tikz 制图指南

在 xelatex 指南之上

万泽[①] | 德山书生[②]

版本：0.01

---

① 作者：湖南常德人氏

② 编 者：德 山 书 生，湖 南 常 德 人 氏。编 者 邮 箱：a358003542@gmail.com。

前言

参考资料：TikZ is not an interactive drawing program.

1.Graphics with TikZ Andrew Mertz and William Slough

2.A very minimal introduction to TikZ Jacques Crémer

3.the tikz 官方文档，这个用 texdoc 命令调不出官方文档，用 google 搜索“tikz pdf”吧

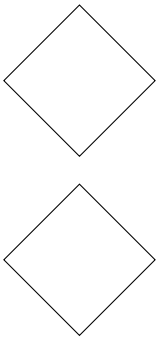
目 录

准备工作

tikz 宏包的加载是必须的，记得加载上。

有两种使用方法，一种命令式的，一种环境式的。命令式用 tikz 命令包围起来，命令式是 inline 模式的。环境式用 tikzpicture 环境命令包围起来。如下所示：

```
\tikz{\draw (1,0) -- (0,1) -- (-1,0) -- (0,-1) -- cycle;}

\begin{tikzpicture}
\draw (1,0) -- (0,1) -- (-1,0) -- (0,-1) -- cycle;
\end{tikzpicture}
```



inline 模式对于注重内容的用户来说会用的较少，有时可能自己建的某些宏包底层会使用到 tikz 命令。但不管是注重华丽表现效果的用户还是注重内容的用户，总是会有需求需要某一整张图片来表达某些内容，而 tikz 以及其他基于 tikz 的宏包在命令行绘图这个领域可以说是很优秀的，这绝不是那些只能绘制基本的线条或者其

他的形状的图形包所能比的。下面将主要使用 tikzpicture 环境命令模式。

quicktikz 软件简介

我自己利用 PyQt 写了一个小软件，名字叫 quicktikz，能够一边画图一边预览，目前还很粗糙，不过只要继续不断优化下去，就会变得很实用了的。

有兴趣的读者可以了解一下，

github 地址.....

tikz 基础

单位

tikz 默认的长度单位是 cm。

第一个例子

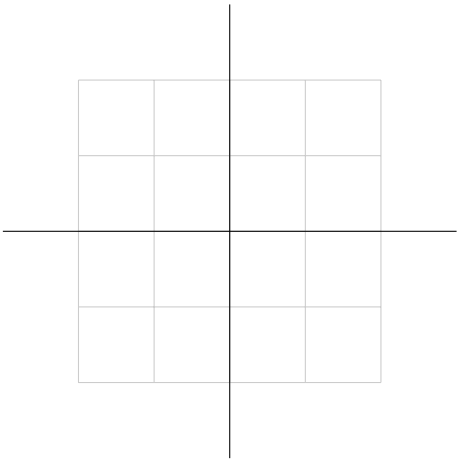
画网格

```
\begin{tikzpicture}
\draw[step=1,color=gray!40] (-2,-2) grid (2,2);
\end{tikzpicture}
```


step 是网格之间的间距，color 是网格的颜色。第一个坐标点是网格的左底点，第二个坐标点是网格的右顶点。我们可以看到 tikzpicture 下每一条命令最后都要跟一个分号; 。

画直线

```
\begin{tikzpicture}
\draw[step=1,color=gray!40] (-2,-2) grid (2,2);
\draw (-3,0) -- (3,0);
\draw (0,-3) -- (0,3);
\end{tikzpicture}
```



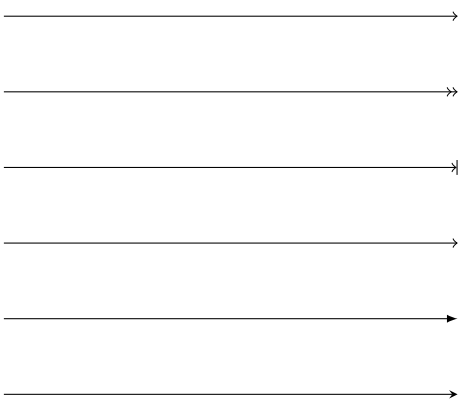
画直线就是两个坐标点相连，中间 -- 符号表示直线的意思。之前网格是 grid 表示网格的意思。

如果几个点用 -- 符号连接起来，表示这几个点连着来画几条折线，有多个画直线命令依次执行的意思。

直线带上箭头

draw 命令可以跟上可选项 **->**，这样直线的右端就有一个箭头了。此外还有：**-»**，**->|**，**-to**，**-latex**，**-stealth**。

他们的效果从上到下依次演示如下：

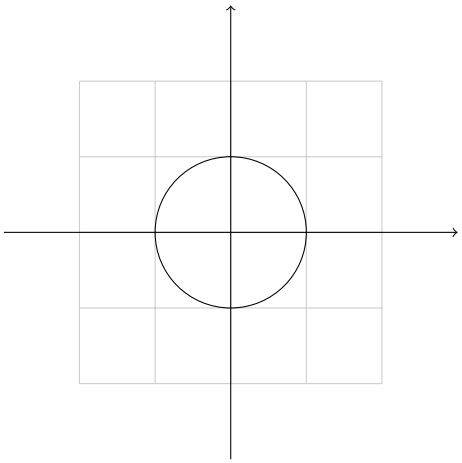


类似的还有左端比如 **<-**，或者两端比如 **latex-latex**，这里就不多说了。

画圆

接著上面的图案画一个圆，加入了以下代码：

```
\draw (0,0) circle (1);
```

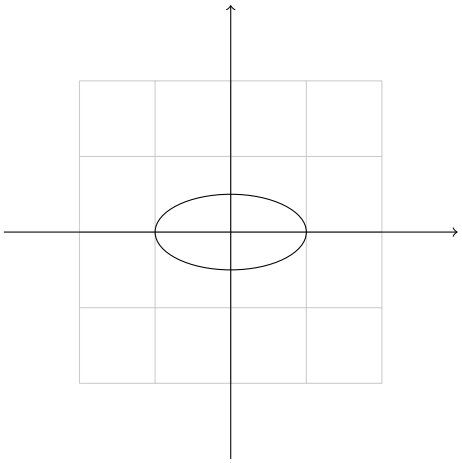


其中第一个点是圆中心，circle 表示画圆，第二个参数是半径大小。

画椭圆

接著画一个椭圆：

```
\draw (0,0) ellipse (1 and 0.5);
```



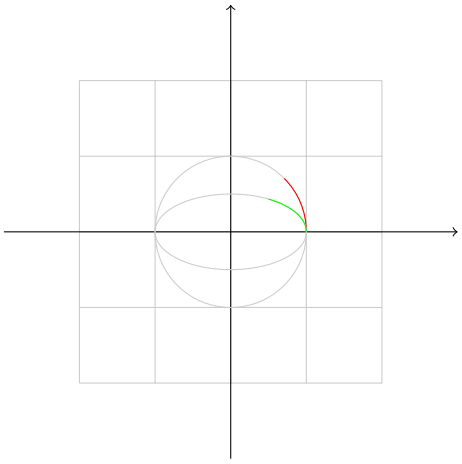
这里第一个点是椭圆的中心点，ellipse 表示画椭圆，后面参数两个值第一个是 a 也就是椭圆的半长轴，第二个是 b 也就是椭圆的半短轴。

画弧线

```
\begin{tikzpicture}
\draw[step=1,color=gray!40] (-2,-2) grid (2,2);
\draw[->] (-3,0) -- (3,0);
\draw[->] (0,-3) -- (0,3);
\draw[color=gray!40] (0,0) circle (1); %
\draw[color=red] (1,0) arc (0:45:1);
\draw[color=gray!40] (0,0) ellipse (1 and 0.5);
\draw[color=green] (1,0) arc (0:60:1 and 0.5);
\end{tikzpicture}
```

最基本的画弧线的命令如上代码第 5 行，其中第一个点是弧线的起点，然后 arc 表示画弧线，接下来括号里面的三个参数：第一个参数是开始的角度，第二个参数是结束时的角度，第三个参数是弧线对应圆的半径。对比第 4 行画的浅灰色的圆可以看出他们之间的关系。

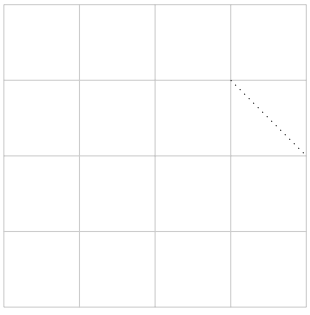
上面代码第 7 行画弧线增加了一个 and 和一个参数，这个时候画的弧线是根据椭圆来的，其中 1 是椭圆的半长轴，0.5 是椭圆的半短轴。对比第 6 行画的浅灰色的椭圆可以看出他们的关系。



点的定义

tikz 中定义一个点方便之后使用：

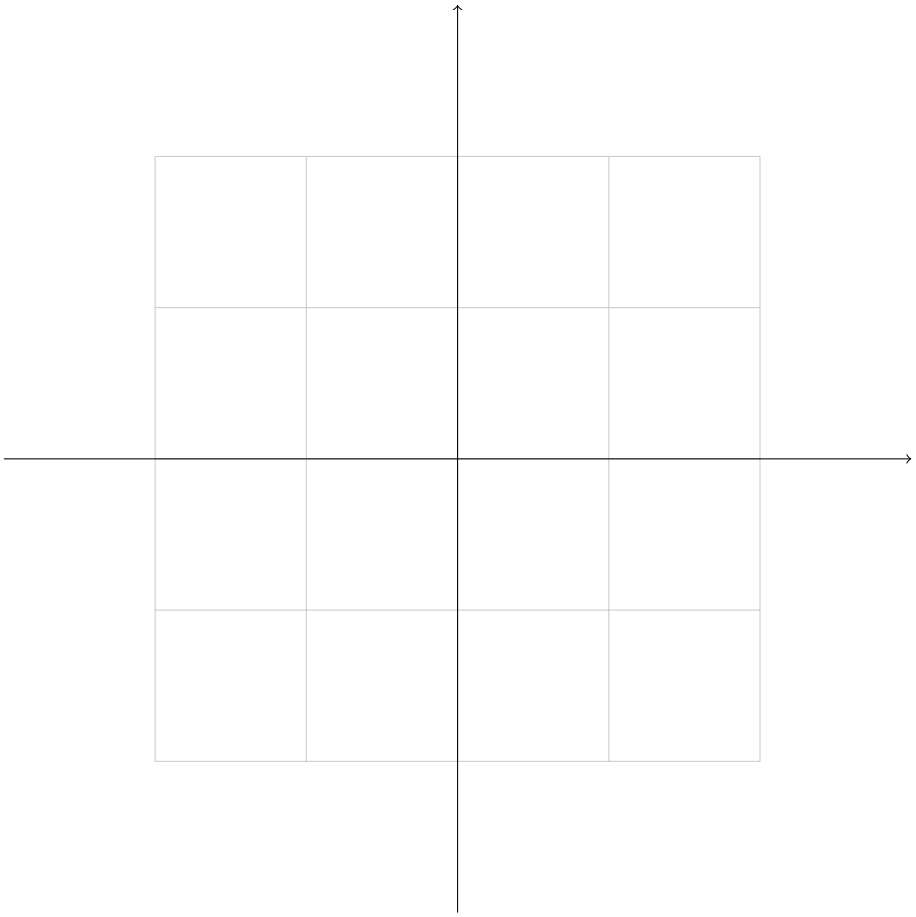
```
\path (1,1) coordinate (point001);
\path (2,0) coordinate (point002);
\draw[dotted] (point001) -- (point002)  ;
```



这里代码的第 6，7 行定义了两个点，名字叫做 point001 和 point002。然后用这两个点作为参数画了一个直线，这个直线有可选项 **dotted**，点线样式。

放大图形

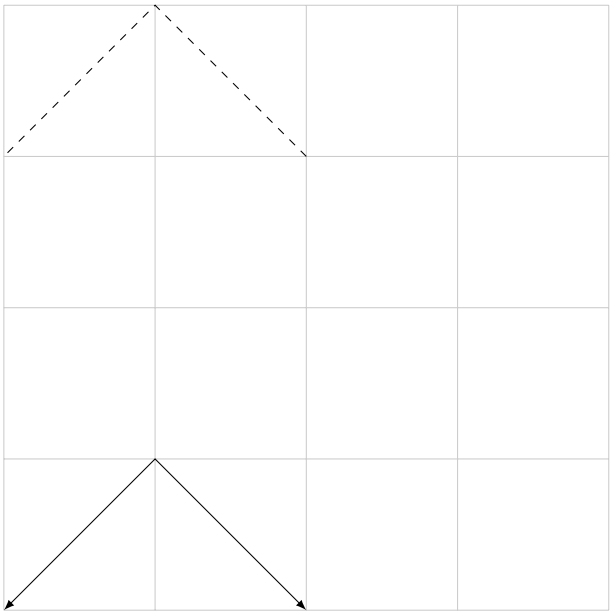
在 tikzpicture 环境后面跟上可选项 [scale=2]，即将图形放大两倍。注意控制别越界了。



点的相对偏移

现在加上这样两行代码：

```
\draw[<->] (0,-2) -- ++(-1,1) -- ++(-1,-1);
\draw[dashed] (0,1) -- +(-1,1) -- +(-2,0);
```

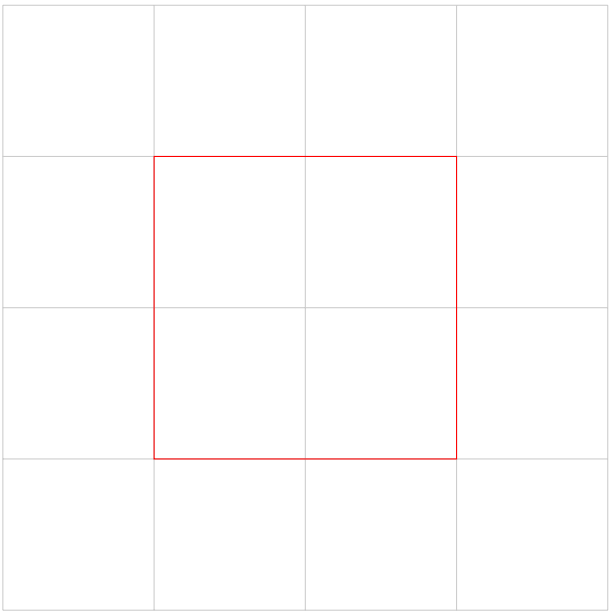


tikz 中有一个重要的概念，当前点，然后点可以通过当前点根据相对偏移来确定一个新的点。上面代码第 9 行的 ++ 符号和第 10 行的 + 符号都根据当前点然后进行了 Δx 和 Δy 的相对偏移从而确定了一个新的点。这两个符号的区别在于是不是更新当前点数据。++ 符号更新当前点，而 + 符号不更新。

画长方形

现在加上这一行代码来画一个长方形：

```
\draw[color=red] (-1,-1) rectangle (1,1);
```



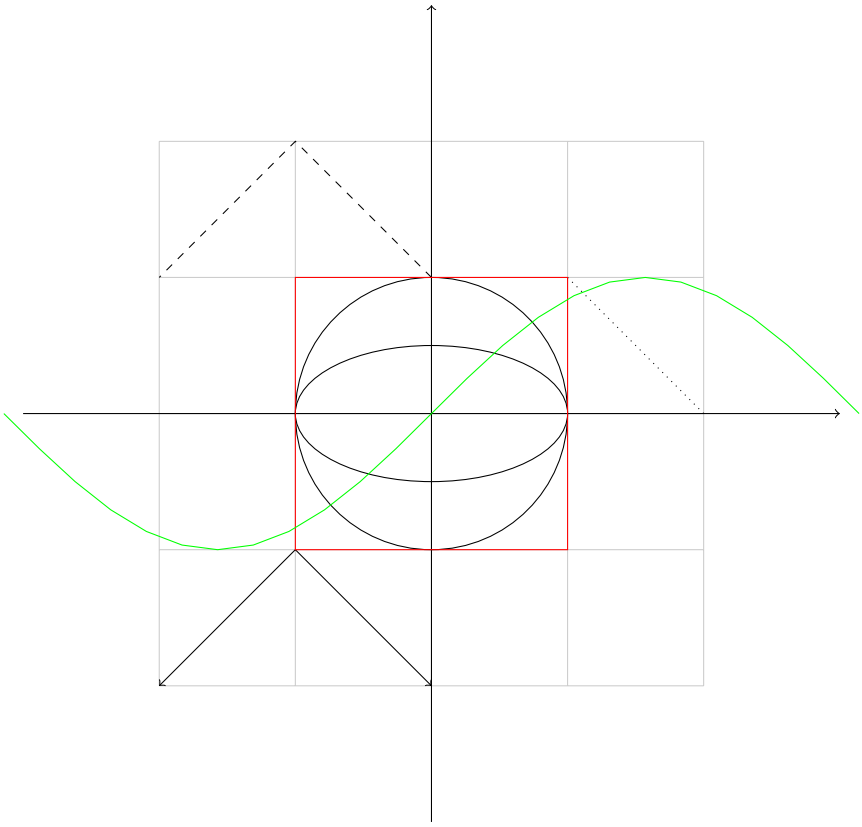
这里使用了可选项 **color=red** 来控制线条的颜色，然后画长方形的第一个点是左底点，rectangle 表示画长方形，第二个点表示右顶点。

画函数

画函数的功能是通过外部程序 gnuplot 来实现了，所以需要打开`--shell-escape`，或者`--enable-write18`

这里最后加了一个语句：

```
\draw[domain=-pi:pi,color=green] plot function{sin(x)};
```



这里可选项 **domain=-pi:pi** 控制画的函数的 x 范围，可以直接用 pi 表示 π，然后接下来 plot function 表示画一个函数，接下来的花括号里面放着 gnuplot 的各种命令，这里就是简单的 $sin(x)$

定义 style

help lines

```
\tikzset{help lines/.style= {step=0.5cm,color=gray!40,very thin}}
\begin{tikzpicture}
\draw[help lines] (0,0) grid (5,5);
\end{tikzpicture}
```

information text

这是一段测试文字。

变量声明

参考网站

def 命令可用，在里面声明一个变量。

pgfmathsetmacro 命令和 def 一样可用来声明变量，不同是里面可以放着一些数学运算公式，tikz 会将其先运算再赋值。

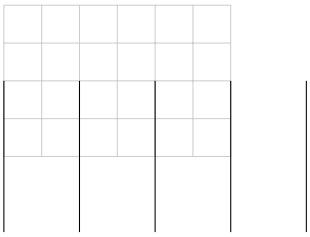
scope 环境

scope 环境就是作用域控制，一个局域环境，参数只影响内部，外部的参数也影响不进来，不过值得一提的是，定义的点外面也可以用。

scope 环境一个有用的特性的里面的 clip 命令不会影响到外面。

迭代语句

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\foreach \x in {0,1,...,4}
\draw[xshift=\x cm] (0,-1) -- (0,1);
\end{tikzpicture}
```



其中... 表示一直这样有规律下去生成迭代列表。迭代语句有很多用法，详见后面的具体例子。

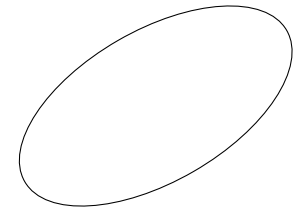
变形

xshift，x 坐标轴平移。**yshift**，y 坐标轴平移。**rotate**，旋转。注意 xshift 默认的单位并不是 cm，如果要单位是 cm 需要写出来。

旋转图形

后面加上可选项 **rotate=30** 即可，意思是图形逆时针旋转 30 度。

```
\begin{tikzpicture}
\draw (0,0)[rotate=30]  ellipse (2 and 1);
\end{tikzpicture}
```



反对称

xscale=-1 或者 yscale=-1 就刚好相对 y 轴或 x 轴反对称。

样式

style，特定图形的样式。定义一个样式比如 style001 如下：

```
style001/.style={color=red,fill=red!20}
```

原有样式修改

```
help lines/.append style=blue!50
```

附加之后最新的样式胜出。

样式带参数

red

blue

样式参数有默认值

default

blue

定义点

定义绝对点

```
\path (0,29) coordinate (top-left);
```

path 命令后面跟着坐标点，然后 coordinate 后面跟着这个点的名字。这里规范为 coordinate 命令后面跟着就是点的名字，node 命令后面跟着 node 的名字。

定义相对点

```
\path (top-left) ++(1,-2) coordinate (name-point);
```

++ 适合描述一连串逐渐变化的点，+ 适合描述多个点围绕着一个点变化的情况。

极座标

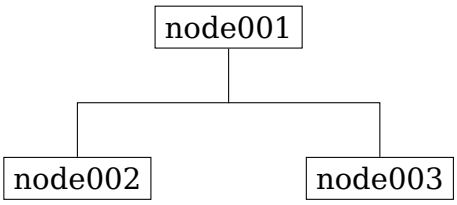
tikz 中的点也支持极座标表示，(30:1cm)，第一个参数是极座标里面的角度，第二个参数是半径。

node 命令中点的定义

test

从这里可以看到只要写上 draw 选项外面就会加上一个长方形，也就是 shape 的默认选项是 rectangle。如果你不希望外面有长方形，不写 draw 选项即可。

这里通过 node 命令定义了一个点，node001，在 (0,2) 那里。后面是可以使用的。



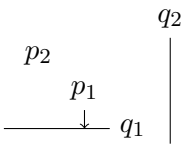
这里通过 **node cs:name=node003** 来获取之前那个 node 所在的点，然后通过 **anchor=north** 来定义那个 node 的接口在北边。除此之外的选项还有：**south**，**east**，**west**。这里 **|-**似乎是画垂直拐线的意思。上面的语法简写为可以 node002.north。

此外还有 **angle** 选项控制 node 接口的开口角度。

两个点定义出一个点

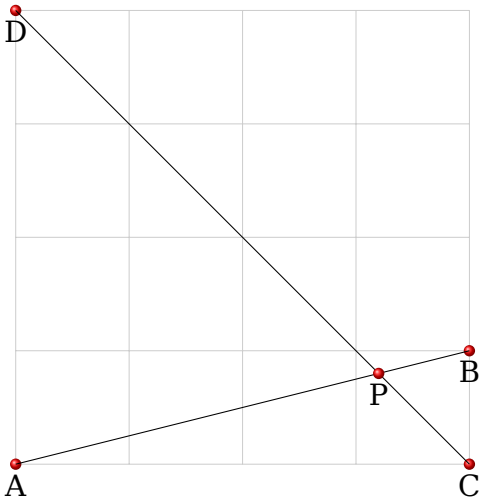
```
\begin{tikzpicture}
\node (p1) at (30:1) {$p_1$} ;
\node (p2) at (75:1) {$p_2$} ;
\draw (-0.2,0) -- (1.2,0) node[right] (xline) {$q_1$};
\draw (2,-0.2) -- (2,1.2) node[above] (yline) {$q_2$};

\draw[->] (p1) -- (p1 |- xline);
\end{tikzpicture}
```

这种形式 (p1 |- xline) 表示取第一个点的 x 和第二个点的 y 组成一个新的点。如果是 (p1 -| xline) 表示取第二个点的 x 和第一个点的 y 组成一个新的点。



两个 path 的交点

```
\begin{tikzpicture}[scale=3]
\draw[help lines] (0,0) grid (2,2);
\coordinate (A) at (0,0);
\coordinate (B) at (2,0.5);
\coordinate (C) at (2,0);
\coordinate (D) at (0,2);
\shade[ball color=red](A) circle (0.025) node[below] {A};
\shade[ball color=red](B) circle (0.025) node[below] {B};
\shade[ball color=red](C) circle (0.025) node[below] {C};
\shade[ball color=red](D) circle (0.025) node[below] {D};
\draw[name path=AB] (A) -- (B);
\draw[name path=CD] (C) -- (D);
\path[name intersections={of=AB and CD}] (intersection-1) coordinate (P);
\shade[ball color=red](P) circle (0.025) node[below] {P};
\end{tikzpicture}
```



这个例子用到了点的定义，点的标出，以及 path 交点的定义，要用到 library：**intersections**。有时候有些路径你不希望显示出来那么就用 path 命令来定义路径。

给新交点取名字

用 **by** 选项可以给画出来的交点取一个名字，默认的 intersection-1 之类的也可以使用。此外还可以加上选项：

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

点的运算

在进行下面说的数学运算之前需要加载 calc 宏包：

```
\usetikzlibrary{calc}
```

基本格式是：

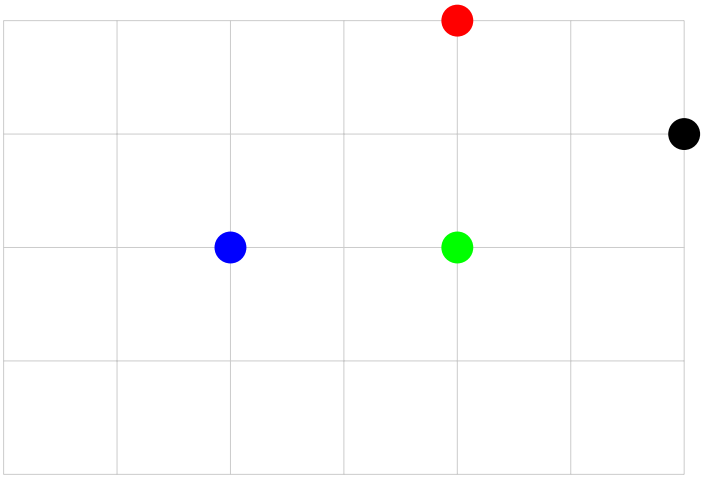
```
([options]$(一些运算)$)
```

这里$$ 表示这里有一些数学运算。里面的基本格式如下：

```
<factor>*<点><其他修饰>
```

```
\begin{tikzpicture}[scale=3]
\draw [help lines] (0,0) grid (3,2);
\fill [red] ($2*(1,1)$) circle (2pt);
\fill [green] (${1+1}*(1,0.5)$) circle (2pt);
\fill [blue] ($cos(0)*sin(90)*(1,1)$) circle (2pt);
\fill [black] (${3*(4-3)}*(1,0.5)$) circle (2pt);
\end{tikzpicture}
```

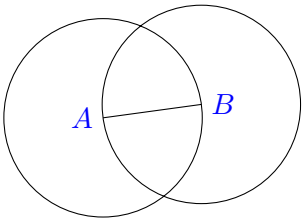
第一个红点是点 (1,1)，然后 x 和 y 都乘以 2 从而得到新点。后面情况类似，不同的是前面的乘法还可以加入更多的运算。



这里有点类似矢量运算计算出点的位置，前面计算出乘量因子，然后后面一个矢量偏移量。

计算两个点之间的距离

```
\begin{tikzpicture}
\coordinate[label=left:\textcolor{blue}{$A$}] (A)
 at ($(0,0) +0.1*(rand,rand)$) ;
\coordinate[label=right:\textcolor{blue}{$B$}] (B)
 at ($(1.25,0.25) +0.1*(rand,rand)$) ;

\draw (A) -- (B);

\draw  let
\p1 = ($ (B) - (A) $),
\n1 = {veclen(\x1,\y1)}
in
(A) circle (\n1)
(B) circle (\n1);

\end{tikzpicture}
```



let ... in ... 语句可以放在任何 path 命令的任何位置来控制变量的计算和定义。\p<*digit*> 定义的是点的变量，\n<*digit*> 定义的是数值的变量，后面可以跟数字从而定义多个变量。

任何点变量都可以用 \x<*digit*> 和 \y<*digit*> 来引用该点的 x 坐标和 y 坐标。

veclen 函数计算某个矢量的长度。

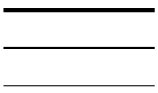
线条

path 路径是最基本的命令，draw 命令等价于`\path[draw]`，fill 命令等价于`\path[fill]`，filldraw 命令等价于`\path[draw,fill]`，其他 clip，shade 命令情况类似。

虚线和点线

线条除了之前说的 dashed 和 dotted 两种样式之外，还有 loosely dashed，densely dashed 和 loosely dotted，densely dotted。比如：- - - -　– – – –　– – – –，这是 dashed 的三种，下面是 dotted 的三种：. . . . . .　. . . . . .　. . . . . .。

线条的粗细



其他选项还有 **ultra thin**, **very thin**, **thin**, **semithick**, **thick**, **very thick** and **ultra thick** 还有 **help lines** 选项那种很淡灰的样式。

或者直接通过可选项 line width 来定义。

圆圆的拐角

线条延长

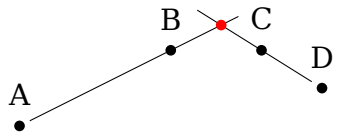
参考网站

shorten >=-0.4cm,shorten <=-0.4cm
可以通过类似上面的选项让两个点确定的线条延长，不过这种延长是不能用 intersection 方法处理的。其中 >= 表示到第二个点超过的部分，负值表示超过；然后 <= 表示到第一个点超过的部分，正值则缩回去了。

第二种线条延长的方法实际上是通过一个新的点来起作用的，这个点定义的语法的如下例所示：参考网站

```
\usetikzlibrary{calc,intersections}
\begin{tikzpicture}
\fill (0,0) circle [radius=2pt] node (A) [label=A] {};
\fill (2,1) circle [radius=2pt] node (B) [label=B] {};
\fill (3.2,1) circle [radius=2pt] node (C) [label=C] {};
```

```
\fill (4,0.5) circle [radius=2pt] node (D) [label=D] {};
\draw [name path=AB] (A) -- ($(B)!-1cm!(A)$);
\draw [name path=CD] (D) -- ($(C)!-1cm!(D)$);

\fill [red,name intersections={of={AB and CD}}] (intersection-1) circle [rad
\end{tikzpicture}
```

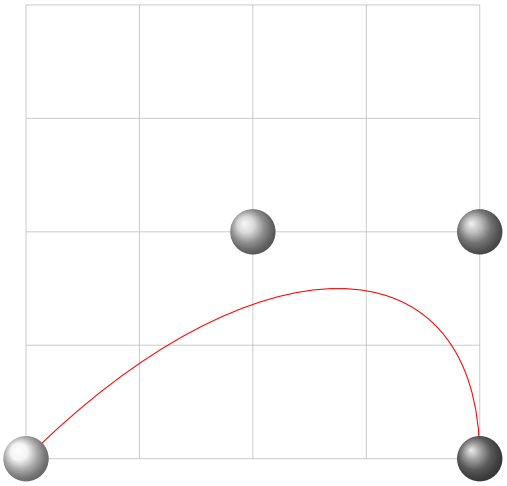


贝塞尔曲线

贝塞尔曲线是四个点画出一个曲线，具体我现在还不太清楚。其中第一个点是起点，第四个点终点，然后另外两个点是控制点。

```
\begin{tikzpicture}[scale=3]
\draw[help lines] (0,0) grid (2,2);
\draw[color=red] (0,0) .. controls (1,1) and (2,1) .. (2,0);
\shade[ball color=gray!10] (0,0) circle (0.1);
\shade[ball color=gray!40] (1,1) circle (0.1);
\shade[ball color=gray!70] (2,1) circle (0.1);
\shade[ball color=gray] (2,0) circle (0.1);
\end{tikzpicture}
```

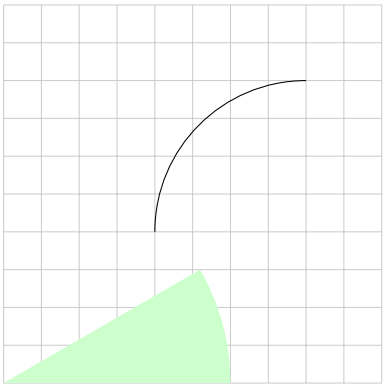
上面第 2 行代码就是画贝塞尔曲线的代码。



弧线

弧线反向

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (5,5);
\fill[green!20] (0,0) -- (3,0)
arc (0:30:3)  -- cycle;
\draw (2,2) arc (0:-90:-2);
\end{tikzpicture}
```



我们可以看到画弧线如果要中心点不是在左边而是在右边，那么可以通过让半径为负值和调整角度获得。其中角度的计算是顺时

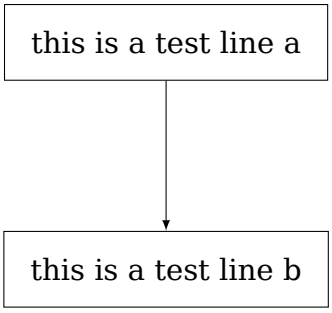
针的负值。

node 命令

node 命令主要用于插入文本，不过最好将其理解为接口。XeLaTeX 文档内部各个命令等都可以使用，然后外面包围一个形状，如 rectangle 长方形，circle 圆等。

```
\newcommand{\testlinea}{this is a test line a}
\newcommand{\testlineb}{this is a test line b}
\begin{tikzpicture}
%\fill[cyan] (0,0) circle  (1) ;
\node[shape=rectangle,draw,inner sep=10pt] at (0,0) (a) {\testlinea};
\node[shape=rectangle,draw,inner sep=10pt] at (0,-3) (b) {\testlineb};
\draw[-latex](a) -- (b);
\end{tikzpicture}
```

这里我们看到 LaTeX 里面自定义的命令是可以正常使用的，然后可选项 **shape** 的意思是外面包围的形状是长方形，**draw** 就是画这个形状是用的 draw 命令方法，比如 fill 就会填充。**inner sep** 控制外面的形状和内部文本之间的间距。然后 **at (0,0)** 控制整个图形的位置，然后 **(a)** 表示整个图形的名字，后面可以调用的，可以看作接口把。然后后面就是 LaTeX 的内容了。



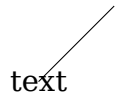
插入文本的位置

node 命令的可选项 **left**，**right**，**above**，**below** 用于控制插入文本的位置。此外还有 **above right**，**below right**，**above left** ，**below left** 等。

文本对齐控制

用 **align=left**，**align=right**，**align=center** 来控制。

在画图形的时候插入文本



在画图形的时候某个当前点下可以直接 node 接某个文本。

node 命令在 path 的任何位置都可插入，具体是 path 完成之后才绘制出 node 要插入的内容。

node 的 **inner sep** 选项调整 node 文本和外围的 shape 之间的间距。

node 的 **minimum size** 选项控制 node 没有文本时候外围 shape 的大小，装上文本可以更大。

node 可以通过 **at** 来具体控制 node 的位置，可以通过 **[below=of wating]** 这样的语句来让新的 node 相对其他 node 而存在。

node 旁插入标签

node 旁边加上标签，使用 **label** 选项，语法是：`label=above:$s\le3$`。除了常用的 above，below 等控制位置外，

还可以直接用 60 这样的数值控制位置，表示 node 圆圈逆时针旋转 60 度的那个位置。

所有标签的样式可以通过重定义 **every label** 样式来实现，

node 用箭头连接

`\draw [->] (critical.west) -- (enter critical.east);` 比如上面这个语句，critical 是 node 的名字，.west表示该 node 的 shape 的西边（也就是左边）出发。

弯曲箭头

用 **to** 语句更加灵活地画弯曲箭头，out 选项控制出来的角度，in 选项控制进去的角度。**bend right** 选项很有用，此外还有 **bend left** 选项，后面跟数值控制偏转量，一般 45。

箭头旁边加标签

to 语句后面跟个 node 就可以直接加上标签，表示在这个箭头 path 上加个 node。这种方法有一个 **swap** 可以让标签交换位置。

shape 穿过某个点

使用 through 包可以让 node 外的某个 shape 自定义穿过某个点，比如 **circle through**=(3,3)

node 的 scale 选项

参考网站

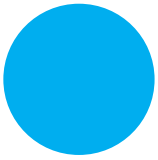
scale 是不改变 node 的大小的，可以用 **every node/.style={scale=0.6}** 等类似的语法来改变所有 node 的大小，或者 scale 用于单独的 node 命令改变某一个 node。

此外还有 **transform shape** 选项可以放这个 node 随着外部的 scale 命令一起变动。

fill 命令

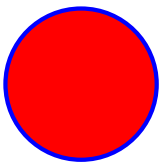
fill 命令就是填充某种颜色的形状，后面跟个 **color** 可选项设置填充的颜色，默认是黑色。比如画一个填充颜色的圆：

```
\fill[cyan] (0,0) circle  (1) ;
```



为了简单起见，draw 命令可以加上 fill 可选项，然后和上面类似的有：

```
\begin{tikzpicture}
\draw [color=blue,fill=red,ultra thick,] (0,0) circle (1);
\end{tikzpicture}
```

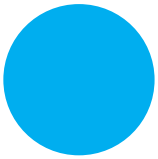


注意到线条的颜色和填充颜色的控制。

填充没有线条

如果你不希望有线条，那么使用 path 命令可以做到这点。

```
\begin{tikzpicture}
\path[fill=cyan] (0,0) circle  (1) ;
\end{tikzpicture}
```



filldraw 命令

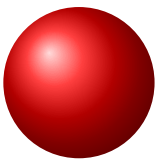
是 draw 命令和 fill 命令的结合。**fill=** 可选项调整填充的颜色，**draw=** 可选项调整画的线条的颜色。

shade 命令

shade 命令和 fill 命令的区别就是填充的颜色是渐变的，其他类似。

其可选项有 **top color 和 bottom color** 表示上下渐变的颜色，**left color 和 right color**，**innercolor 和 outercolor**，这些是配对的。此外还有 **ball color** 让颜色渐变像一个有光照的球。

小红球



tikz 中的随机数

rand 产生一个随机数，范围在 -1 ～ 1 之间。

tikz 高级知识

文本中的图片

wherever you want

clip 命令

clip 就是剪切的意思，就是通过 clip 命令按照某个形状来剪切，外面的图形都不保留，可以跟一个可选项 **draw**，这样剪切的时候同时画出了这个形状。

path 路径闭合

任何构建的 path 最后都可以通过 **-- cycle** 将其闭合起来。

插入其他图片

在 node 里面用 includegraphics 命令可以插入图片。参考网站

baseline 选项

这个主要控制 inline 模式下图片的位置，默认 **baseline=0pt**。

tikzmark

pgfplots 宏包介绍

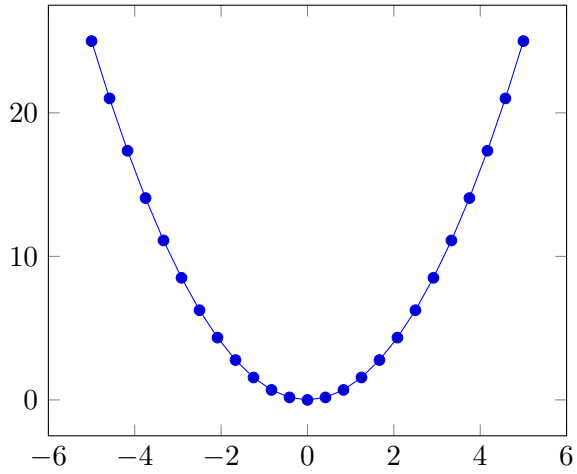
老实说 pgfplots 宏包真的编写的很好，有时甚至画一个基本的坐标轴都懒得动用其他宏包命令了，直接调用一个 axis 环境和进行一些简单的优化即可。当然就作为坐标轴作图可能总是用 pgfplots 宏包可能会稍显单调，但如果要求不是特别高的确实用 pgfplots 宏包会基于坐标轴的各个图形非常的称心如意。

这个宏包内容还是挺多的，先就最基本的内容整理一下。

简单了解

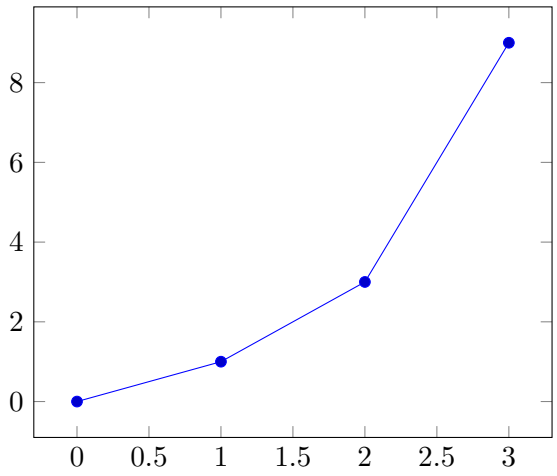
直接画函数

```
\begin{tikzpicture}
\begin{axis}
\addplot {x^2};

\end{axis}
\end{tikzpicture}
```



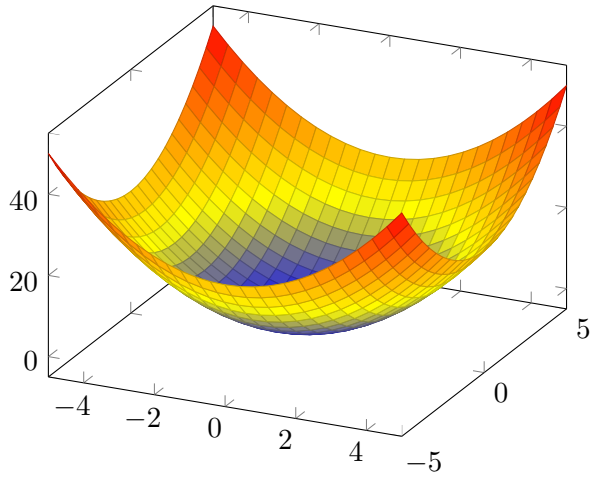
根据数据点来

```
\begin{tikzpicture}
\begin{axis}
\addplot coordinates
{(0,0)
(1,1)
(2,3)
(3,9)};

\end{axis}
\end{tikzpicture}
```



画三维数据图

```
\begin{tikzpicture}
\begin{axis}
\addplot3[surf] {x^2+y^2};

\end{axis}
\end{tikzpicture}
```

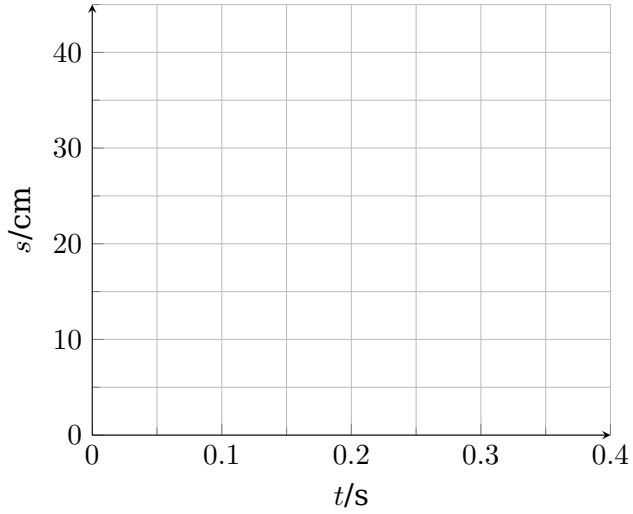


纯坐标轴优化

首先我们来看一个简单的例子，然后就这些选项做出说明。

```
\begin{tikzpicture}
\begin{axis}[grid=major,
xtick={0,0.1,0.2,0.3,0.4},
ytick={0,10,20,30,40},
%extra x ticks={1.5},
axis x line=bottom,
axis y line =left,
%legend pos=outer north east,
```

```
xmin=0,xmax=0.4,ymin=0,ymax=45,
minor tick num=1,
tick align=inside,
xlabel=$t$/s,
ylabel=$s$/cm,
grid=both]

\end{axis}
\end{tikzpicture}
```



x 坐标轴标记指定

参考网站

默认的看好不好用，**xtick=data** 选项会让所有输入的数据点都有标记。

xtick={1,2,3,5}：指定的某些 x 轴显示。

额外的坐标轴标记

用 **extra x ticks={1.25}** 来显示额外的坐标轴标记。

坐标轴范围

xmin，xmax，ymin，ymax 选项使用来配置坐标轴范围的。

画多个线条

用 **addlegendentry** 命令来为某个图例加上名字。

网格

加上选项 **grid=major** 即加上网格。

电路图

用 tikz 绘制一般电路图的解决方案可以说是完美，首先需要加载宏包：

```
\usetikzlibrary{circuits.ee.IEC}
```

然后电路基本的组成单元是以 node 的某个特定样式的形式引入的，比如电池：

```
\node[battery] (battery) at(0,3) {};
```

下面列出常用的符号列表

电路基本符号

表 5.1: 电路基本符号

选项	说明	演示
battery	电池	
bulb	灯泡	
make contact	开关	
make contact	开关另一种形式[a]	
resistor	电阻[b]	
contact	电线交点	
current direction	to 路径上加上电流方向[c]	

[a] 额外选项 [set make contact graphic= var make contact IEC graphic]

[b] 加上选项 [ohm=20k] 则上面写上电阻数值

[c] 如果是 [**current direction'**] 则方向反向。

连线问题

各个元器件之间的连线除了一般的- -连直线外，还可以通过- |或者| -来处理垂直拐线的问题，其中- | 你可以理解为从第一个点先横着走再竖着走，而| -你可以理解为先从第一个点竖着走再横着走。

翻转问题

四个基本的选项 [**point up point down point left point right**]，分别是朝上，朝下，朝左和朝右。

其他复杂的角度的处理方法不是用 rotate 选项，而是在路径上

加上上面的电路符号选项，这样那些元器件会自动跟随路径对齐的。

电压表和电流表

电压表电流表实际上 **circuit ee IEC** 里面也有，不过不是我们（中国大陆）初高中物理书上常见的那种，而我们其实可以很简单的用 node 命令就画出了类似书本上的那种符号：

```
\node[draw,circle,inner sep =1pt] (A) at (-2,1.5) {\footnotesize A};
\node[draw,circle,inner sep =1pt] (V) at (1,4) {\footnotesize V};
```

info 选项

这些 node 命令都支持`info` 选项，也就是旁边加上标签信息。此外还有一个`info'` 选项会让信息位置反向，有时很有用。如果位置还是不满意那就只好用`info=angle:text` 的形式了，其中 angle 填上你想要的角度。一般通过角度控制能够达到满意的效果了吧。

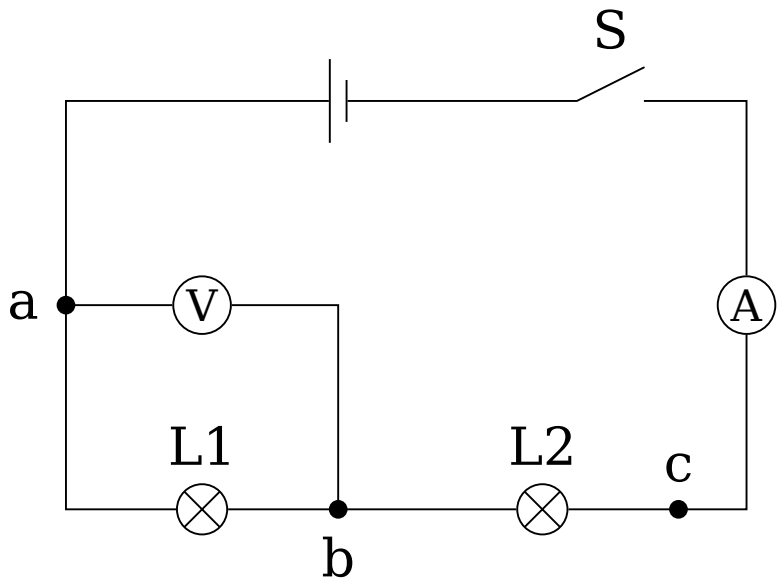
此外还有一个 **info sloped** 选项，这样文字会跟随路径出现旋转效果。

两个综合性的例子作为演示

例子一

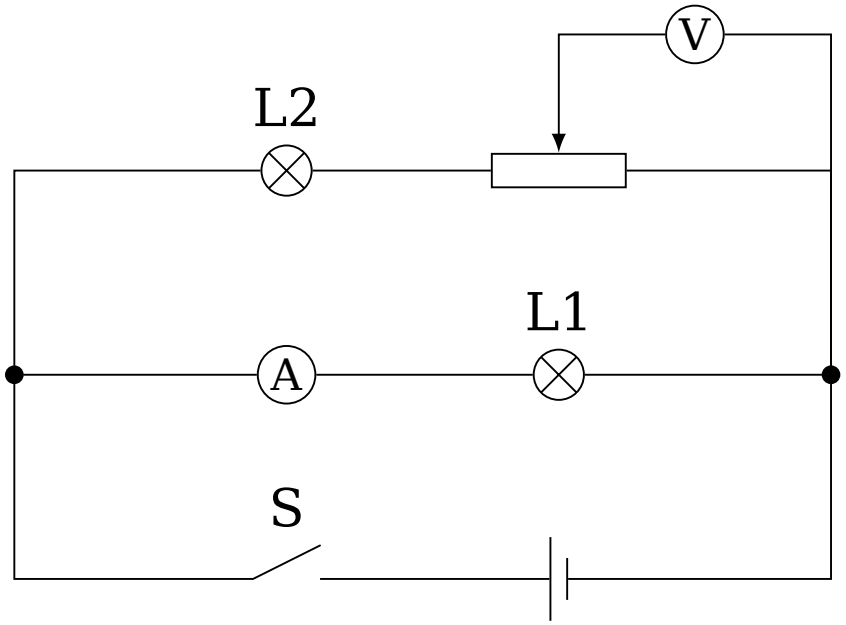
```
\usetikzlibrary{circuits.ee.IEC}

\begin{tikzpicture}[circuit ee IEC]
\node[battery] (battery) at(0,3) {};
\node[make contact,info=S] (contact1) at(2,3) {};
```

```
\draw (battery) -- (contact1);
\node[draw,circle,inner sep =1pt] (A) at (3,1.5) {\footnotesize A};
\draw (contact1) -| (A);
\node[bulb,info=L1] (L1) at (-1,0){};
\node[bulb,info=L2] (L2) at (1.5,0){};
\node[contact,info=c] (c) at (2.5,0) {};
\draw (L2) -| (A);
\draw (L1) -- (L2);
\node[contact,info’=b] (b) {};
\node[contact,info=180:a] (a) at(-2,1.5) {};
\node[draw,circle,inner sep =1pt] (V) at (-1,1.5) {\footnotesize V};
\draw (a) -- (V);
\draw (V) -| (b);
\draw (L1) -| (a);
\draw (a) |- (battery);
\end{tikzpicture}
```



例子二

```
\usetikzlibrary{circuits.ee.IEC}

\begin{tikzpicture}[circuit ee IEC]
\node[battery] (battery) at(0,0) {};
\node[make contact,info=S] (S) at(-2,0) {};
\draw (battery) -- (S);
\node[draw,circle,inner sep =1pt] (A) at (-2,1.5) {\footnotesize A};
\node[bulb,info=L1] (L1) at (0,1.5){};
\draw (A) -- (L1);
\node[resistor] (R) at (0,3) {};
\node[bulb,info=L2] (L2) at (-2,3){};
\draw (L2) -- (R);
\node[contact] (a) at(-4,1.5) {};
\node[contact] (b) at(2,1.5) {};
\draw (S) -| (a);
\draw (battery) -| (b);
\draw (a) |- (L2);
\draw (a) -- (A);
\node[draw,circle,inner sep =1pt] (V) at (1,4) {\footnotesize V};
\draw[-latex] (V) -| (R.north);
\draw (L1) -- (b);
\draw (b) |- (R);
\draw (b) |- (V) ;

\end{tikzpicture}
```



更多的例子

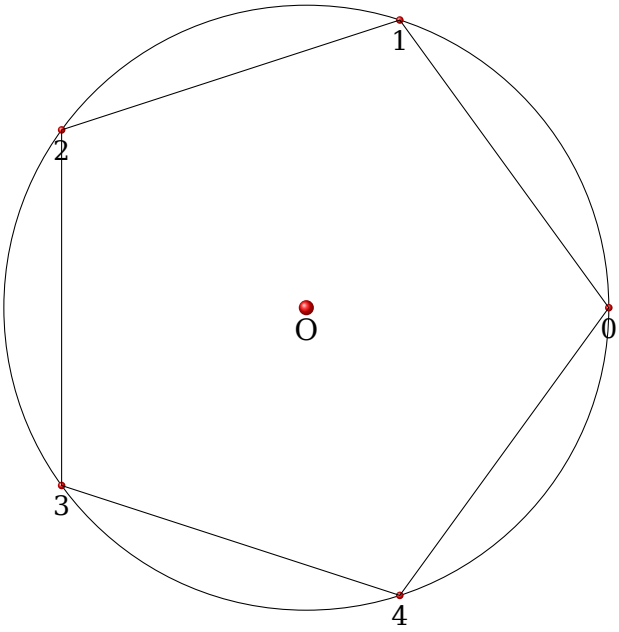
更多的例子相关技巧请参看文件夹 [*tikz* 制图]→[电路图] 文件夹里面的内容。

其他

tikz 的一些例子

画正多边形

```
\begin{tikzpicture}
\draw (0,0) circle (4) ;
\coordinate (O) at (0,0);
\shade[ball color=red](O) circle (0.1) node[below] {O};
\def\n{5}
\pgfmathsetmacro\i{\n-1}
\foreach \x in {0,...,\i}
{
\def\pointname{\x}
\coordinate (\pointname) at ($(0,0) +(\x*360/\n:4cm)$)  ;
\shade[ball color=red](\pointname) circle (0.05) node[below] {\small \x};
}

\draw (0)
\foreach \x in {0,...,\i}
{ -- (\x) } -- cycle;

\end{tikzpicture}
```

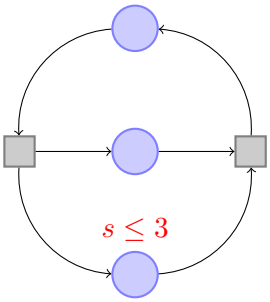


这个例子核心内容是批量定义点和点的运算，把这个弄懂了，后面 tikz 的核心大门就为你打开了，然后很多图形都可以用简洁的命令生成出来了。

多个 node 连接

```
\usetikzlibrary{positioning}
\tikzset{place/.style={circle,draw=blue!50,fill=blue!20,
thick,inner sep=0pt,minimum size=6mm}}
\tikzset{transition/.style={rectangle,draw=black!50,
fill=black!20,thick,inner sep=0pt,minimum size=4mm}}
\tikzset{every label/.style=red}
\begin{tikzpicture}[bend angle=45]
\node[place] (waiting)  {};
\node[place] (critical) [below=of waiting] {};
\node[place](semaphore) [below=of critical,label=above:$s\le3$] {};
\node[transition](leave critical) [right=of critical]{};
\node[transition] (enter critical)[left=of critical]{};
```

```
\draw [->] (enter critical) to (critical);
\draw [->] (waiting) to [bend right] (enter critical);
\draw [->] (enter critical) to [bend right] (semaphore);
\draw [->] (semaphore) to [bend right] (leave critical);
\draw [->] (critical) to (leave critical);
\draw [->] (leave critical) to [bend right] (waiting);
\end{tikzpicture}
```



这个例子需要加载 positioning 包，这个例子很好地展示了多个 node 和用箭头连接来表示他们关系的图形如何绘制。

几何第一个例子

为了减少文档大小，我编写的 tikz 的例子都放入文件夹 [*tikz* 制图] 里面了。

```
\begin{tikzpicture}
\coordinate[label=left:\textcolor{blue}{$A$}] (A)
at ($(0,0) +0.1*(rand,rand)$) ;
\coordinate[label=right:\textcolor{blue}{$B$}] (B)
at ($(1.25,0.25) +0.1*(rand,rand)$) ;

\draw [name path=A--B] (A) -- (B);
```

```
\node [name path=D,draw,circle through=(B),label=left:$D$] at (A) {};
\node [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 [red] (A) -- (C);
\draw [red] (B) -- (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=1,label=-45:$F$] at (F) {};

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

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

\end{tikzpicture}
```