$ ，那么图的标签中也应当用“ $1/2$ ”，而不是“0.5”。 $\pi$  就是“ $\pi$ ”，而不是“3.141”。最后， $e^{-i\pi}$  就是“ $e^{-i\pi}$ ”，而不是“-1”，更不是“-1”。
- 标签要清楚易读。不仅要有合适的大小，还不应被直线或者文字遮挡。这也适用于直线的标签和标签后面的文字。
- 标签的位置要适当。不管空间是否足够，标签都应当靠近所标的内容。只有在必要时，才在标签和所标内容之间加上（柔和的）连接线。标签不应单纯指向外部的图例说明，否则读者得在图中内容和外部说明之间来回跳读。
- 考虑将“不重要的”标签柔和化，比如用灰色，这能让读者更关注对应的图形。

## 2.6 Plots 和 Charts

最常见的图形，尤其是在科学文章中，就是它囊括的范围很广，包括简单的折线图、参数方程图、三维图、饼图，等等等等。

不幸的是，众所周知，很难作好图。部分原因在于，GNUPLOT 或者 Excel 这类程序的默认设置，很容易就作出糟糕的图。

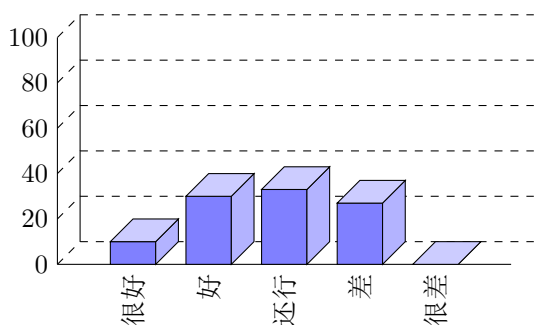
你在作图时，首先得问自己：数据点的个数值得作图吗？如果答案是“不太值得”，那么用一张表。

一个典型的没必要作图的例子，就是把几个数画在柱状图中。这里有个现实的例子：一个研讨会快结束时，演讲者向与会者征询反馈。有 50 名与会者，收回了 30 份反馈表。根据反馈，3 人觉得研讨会“很好”，9 人觉得“好”，10 人觉得“还行”，8 人觉得“差”，0 人觉得“很差”。

要概括这些信息，一个简单的方法就是下面这张表格：

等级	评价人数（共计 50）	百分比
“很好”	3	6%
“好”	9	18%
“还行”	10	20%
“差”	8	16%
“很差”	0	0%
未提交	20	40%

演讲者用三维柱状图将数据可视化出来。看起来像这样：（不过在实际中，打印数字用的是分辨率极低的位图字体，几乎看不清楚）



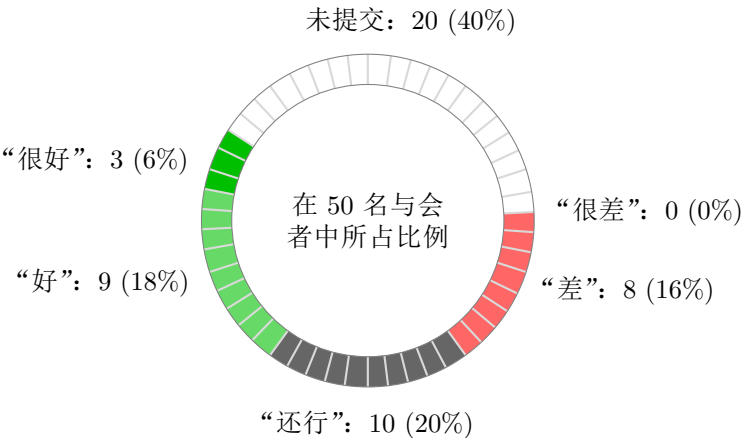
表格和柱状图大小差不多，如果你第一反应是“柱状图比表格好看多了”，那么请根据表格或者柱状图中的信息，试着回答如下问题：

1. 与会者总人数是多少？
2. 提交反馈表的与会者人数是多少？
3. 提交反馈表的与会者百分比是多少？
4. 选择“很好”的与会者人数是多少？
5. 选择“很好”的与会者占有与会者的百分比是多少？
6. 选择“差”或者“很差”的与会者是否超过了四分之一？
7. 选择“很好”的与会者占提交了反馈表的人的百分比是多少？

悲伤的是，从柱状图中我们不能回答上述任何一个问题，而表格则能直接告诉我们所有问题的答案，除了最后一个。实际上，这张柱状图的信息密度几近于零，而表格的信息密度则远高它，尽管只是为了展示几个数，表格就用了许多空白。下面是三维柱状图中的一系列错误：

- 整张图都被恼人的背景线支配了。
- 左侧的数字含义不明，猜测可能是百分比，但也可能是与会者的绝对人数。
- 底部标签旋转了一个角度，很难读。  
(在我看到的实际展示中，文本分辨率很低，每个字母只有  $10 \times 6$  个像素，并且字距还有问题，这让旋转后的文字几乎完全没法读。)
- 第三个维度增加了图的复杂度，但没有提供更多的信息。
- 第三个维度影响了对柱形高度的判断。看下“差”的那根，它比 20 高还是低？柱形的正面比 20 低，背面（真正重要的）比 20 高。
- 几乎不可能说出这些柱形表示的数字，因此，柱形掩盖了想要传递的信息。
- 这些柱形的高度总和是多少？是 100% 还是 60%？
- “很差”的那根表示的是 0 还是 1？
- 为什么柱形用蓝色？

你可能会争辩道，这个例子里，数字是否准确对图形并不重要，重要的是“信息”，也就是“很好”和“好”的比例高于“差”和“很差”。然而，想传达这个信息只要一句话，或者像下图这样更清晰地展示出来：



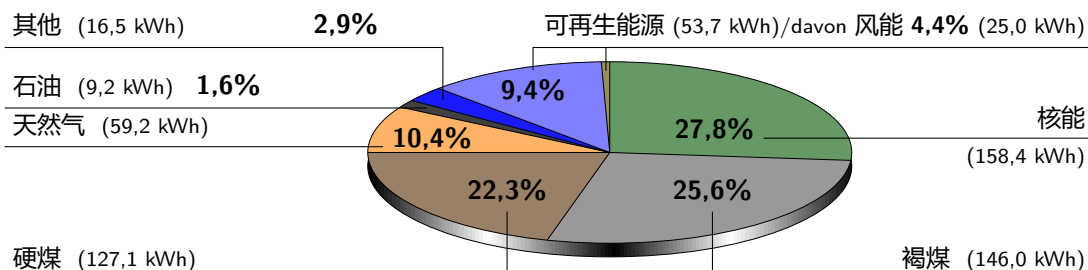
上图拥有和表格相同的信息密度（尺寸和展示的数字都相同）。此外，人们可以直接“看到”好的评价多于差的评价。还可以看出，没有评价的人数不能忽略，这在使用反馈表时很常见。

用图表并不总是好主意。我们来看个例子，这是我从 2005 年 6 月 4 日的德国时代周刊中选取并且重绘的一张饼图。

## 煤最重要

### 2004 年德国发电能源结构

占总体发电量百分比，单位为十亿千瓦时 (Mrd. kWh)



这张图是用 TikZ 重绘的，不过原来的和这个几乎完全一样。

乍看上去，这张图“漂亮且翔实”，但是很多地方都有问题：

- 图是三维的，然而阴影没有提供任何信息，顶多是让人分神。
- 三维饼图的相对尺寸失真得很厉害。比如，灰色部分的“褐煤”比绿色部分的“核能”面积要大，然而实际百分比的大小却正好相反。
- 对于面积小的地方，三维失真更加明显。“核能”比“天然气”的面积要大一些；“风能”比“石油”的面积要小一点，尽管“风能”的百分比几乎是“石油”的三倍。  
在最后一种情况中，失真只是尺寸不同的部分原因。原图的作者把“风能”部分画得太小了，即使是失真也不至于如此。（只要比较一下“风能”和“可再生能源”的大小就能看出来。）
- 根据标题，这张图是想告诉我们，煤是 2004 年德国最重要的能源。然而即使不考虑多余的和误导性的三维设置造成的严重失真，要想领会这个信息也要花上一会。  
煤作为能源被分成了两块：“硬煤”和“褐煤”（两种不同的煤）。把它们加起来后，你会发现饼图的整个下半部分都被煤占据了。  
在视觉上，这两种同属煤的区域却毫无关联，不但颜色不同，标签也位于图的两侧。相比之下，“可再生能源”和“风能”则联系紧密。
- 图中的色彩模式也不合逻辑。为什么“核能”是绿色的？“可再生能源”是浅蓝色的，而“其他能源”是蓝色的？更好笑的是，“褐煤”（字面意思是“褐色的煤”）是石灰色的，而“硬煤”（字面意思是“石头煤”<sup>1</sup>）是褐色的。
- 颜色最亮的区域是“天然气”，因此这个区域也最显眼，然而在这张图里“天然气”一点也不重要。

Edward Tufte 把上面这样的图称为“垃圾图”。（不过，我很开心德国时代周刊不再用三维饼图了，并且信息图也好一点了。）

这里有几条建议，也许能帮你避免画出垃圾图：

- 不要用三维饼图，它们是魔鬼。
- 考虑用表格而不是饼图。
- 不要随便配色，要用颜色来引导读者的注意力，以及对事物分类。
- 用颜色，而不是交叉线或者对角线这样的背景图案。信息图中背景图案是魔鬼。

<sup>1</sup>这里“硬煤”的原文是德语 Steinkohle，译者根据各种资料和德英转译，中文作“硬煤”。尽管 Stein 意为“石头”，kohle 意为“煤”，但是很多科技术语不能顾名思义译。从原文括号中所附的字面含义看，也许翻译成“石煤”更符合上下文语境。“石煤”属于无烟煤，包含于“硬煤”。因此该处译文存疑。

## 2.7 注意力的集中与分散

拿起你最爱的小说，看一眼其中典型的一页。你可以注意到页面非常统一。没有什么让读者分神；没有大标题，没有粗体，更没有大片空白。事实上，即使作者想要强调某些东西，也是用斜体。这些斜体字很好地融入了正文——你很难看出一页中是否有斜体字，但是你能一眼看出一个粗体单词。小说用这种方式排版，是遵循这个范式：避免分神。

好的排版（如同好的组织）润物细无声。这里排版的作用是，让人在阅读文本时，尽可能容易地“吸收”信息。看小说时，读者通过一行又一行的文字获取信息，就像在听人讲故事一样。这种情况下，只要是影响人眼迅速平滑地掠过字里行间，页面上的任何其他东西都会让文本更难读。

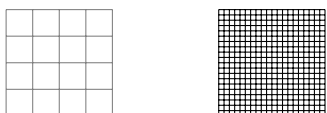
现在，拿起你最爱的周刊或者报纸，看一眼其中典型的一页。你可以看到页面上有非常多“正在发生”的事情。字体大小不一、布局相异，文本放在窄列中，通常还夹杂着图片。杂志用这种方式排版，是遵循另一种范式：引导注意力。

读者不会像读小说一样看杂志。我们不是读杂志的每一行，而是看大标题和短摘要，确定自己是不是想看某一篇文章。这里排版的作用是，先把我们的注意力引导到摘要和标题上。一旦我们想读某篇文章了，我们就不会再容忍令人分神的内容，因此文章正文的排版方式就和小说一样。

“避免分神”和“引导注意力”这两个原则也适用于图形。当你设计图形时，你应该尽可能去除所有“令人分神”的内容。与此同时，你也应该试着用不同的字体、颜色和线宽，来突出不同的部分，以此积极帮助读者“一览图形全貌”。

这里部分列举了会让读者分神的东西：

- 强烈的对比总是会最先吸引眼球，比如下面这两个网格：



按照英文阅读顺序，应该先看到左边的网格，尽管如此，人们却更可能先看到右边的网格：黑白对比相较于灰白对比更加强烈，而且更多的“空间”增强了右边网格整体对比。

像网格或者是辅助线这样更普遍的东西，通常不应该分散读者的注意力。因此，排版这类内容时，同背景的对比应该弱一些。另外，稠密的网格比稀疏的网格更分散注意力。

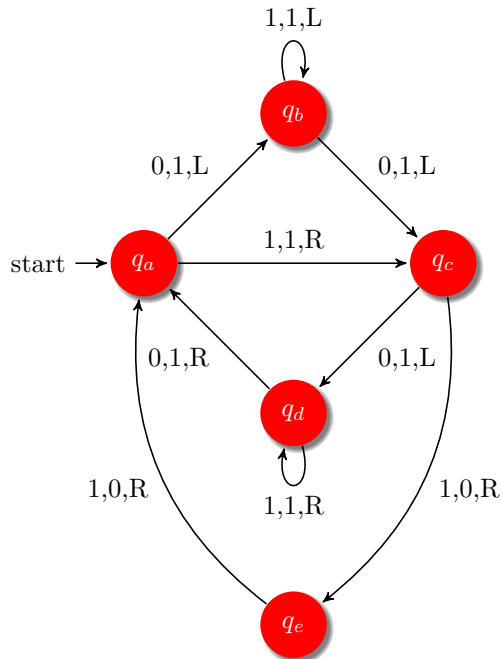
- 虚线有很多点，形成了黑白对比。短划线或者点线可能很令人分神，因此一般避免使用。  
不要在作图中用不同类型的虚线样式来区分曲线。这种方式会丢失数据点，而且眼睛并不擅长“按照虚线样式划分事物”，而是更擅长按照颜色划分。
- 用对角线、水平线、竖直线或者只是点形来填充背景，几乎总会令人分神，而且通常毫无用处。
- 背景图片和渐变也令人分神，只能略微增加一些图形的重要性。
- 可爱的剪贴画很容易分散读者对数据的注意。

## 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) [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}

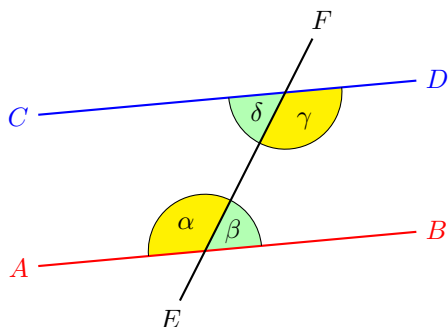
```



## Part III

# 绘何物为

<sup>2</sup>  
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., $\color{red}AB \parallel \color{blue}CD$,
  then $\alpha = \delta$ and $\beta = \gamma$.
};
\end{tikzpicture}
```

<sup>2</sup>该部分标题原文为德文：TikZ ist *kein* Zeichenprogramm, 翻译成英文就是“TikZ is not a drawing program”，中文意思是“TikZ 不是一个绘图程序”。然而德文采用的是“GNU’s Not Unix!”式的递归缩写，这里如果直接采用原文也并非不可以，但是中文博大精深，一定有对应的贴切的翻译。

我这里译成“绘何物为”，用了拼音的递归：Huì hé wù wéi。意即“‘绘’是什么呢”，当然也可以将“绘”直接作为动词，理解成“绘制什么呢”。这样中文含义就和原文含义形成一问一答，无论是形式上还是内容上，都有了合理的对应。当然，这里着实夹杂了译者的私货，正文中依旧使用 TikZ 来指代这一绘图系统。



## 3 设计原则

这一节论述 TikZ 前端背后的设计原则，TikZ 意思是“TikZ 不是一个绘图程序”。要使用 TikZ，L<sup>A</sup>T<sub>E</sub>X 用户得在序言部分加上 `\usepackage{tikz}`，而 plainT<sub>E</sub>X 用户则是 `\input tikz.tex`。TikZ 提供了一种描述图形的语法，易学易用，让你的生活更简单。

TikZ 的命令和语法受几个来源影响。基本的命令名称和路径操作的概念来自 METAFONT，选项机制源于 PSTricks，样式的概念联想自 SVG，图的语法取材于 GRAPHVIZ。为了让它们一起工作，一些折中是必要的。我还加了一些自己的想法，比如坐标变换。

TikZ 遵循下面的基本设计原则：

- 1.
2. 指定的
3. 图形参数用键值对

### 3.1 Special Syntax For Specifying Points

TikZ provides a special syntax for specifying points and coordinates. In the simplest case, you provide two T<sub>E</sub>X dimensions, separated by commas, in round brackets as in `(1cm,2pt)`.

You can also specify a point in polar coordinates by using a colon instead of a comma as in `(30:1cm)`, which means “1cm in a 30 degrees direction.”

If you do not provide a unit, as in `(2,1)`, you specify a point in PGF’s *xy*-coordinate system. By default, the unit *x*-vector goes 1cm to the right and the unit *y*-vector goes 1cm upward.

By specifying three numbers as in `(1,1,1)` you specify a point in PGF’s *xyz*-coordinate system.

It is also possible to use an anchor of a previously defined shape as in `(first node.south)`.


You can add two plus signs before a coordinate as in `++(1cm,0pt)`. This means “1cm to the right of the last point used.” This allows you to easily specify relative movements. For example, `(1,0) ++(1,0) ++(0,1)` specifies the three coordinates (1,0), then (2,0), and (2,1).

Finally, instead of two plus signs, you can also add a single one. This also specifies a point in a relative manner, but it does not “change” the current point used in subsequent relative commands. For example, `(1,0) +(1,0) +(0,1)` specifies the three coordinates (1,0), then (2,0), and (1,1).

### 3.2 Special Syntax For Path Specifications


When creating a picture using TikZ, your main job is the specification of *paths*. A path is a series of straight or curved lines, which need not be connected. TikZ makes it easy to specify paths, partly using the syntax of METAPOST. For example, to specify a triangular path you use

```
(5pt,0pt) -- (0pt,0pt) -- (0pt,5pt) -- cycle
```

and you get  when you draw this path.

### 3.3 Actions on Paths

A path is just a series of straight and curved lines, but it is not yet specified what should happen with it. One can *draw* a path, *fill* a path, *shade* it, *clip* it, or do any combination of these. Drawing (also known as *stroking*) can be thought of as taking a pen of a certain thickness and moving it along the path, thereby drawing on the canvas. Filling means that the interior of the path is filled with a uniform color. Obviously, filling makes sense only for *closed* paths and a path is automatically closed prior to filling, if necessary.

Given a path as in `\path (0,0) rectangle (2ex,1ex);`, you can draw it by adding the `draw` option as in `\path[draw] (0,0) rectangle (2ex,1ex);`, which yields . The `\draw` command is just an abbreviation for `\path[draw]`. To fill a path, use the `fill` option or the `\fill` command, which is an abbreviation for `\path[fill]`. The `\filldraw` command is an abbreviation for `\path[fill,draw]`. Shading is caused by the `shade` option (there are `\shade` and `\shadedraw` abbreviations) and clipping by the `clip` option. There is also a `\clip` command, which does the same as `\path[clip]`, but not commands like `\drawclip`. Use, say, `\draw[clip]` or `\path[draw,clip]` instead.

All of these commands can only be used inside `{tikzpicture}` environments.

TikZ allows you to use different colors for filling and stroking.

### 3.4 Key-Value Syntax for Graphic Parameters

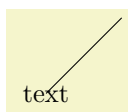
Whenever TikZ draws or fills a path, a large number of graphic parameters influences the rendering. Examples include the colors used, the dashing pattern, the clipping area, the line width, and many others. In TikZ, all these options are specified as lists of so called key-value pairs, as in `color=red`, that are passed as optional parameters to the path drawing and filling commands. This usage is similar to PSTricks. For example, the following will draw a thick, red triangle;



```
\tikz \draw[line width=2pt,color=red] (1,0) -- (0,0) -- (0,1) -- cycle;
```

### 3.5 Special Syntax for Specifying Nodes

TikZ introduces a special syntax for adding text or, more generally, nodes to a graphic. When you specify a path, add nodes as in the following example:



```
\tikz \draw (1,1) node {text} -- (2,2);
```

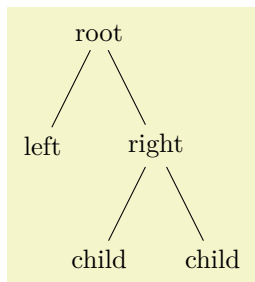
Nodes are inserted at the current position of the path, but either *after* (the default) or *before* the complete path is rendered. When special options are given, as in `\draw (1,1) node[circle,draw] {text};`, the text is not just put at the current position. Rather, it is surrounded by a circle and this circle is “drawn.”

You can add a name to a node for later reference either by using the option `name=(node name)` or by stating the node name in parentheses outside the text as in `node[circle](name){text}`.

Predefined shapes include `rectangle`, `circle`, and `ellipse`, but it is possible (though a bit challenging) to define new shapes.

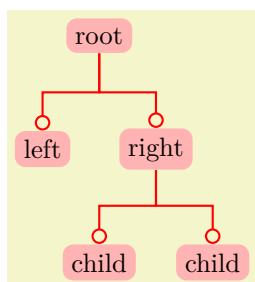
### 3.6 Special Syntax for Specifying Trees

The “node syntax” can also be used to draw trees: A `node` can be followed by any number of children, each introduced by the keyword `child`. The children are nodes themselves, each of which may have children in turn.

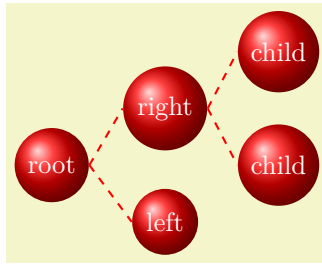


```
\begin{tikzpicture}
  \node {root}
    child {node {left}}
    child {node {right}}
      child {node {child}}
      child {node {child}}
};
\end{tikzpicture}
```

Since trees are made up from nodes, it is possible to use options to modify the way trees are drawn. Here are two examples of the above tree, redrawn with different options:



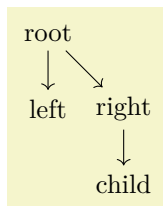
```
\begin{tikzpicture}
  [edge from parent fork down, sibling distance=15mm, level distance=15mm,
  every node/.style={fill=red!30,rounded corners},
  edge from parent/.style={red,-o,thick,draw}]
  \node {root}
    child {node {left}}
    child {node {right}}
      child {node {child}}
      child {node {child}}
};
\end{tikzpicture}
```



```
\begin{tikzpicture}
[parent anchor=east,child anchor=west,grow=east,
 sibling distance=15mm, level distance=15mm,
 every node/.style={ball color=red,circle,text=white},
 edge from parent/.style={draw,dashed,thick,red}]
\node {root}
  child {node {left}}
  child {node {right}}
    child {node {child}}
    child {node {child}}
  };
\end{tikzpicture}
```

### 3.7 Special Syntax for Graphs

The `\node` command gives you fine control over where nodes should be placed, what text they should use, and what they should look like. However, when you draw a graph, you typically need to create numerous fairly similar nodes that only differ with respect to the name they show. In these cases, the `graph` syntax can be used, which is another syntax layer build “on top” of the node syntax.



```
\tikz \graph [grow down, branch right] {
  root -> { left, right -> {child, child} }
};
```

The syntax of the `graph` command extends the so-called DOT-notation used in the popular GRAPHVIZ program.

Depending on the version of T<sub>E</sub>X you use (it must allow you to call Lua code, which is the case for LuaT<sub>E</sub>X), you can also ask TikZ to do automatically compute good positions for the nodes of a graph using one of several integrated *graph drawing algorithms*.

### 3.8 Grouping of Graphic Parameters

Graphic parameters should often apply to several path drawing or filling commands. For example, we may wish to draw numerous lines all with the same line width of 1pt. For this, we put these commands in a `{scope}` environment that takes the desired graphic options as an optional parameter. Naturally, the specified graphic parameters apply only to the drawing and filling commands inside the environment. Furthermore, nested `{scope}` environments or individual drawing commands can override the graphic parameters of outer `{scope}` environments. In the following example, three red lines, two green lines, and one blue line are drawn:



```
\begin{tikzpicture}
\begin{scope}[color=red]
\draw (0mm,10mm) -- (10mm,10mm);
\draw (0mm, 8mm) -- (10mm, 8mm);
\draw (0mm, 6mm) -- (10mm, 6mm);
\end{scope}
\begin{scope}[color=green]
\draw (0mm, 4mm) -- (10mm, 4mm);
\draw (0mm, 2mm) -- (10mm, 2mm);
\draw[color=blue] (0mm, 0mm) -- (10mm, 0mm);
\end{scope}
\end{tikzpicture}
```

The `{tikzpicture}` environment itself also behaves like a `{scope}` environment, that is, you can specify graphic parameters using an optional argument. These optional apply to all commands in the picture.

### 3.9 Coordinate Transformation System

TikZ supports both PGF’s *coordinate* transformation system to perform transformations as well as *canvas* transformations, a more low-level transformation system. (For details on the difference between coordinate transformations and canvas transformations see Section ??.)

The syntax is set up in such a way that it is harder to use canvas transformations than coordinate transformations. There are two reasons for this: First, the canvas transformation must be used with great care and often results in “bad” graphics with changing line width and text in wrong sizes. Second, PGF loses track of where nodes and shapes are positioned when canvas transformations are used. So, in almost all circumstances, you should use coordinate transformations rather than canvas transformations.

## Part IV

# Graph Drawing

*by Till Tantau et al.*

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

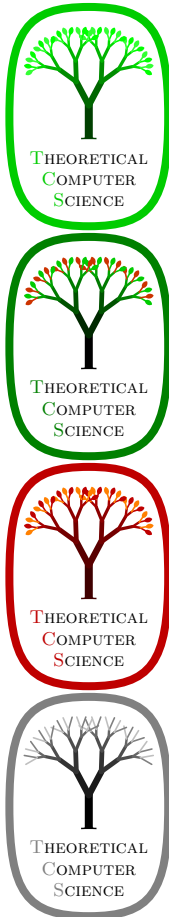
You need to use `LuaTeX` to typeset this part of the manual (and, also, to use algorithmic graph drawing).

# Part V

## Libraries

*by Till Tantau*

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



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

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

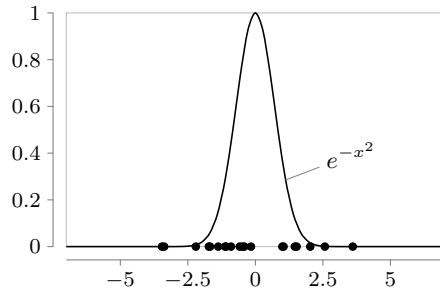
    \coordinate (root) [grow cyclic,rotate=90]
    child {
      child [line cap=round] foreach \a in {0,1} {
        child foreach \b in {0,1} {
          child foreach \c in {0,1} {
            child foreach \d in {0,1} {
              child foreach \leafcolor in {leaf a,leaf b}
                { edge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

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

# Part VI

## Data Visualization

by Till Tantau



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

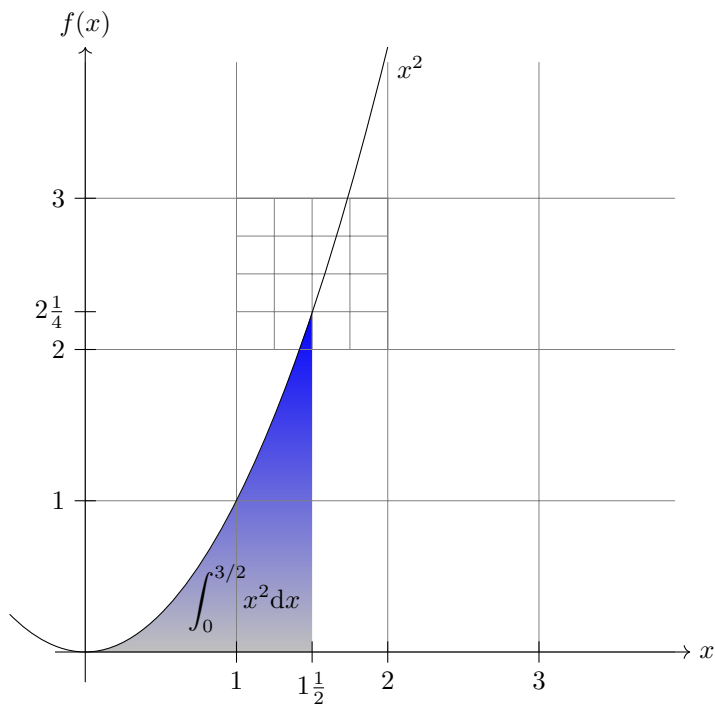
```
\tikz \datavisualization [scientific axes=clean]
[
  visualize as smooth line=Gaussian,
  Gaussian={pin in data={text={\mathit{e}^{-x^2}}},when=x is 1}}
]
data [format=function] {
  var x : interval [-7:7] samples 51;
  func y = exp(-\value x*\value x);
}
[
  visualize as scatter,
  legend={south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

# Part VII

## Utilities

*by Till Tantau*

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



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

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

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

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

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

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



## Part VIII

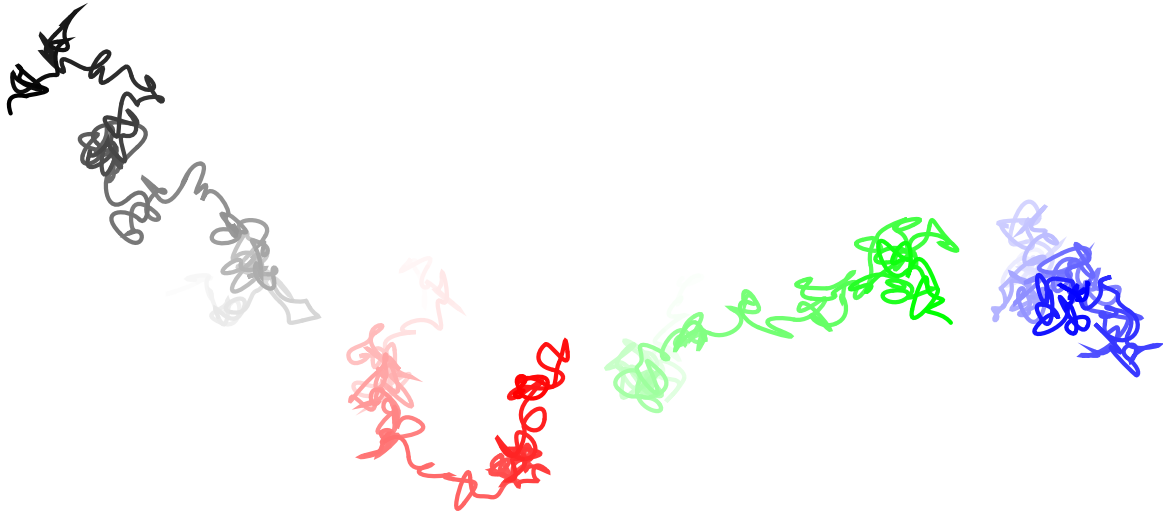
# Mathematical and Object-Oriented Engines

*by Mark Wibrow and Till Tantau*

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

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

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



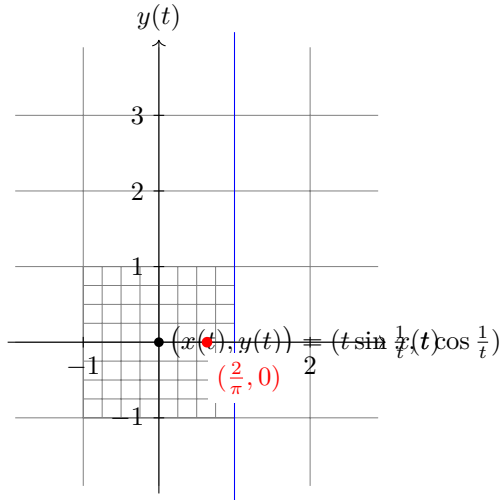
```
\pgfmathsetseed{1}
\foreach \col in {black,red,green,blue}
{
  \begin{tikzpicture}[x=10pt,y=10pt,ultra thick,baseline,line cap=round]
    \coordinate (current point) at (0,0);
    \coordinate (old velocity) at (0,0);
    \coordinate (new velocity) at (rand,rand);

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

## Part IX

# The Basic Layer

by Till Tantau



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

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

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

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

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

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

## Part X

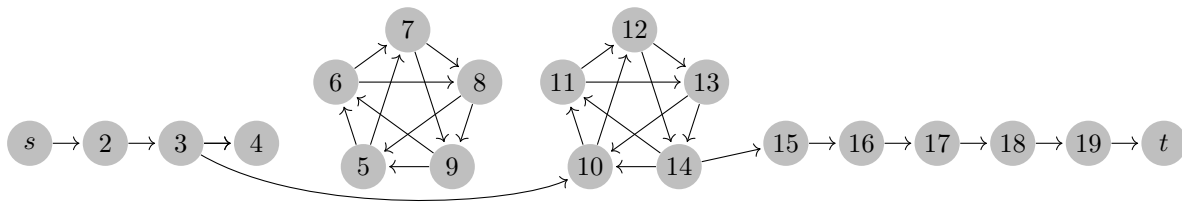
# The System Layer

*by Till Tantau*

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

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

In the following it is assumed that you are familiar with the basic workings of the `graphics` package and that you know what  $\text{\TeX}$ -drivers are and how they work.



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

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

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

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

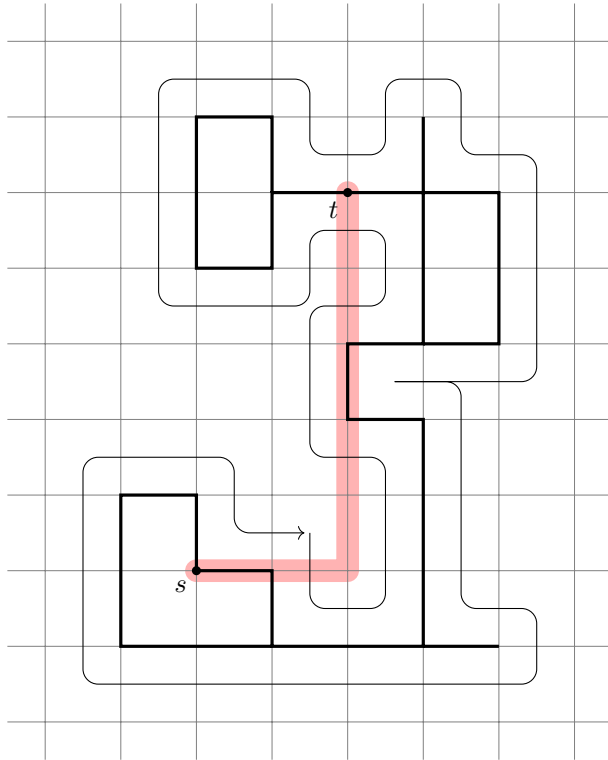
  \foreach \from/\to in {s/2,2/3,3/4,3/4,15/16,16/17,17/18,18/19,19/t}
    \draw (G-\from) -- (G-\to);

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

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

## Part XI

# References and Index



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