13 Specifying Coordinates 指定坐标

13.1 Overview 概述

A *coordinate* is a position on the canvas on which your picture is drawn. Ti*k*Z uses a special syntax for specifying coordinates. Coordinates are always put in round brackets. The general syntax is `([`⟨*options*⟩`]`⟨*coordinate specification*⟩`)`.

坐标是在绘图画布上的位置。Ti*k*Z 使用一种特殊的语法来指定坐标。坐标总是放在圆括号中。一般的语法格式为 `([`⟨*选项*⟩`]`⟨*坐标规范*⟩`)`。

The ⟨*coordinate specification*⟩ specifies coordinates using one of many different possible *coordinate systems*. Examples are the Cartesian coordinate system or polar coordinates or spherical coordinates. No matter which coordinate system is used, in the end, a specific point on the canvas is represented by the coordinate.

⟨*坐标规范*⟩ 使用许多不同的坐标系之一来指定坐标。例如，笛卡尔坐标系、极坐标或球面坐标。无论使用哪种坐标系，最终都会用坐标表示画布上的特定点。

There are two ways of specifying which coordinate system should be used:

有两种指定使用的坐标系的方法：

Explicitly You can specify the coordinate system explicitly. To do so, you give the name of the coordinate system at the beginning, followed by `cs:`, which stands for "coordinate system", followed by a specification of the coordinate using the key–value syntax. Thus, the general syntax for ⟨*coordinate specification*⟩ in the explicit case is `(`⟨*coordinate system*⟩ `cs:`⟨*list of key–value pairs specific to the coordinate system*⟩`)`.

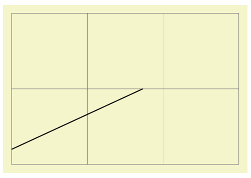
显式 可以显式指定坐标系。为此，在开头给出坐标系的名称，后面跟着 `cs:`，其代表 "坐标系"，然后使用键-值语法来指定坐标。因此，显式情况下 ⟨*坐标规范*⟩ 的一般语法为 `(`⟨*坐标系*⟩ `cs:`⟨*坐标系特定的键-值对列表*⟩`)`。

Implicitly The explicit specification is often too verbose when numerous coordinates should be given. Because of this, for the coordinate systems that you are likely to use often a special syntax is provided. Ti*k*Z will notice when you use a coordinate specified in a special syntax and will choose the correct coordinate system automatically.

隐式 当需要给出大量坐标时，显式指定通常太冗长。因此，对于经常使用的坐标系，提供了一种特殊的语法。当使用特殊语法指定坐标时，Ti*k*Z 会自动识别并选择正确的坐标系。

Here is an example in which explicit the coordinate systems are specified explicitly:

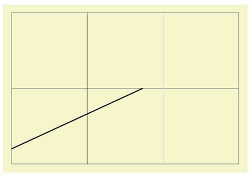
下面是一个显示显式坐标系的示例：



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (canvas cs:x=0cm,y=2mm)
     -- (canvas polar cs:radius=2cm,angle=30);
\end{tikzpicture}
```

In the next example, the coordinate systems are implicit:

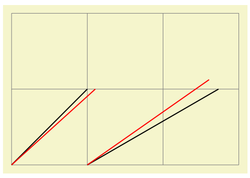
在下一个示例中，坐标系是隐式指定的：



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (0cm,2mm) -- (30:2cm);
\end{tikzpicture}
```

It is possible to give options that apply only to a single coordinate, although this makes sense for transformation options only. To give transformation options for a single coordinate, give these options at the beginning in brackets:

可以为单个坐标提供仅适用于变换选项的选项，尽管这仅对于变换选项有意义。要为单个坐标提供变换选项，请在方括号中的开头给出这些选项：



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw      (0,0) -- (1,1);
  \draw[red] (0,0) -- ([xshift=3pt] 1,1);
  \draw      (1,0) -- +(30:2cm);
  \draw[red] (1,0) -- +([shift=(135:5pt)] 30:2cm);
\end{tikzpicture}
```

13.2 Coordinate Systems
坐标系

13.2.1 Canvas, XYZ, and Polar Coordinate Systems
画布、XYZ 和极坐标系

Let us start with the basic coordinate systems.
让我们从基本的坐标系开始。

Coordinate system canvas

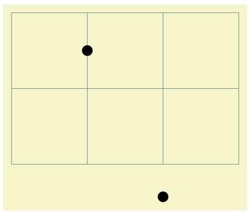
The simplest way of specifying a coordinate is to use the `canvas` coordinate system. You provide a dimension d_x using the `x=` option and another dimension d_y using the `y=` option. The position on the canvas is located at the position that is d_x to the right and d_y above the origin.
指定坐标的最简单方法是使用 `canvas` 坐标系。使用 `x=` 选项提供一个维度 d_x，使用 `y=` 选项提供另一个维度 d_y。画布上的位置位于距离原点向右 d_x、向上 d_y 的位置。

`/tikz/cs/x=`⟨*dimension*⟩ (no default, initially `0pt`)

Distance by which the coordinate is to the right of the origin. You can also write things like `1cm+2pt` since the mathematical engine is used to evaluate the ⟨*dimension*⟩.
坐标相对于原点向右的距离。可以写成 `1cm+2pt` 这样的形式，因为使用数学引擎来评估 ⟨ 尺寸⟩。

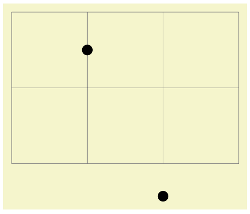
`/tikz/cs/y=`⟨*dimension*⟩ (no default, initially `0pt`)

Distance by which the coordinate is above the origin.
坐标相对于原点向上的距离。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);

  \fill (canvas cs:x=1cm,y=1.5cm)    circle (2pt);
  \fill (canvas cs:x=2cm,y=-5mm+2pt) circle (2pt);
\end{tikzpicture}
```

To specify a coordinate in the coordinate system implicitly, you use two dimensions that are separated by a comma as in `(0cm,3pt)` or `(2cm,\textheight)`.
要隐式指定坐标系，可以使用用逗号分隔的两个维度，如 `(0cm,3pt)` 或 `(2cm,\textheight)`。



```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);

  \fill (1cm,1.5cm)    circle (2pt);
  \fill (2cm,-5mm+2pt) circle (2pt);
\end{tikzpicture}
```

Coordinate system xyz

The `xyz` coordinate system allows you to specify a point as a multiple of three vectors called the x-, y-, and z-vectors. By default, the x-vector points 1cm to the right, the y-vector points 1cm upwards, but this can be changed arbitrarily as explained in Section ??. The default z-vector points to $(-3.85\text{mm}, -3.85\text{mm})$.

xyz 坐标系允许您将点指定为三个向量（x 向量、y 向量和 z 向量）的倍数。默认情况下，x 向量指向右侧 1cm，y 向量指向上方 1cm，但可以根据需要进行任意更改，如第 ?? 节中所述。默认的 z 向量指向 $(-3.85\text{mm}, -3.5\text{mm})$。

To specify the factors by which the vectors should be multiplied before being added, you use the following three options:

要指定向量相加之前的倍数，可以使用以下三个选项：

/tikz/cs/x=⟨*factor*⟩ (no default, initially 0)

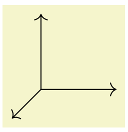
Factor by which the x-vector is multiplied.

/tikz/cs/y=⟨*factor*⟩ (no default, initially 0)

Works like x.

/tikz/cs/z=⟨*factor*⟩ (no default, initially 0)

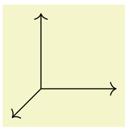
Works like x.



```
\begin{tikzpicture}[->]
  \draw (0,0) -- (xyz cs:x=1);
  \draw (0,0) -- (xyz cs:y=1);
  \draw (0,0) -- (xyz cs:z=1);
\end{tikzpicture}
```

This coordinate system can also be selected implicitly. To do so, you just provide two or three comma-separated factors (not dimensions).

这个坐标系也可以隐式选择。为此，只需提供两个或三个以逗号分隔的因子（不是尺寸）。



```
\begin{tikzpicture}[->]
  \draw (0,0) -- (1,0);
  \draw (0,0) -- (0,1,0);
  \draw (0,0) -- (0,0,1);
\end{tikzpicture}
```

Note: It is possible to use coordinates like (1,2cm), which are neither canvas coordinates nor xyz coordinates. The rule is the following: If a coordinate is of the implicit form (⟨x⟩,⟨y⟩), then ⟨x⟩ and ⟨y⟩ are checked, independently, whether they have a dimension or whether they are dimensionless. If both have a dimension, the canvas coordinate system is used. If both lack a dimension, the xyz coordinate system is used. If ⟨x⟩ has a dimension and ⟨y⟩ has not, then the sum of two coordinate (⟨x⟩,0pt) and (0,⟨y⟩) is used. If ⟨y⟩ has a dimension and ⟨x⟩ has not, then the sum of two coordinate (⟨x⟩,0) and (0pt,⟨y⟩) is used.

*注意：*可以使用诸如 (1,2cm) 这样的坐标，它们既不是 canvas 坐标也不是 xyz 坐标。规则如下：如果坐标的隐式形式为 (⟨x⟩,⟨y⟩)，则会独立检查 ⟨x⟩ 和 ⟨y⟩ 是否具有尺寸或无尺寸。如果两者都有尺寸，则使用 canvas 坐标系。如果两者都没有尺寸，则使用 xyz 坐标系。如果 ⟨x⟩ 有尺寸而 ⟨y⟩ 没有，则使用两个坐标 (⟨x⟩,0pt) 和 (0,⟨y⟩) 的和。如果 ⟨y⟩ 有尺寸而 ⟨x⟩ 没有，则使用两个坐标 (⟨x⟩,0) 和 (0pt,⟨y⟩) 的和。

Note furthermore: An expression like (2+3cm,0) does *not* mean the same as (2cm+3cm,0). Instead, if ⟨x⟩ or ⟨y⟩ internally uses a mixture of dimensions and dimensionless values, then all dimensionless values are "upgraded" to dimensions by interpreting them as pt. So, 2+3cm is the same dimension as 2pt+3cm.

*此外注意：*表达式 (2+3cm,0) 与 (2cm+3cm,0) 不相同。如果 ⟨x⟩ 或 ⟨y⟩ 内部使用尺寸和无尺寸值的混合，则所有无尺寸值都会通过将其解释为 pt 而被"升级"为尺寸。因此，2+3cm 和 2pt+3cm 表示相同的尺寸。

Coordinate system canvas polar

The canvas polar coordinate system allows you to specify polar coordinates. You provide an angle using the angle= option and a radius using the radius= option. This yields the point on the canvas that is at the given radius distance from the origin at the given degree. An angle of zero degrees to the right, a degree of 90 upward.

canvas polar 坐标系允许您指定极坐标。您可以使用angle= 选项提供角度，并使用radius= 选项提供半径。这将给出位于给定角度处距离原点给定半径距离的画布上的点。角度为零度表示向右，90 度表示向上。

/tikz/cs/angle=⟨*degrees*⟩ (no default)

The angle of the coordinate. The angle must always be given in degrees and should be between −360 and 720.

坐标的角度。角度必须始终以度为单位，并且应在 −360 到 720 之间。

`/tikz/cs/radius=`⟨*dimension*⟩ (no default)

The distance from the origin.

距离原点的距离。

`/tikz/cs/x radius=`⟨*dimension*⟩ (no default)

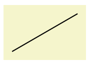
A polar coordinate is, after all, just a point on a circle of the given ⟨*radius*⟩. When you provide an x-radius and also a y-radius, you specify an ellipse instead of a circle. The `radius` option has the same effect as specifying identical `x radius` and `y radius` options.

极坐标实际上只是给定⟨*radius*⟩ 的圆上的一个点。当您提供 x 半径和 y 半径时，您指定的是椭圆而不是圆。`radius` 选项与指定相同的`x radius` 和`y radius` 选项具有相同的效果。

`/tikz/cs/y radius=`⟨*dimension*⟩ (no default)

Works like `x radius`.

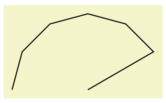
与`x radius` 相同。



```
\tikz \draw (0,0) -- (canvas polar cs:angle=30,radius=1cm);
```

The implicit form for canvas polar coordinates is the following: you specify the angle and the distance, separated by a colon as in `(30:1cm)`.

画布极坐标的隐式形式如下：您使用冒号分隔的角度和距离来指定，例如 `(30:1cm)`。



```
\tikz \draw    (0cm,0cm) -- (30:1cm) -- (60:1cm) -- (90:1cm)
            -- (120:1cm) -- (150:1cm) -- (180:1cm);
```

Two different radii are specified by writing `(30:1cm and 2cm)`.

通过编写 `(30:1cm and 2cm)` 可以指定两个不同的半径。

For the implicit form, instead of an angle given as a number you can also use certain words. For example, `up` is the same as `90`, so that you can write `\tikz \draw (0,0) -- (2ex,0pt) -- +(up:1ex);` and get ⌐. Apart from `up` you can use `down`, `left`, `right`, `north`, `south`, `west`, `east`, `north east`, `north west`, `south east`, `south west`, all of which have their natural meaning.

对于隐式形式，您可以使用某些单词而不是数字来表示角度。例如，`up` 等同于 `90`，因此您可以编写 `\tikz \draw (0,0) -- (2ex,0pt) -- +(up:1ex);` 并获得 ⌐。除了 `up`，您还可以使用 `down`、`left`、`right`、`north`、`south`、`west`、`east`、`north east`、`north west`、`south east` 和 `south west`，它们都有其自然的含义。

Coordinate system `xyz polar`

This coordinate system work similarly to the `canvas polar` system. However, the radius and the angle are interpreted in the xy-coordinate system, not in the canvas system. More detailed, consider the circle or ellipse whose half axes are given by the current x-vector and the current y-vector. Then, consider the point that lies at a given angle on this ellipse, where an angle of zero is the same as the x-vector and an angle of 90 is the y-vector. Finally, multiply the resulting vector by the given radius factor. Voilà.

这个坐标系与`canvas polar` 系统类似。但是，半径和角度是在 xy 坐标系中解释的，而不是在画布坐标系中解释的。更详细地说，考虑由当前 x 向量和当前 y 向量给出的圆或椭圆。然后，考虑位于此椭圆上给定角度处的点，其中角度零与 x 向量相同，角度 90 与 y 向量相同。最后，将得到的向量乘以给定的半径因子。Voil'a。

`/tikz/cs/angle=`⟨*degrees*⟩ (no default)

The angle of the coordinate interpreted in the ellipse whose axes are the x-vector and the y-vector.

坐标的角度，解释为以 x 向量和 y 向量为轴的椭圆中的角度。

/tikz/cs/radius=⟨*factor*⟩ (no default)

A factor by which the x-vector and y-vector are multiplied prior to forming the ellipse.

在形成椭圆之前，x 向量和 y 向量乘以的因子。

/tikz/cs/x radius=⟨*dimension*⟩ (no default)

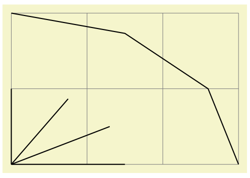
A specific factor by which only the x-vector is multiplied.

仅将 x 向量乘以的特定因子。

/tikz/cs/y radius=⟨*dimension*⟩ (no default)

Works like `x radius`.

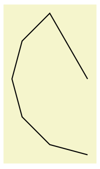
与`x radius` 相同。



```
\begin{tikzpicture}[x=1.5cm,y=1cm]
  \draw[help lines] (0cm,0cm) grid (3cm,2cm);

  \draw (0,0) -- (xyz polar cs:angle=0,radius=1);
  \draw (0,0) -- (xyz polar cs:angle=30,radius=1);
  \draw (0,0) -- (xyz polar cs:angle=60,radius=1);
  \draw (0,0) -- (xyz polar cs:angle=90,radius=1);

  \draw (xyz polar cs:angle=0,radius=2)
     -- (xyz polar cs:angle=30,radius=2)
     -- (xyz polar cs:angle=60,radius=2)
     -- (xyz polar cs:angle=90,radius=2);
 \end{tikzpicture}
```

The implicit version of this option is the same as the implicit version of `canvas polar`, only you do not provide a unit.

这个选项的隐式版本与`canvas polar` 的隐式版本相同，只是您不需要提供单位。



```
\tikz[x={(0cm,1cm)},y={(-1cm,0cm)}]
  \draw  (0,0) -- (30:1) -- (60:1) -- (90:1)
             -- (120:1) -- (150:1) -- (180:1);
```

Coordinate system xy polar

This is just an alias for `xyz polar`, which some people might prefer as there is no z-coordinate involved in the `xyz polar` coordinates.

这只是`xyz polar` 的别名，一些人可能更喜欢使用它，因为`xyz polar` 坐标中没有涉及 z 坐标。

13.2.2 Barycentric Systems 重心系

In the barycentric coordinate system a point is expressed as the linear combination of multiple vectors. The idea is that you specify vectors v_1, v_2, ..., v_n and numbers α_1, α_2, ..., α_n. Then the barycentric coordinate specified by these vectors and numbers is

在重心坐标系中，一个点被表示为多个向量的线性组合。思路是您指定向量 v_1、v_2、...、v_n 和数字 α_1、α_2、...、α_n。然后由这些向量和数字指定的重心坐标为

$$\frac{\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n}{\alpha_1 + \alpha_2 + \cdots + \alpha_n}$$

The `barycentric cs` allows you to specify such coordinates easily.

`barycentric cs` 允许您轻松指定这样的坐标。

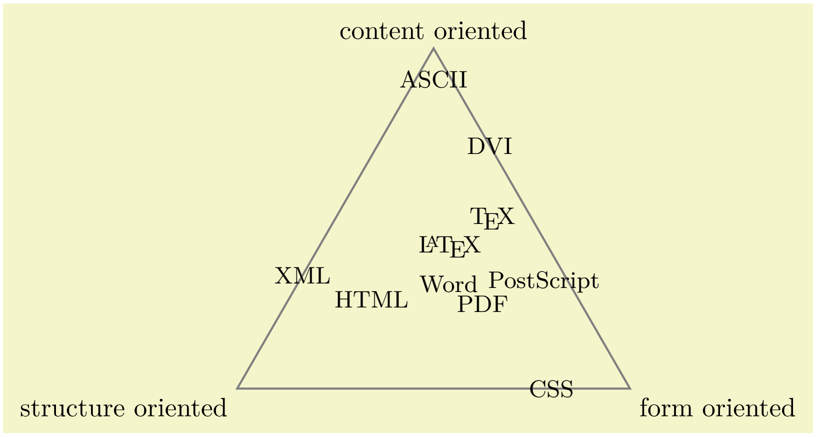
Coordinate system barycentric

For this coordinate system, the ⟨*coordinate specification*⟩ should be a comma-separated list of expressions of the form ⟨*node name*⟩=⟨*number*⟩. Note that (currently) the list should not contain any spaces before or after the ⟨*node name*⟩ (unlike normal key–value pairs).

对于这个坐标系，⟨*coordinate specification*⟩ 应该是一个由逗号分隔的表达式列表，每个表达式的格式为⟨*node name*⟩=⟨*number*⟩。请注意（目前）列表中不应在⟨*node name*⟩ 之前或之后包含任何空格（与常规键值对不同）。

The specified coordinate is now computed as follows: Each pair provides one vector and a number. The vector is the `center` anchor of the ⟨*node name*⟩. The number is the ⟨*number*⟩. Note that (currently) you cannot specify a different anchor, so that in order to use, say, the `north` anchor of a node you first have to create a new coordinate at this north anchor. (Using for instance `\coordinate(mynorth) at (mynode.north);`.)

现在，指定的坐标计算如下：每对提供一个向量和一个数字。向量是⟨*node name*⟩ 的`center` 锚点。数字是⟨*number*⟩。请注意（目前）您不能指定不同的锚点，因此为了使用节点的`north` 锚点，您首先必须在该北锚点处创建一个新的坐标（例如，使用`\coordinate(mynorth) at (mynode.north);`）。



```
\begin{tikzpicture}
  \coordinate (content)   at (90:3cm);
  \coordinate (structure) at (210:3cm);
  \coordinate (form)      at (-30:3cm);

  \node [above]       at (content)   {content oriented};
  \node [below left]  at (structure) {structure oriented};
  \node [below right] at (form)      {form oriented};

  \draw [thick,gray] (content.south) -- (structure.north east) -- (form.north west) -- cycle;

  \small
  \node at (barycentric cs:content=0.5,structure=0.1 ,form=1)    {PostScript};
  \node at (barycentric cs:content=1  ,structure=0   ,form=0.4)  {DVI};
  \node at (barycentric cs:content=0.5,structure=0.5 ,form=1)    {PDF};
  \node at (barycentric cs:content=0  ,structure=0.25,form=1)    {CSS};
  \node at (barycentric cs:content=0.5,structure=1   ,form=0)    {XML};
  \node at (barycentric cs:content=0.5,structure=1   ,form=0.4)  {HTML};
  \node at (barycentric cs:content=1  ,structure=0.2 ,form=0.8)  {\TeX};
  \node at (barycentric cs:content=1  ,structure=0.6 ,form=0.8)  {\LaTeX};
  \node at (barycentric cs:content=0.8,structure=0.8 ,form=1)    {Word};
  \node at (barycentric cs:content=1  ,structure=0.05,form=0.05) {ASCII};
\end{tikzpicture}
```

13.2.3 Node Coordinate System 节点坐标系

In PGF and in Ti*k*Z it is quite easy to define a node that you wish to reference at a later point. Once you have defined a node, there are different ways of referencing points of the node. To do so, you use the following coordinate system:

在 PGF 和 Ti*k*Z 中，定义一个稍后要引用的节点非常容易。一旦定义了一个节点，就有不同的方法来引用节点的点。为此，您可以使用以下坐标系：

Coordinate system `node`

This coordinate system is used to reference a specific point inside or on the border of a previously defined node. It can be used in different ways, so let us go over them one by one.

这个坐标系用于引用先前定义的节点内部或边界上的特定点。它可以以不同的方式使用，我们逐一介绍它们。

You can use three options to specify which coordinate you mean:

您可以使用三个选项来指定您指的是哪个坐标：

`/tikz/cs/name=`⟨*node name*⟩ (no default)

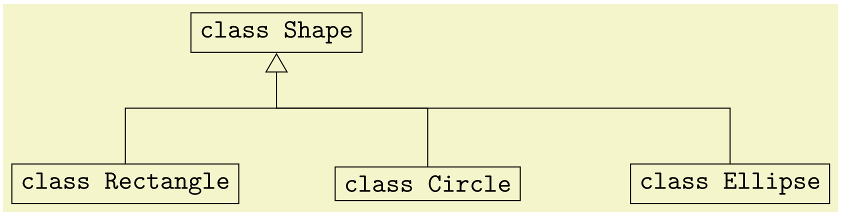
Specifies the node that you wish to use to specify a coordinate. The ⟨*node name*⟩ is the name that was previously used to name the node using the `name=`⟨*node name*⟩ option or the special node name syntax.

指定您希望用于指定坐标的节点。⟨*node name*⟩ 是先前使用`name=`⟨*node name*⟩ 选项或特殊节点名称语法命名节点时使用的名称。

`/tikz/anchor=`⟨*anchor*⟩ (no default)

Specifies an anchor of the node. Here is an example:

指定节点的锚点。以下是一个示例：

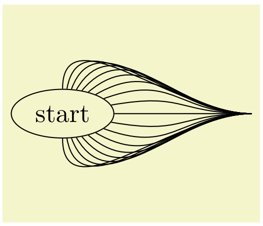


```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}
  \node (shape)   at (0,2)  [draw] {|class Shape|};
  \node (rect)    at (-2,0) [draw] {|class Rectangle|};
  \node (circle)  at (2,0)  [draw] {|class Circle|};
  \node (ellipse) at (6,0)  [draw] {|class Ellipse|};

  \draw (node cs:name=circle,anchor=north) |- (0,1);
  \draw (node cs:name=ellipse,anchor=north) |- (0,1);
  \draw [arrows = -{Triangle[open, angle=60:3mm]}]
          (node cs:name=rect,anchor=north)
        |- (0,1) -| (node cs:name=shape,anchor=south);
\end{tikzpicture}
```

`/tikz/cs/angle=`⟨*degrees*⟩ (no default)

It is also possible to provide an angle *instead* of an anchor. This coordinate refers to a point of the node's border where a ray shot from the center in the given angle hits the border. Here is an example:

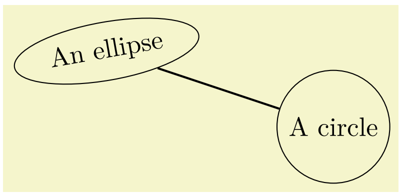
也可以提供一个角度*而不是*一个锚点。这个坐标是指节点边界上的一个点，从中心以给定角度射出的射线与边界相交的位置。这是一个例子：



```
\usetikzlibrary {shapes.geometric}
\begin{tikzpicture}
  \node (start) [draw,shape=ellipse] {start};
  \foreach \angle in {-90, -80, ..., 90}
    \draw (node cs:name=start,angle=\angle)
      .. controls +(\angle:1cm) and +(-1,0) .. (2.5,0);
  \end{tikzpicture}
```

It is possible to provide *neither* the `anchor=` option nor the `angle=` option. In this case, Ti*k*Z will calculate an appropriate border position for you. Here is an example:

可以*既不*使用 `anchor=` 选项，也不使用 `angle=` 选项。在这种情况下，Ti*k*Z 会为您计算一个适当的边界位置。这是一个例子：



```
\usetikzlibrary {shapes.geometric}
\begin{tikzpicture}
  \path (0,0)  node(a) [ellipse,rotate=10,draw] {An ellipse}
        (3,-1) node(b) [circle,draw]            {A circle};
  \draw[thick] (node cs:name=a) -- (node cs:name=b);
\end{tikzpicture}
```

Ti*k*Z will be reasonably clever at determining the border points that you "mean", but, naturally, this may fail in some situations. If Ti*k*Z fails to determine an appropriate border point, the center will be used instead.

Ti*k*Z 在确定您所"指定"的边界点时会相当聪明，但是自然地，在某些情况下会出现失败。如果 Ti*k*Z 无法确定适当的边界点，则使用中心点代替。

Automatic computation of anchors works only with the line-to operations `--`, the vertical/horizontal versions `|-` and `-|`, and with the curve-to operation `..`. For other path commands, such as `parabola` or `plot`, the center will be used. If this is not desired, you should give a named anchor or an angle anchor.

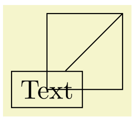
自动计算锚点仅适用于线性操作 `--`，垂直/水平版本 `|-` 和 `-|`，以及曲线操作 `..`。对于其他路径命令（如 `parabola` 或 `plot`），将使用中心点。如果不希望这样，请给出一个命名的锚点或角度锚点。

Note that if you use an automatic coordinate for both the start and the end of a line-to, as in `--(node cs:name=b)--`, then *two* border coordinates are computed with a move-to between them. This is usually exactly what you want.

请注意，如果您对一条线段的起点和终点都使用自动坐标，例如 `--(node cs:name=b)--`，则将计算出*两个*边界坐标，并在它们之间进行移动。这通常是您想要的。

If you use relative coordinates together with automatic anchor coordinates, the relative coordinates are computed relative to the node's center, not relative to the border point. Here is an example:

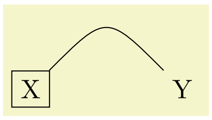
如果您同时使用相对坐标和自动锚点坐标，则相对坐标是相对于节点的中心计算的，而不是相对于边界点计算的。这是一个例子：



```
\tikz \draw (0,0) node(x) [draw] {Text}
            rectangle (1,1)
            (node cs:name=x) -- +(1,1);
```

Similarly, in the following examples both control points are $(1,1)$:

类似地，在以下示例中，两个控制点都是 $(1,1)$：



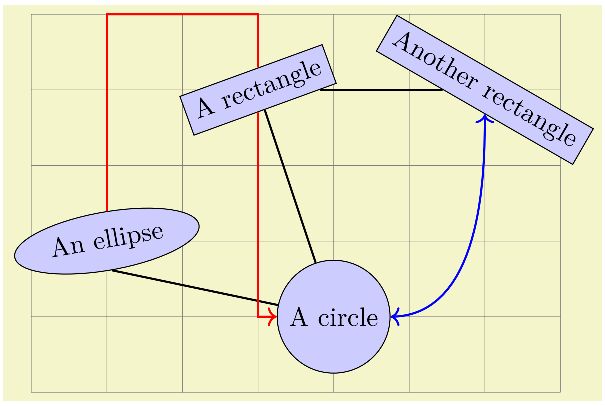
```
\tikz \draw (0,0) node(x) [draw] {X}
            (2,0) node(y) {Y}
            (node cs:name=x) .. controls +(1,1) and +(-1,1) ..
            (node cs:name=y);
```

The implicit way of specifying the node coordinate system is to simply use the name of the node in parentheses as in `(a)` or to specify a name together with an anchor or an angle separated by a dot as in `(a.north)` or `(a.10)`.

隐式指定节点坐标系统的方法是简单地使用节点的名称，如 `(a)`，或者使用由点分隔的名称和锚点或角度，如 `(a.north)` 或 `(a.10)`。

Here is a more complete example:

这是一个更完整的示例：



```
\usetikzlibrary {shapes.geometric}
\begin{tikzpicture}[fill=blue!20]
  \draw[help lines] (-1,-2) grid (6,3);
  \path (0,0)  node(a) [ellipse,rotate=10,draw,fill]    {An ellipse}
        (3,-1) node(b) [circle,draw,fill]               {A circle}
        (2,2)  node(c) [rectangle,rotate=20,draw,fill]  {A rectangle}
        (5,2)  node(d) [rectangle,rotate=-30,draw,fill] {Another rectangle};
  \draw[thick] (a.south) -- (b) -- (c) -- (d);
  \draw[thick,red,->] (a) |- +(1,3) -| (c) |- (b);
  \draw[thick,blue,<->] (b) .. controls +(right:2cm) and +(down:1cm) .. (d);
\end{tikzpicture}
```

13.2.4 Intersection Coordinate Systems Deprecated 交点坐标系统

Often you wish to specify a point that is on the intersection of two lines or shapes. For this, the following coordinate system is useful:
通常情况下，你希望指定一个位于两条线或形状的交点上的点。为此，以下坐标系统很有用：

Coordinate system `intersection`

First, you must specify two objects that should be intersected. These “objects” can either be lines or the shapes of nodes. There are two option to specify the first object:
首先，你需要指定两个想要相交的对象。这些“对象”可以是线或节点的形状。有两种选项可以指定第一个对象：

`/tikz/cs/first line={(`⟨*first coordinate*⟩`)-(`⟨*second coordinate*⟩`)}` (no default)

Specifies that the first object is a line that goes from ⟨*first coordinate*⟩ to metasecond coordinate.
指定第一个对象是一条从 ⟨ 第一个坐标⟩ 到 ⟨ 第二个坐标⟩ 的线。

Note that you have to write `--` between the coordinate, but this does not mean that anything is added to the path. This is simply a special syntax.
注意，在坐标之间你必须使用 `--`，但这并不意味着路径中添加了任何内容。这只是一种特殊的语法。

`/tikz/cs/first node=`⟨*node*⟩ (no default)

Specifies that the first object is a previously defined node named ⟨*node*⟩.
指定第一个对象是一个之前定义的名为 ⟨ 节点⟩ 的节点。

To specify the second object, you use one of the following keys:
要指定第二个对象，你可以使用以下键之一：

`/tikz/cs/second line={(`⟨*first coordinate*⟩`)-(`⟨*second coordinate*⟩`)}` (no default)

As above.

`/tikz/cs/second node=`⟨*node*⟩ (no default)

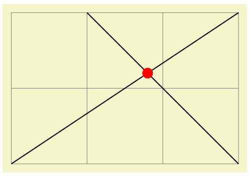
Specifies that the second object is a previously defined node named ⟨*node*⟩.
指定第二个对象是一个之前定义的名为 ⟨ 节点⟩ 的节点。

Since it is possible that two objects have multiple intersections, you may need to specify which solution you want:
由于两个对象可能有多个交点，你可能需要指定要使用的解决方案：

`/tikz/cs/solution=`⟨*number*⟩ (no default, initially `1`)

Specifies which solution should be used. Numbering starts with 1.
指定要使用的解决方案。编号从 1 开始。

The coordinate specified in this way is the ⟨*number*⟩th intersection of the two objects. If the objects do not intersect, an error may occur.
以这种方式指定的坐标是两个对象的第 ⟨ 数字⟩ 个交点。如果对象不相交，可能会出现错误。

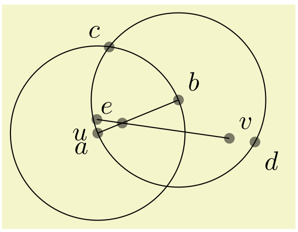


```
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);
  \draw (0,0) coordinate (A) -- (3,2) coordinate (B)
        (1,2)                -- (3,0);

  \fill[red] (intersection cs:
    first line={(A)--(B)},
    second line={(1,2)--(3,0)}) circle (2pt);
\end{tikzpicture}
```

The implicit way of specifying this coordinate system is to write `(intersection` ⟨*number*⟩ `of` ⟨*first object*⟩ `and` ⟨*second object*⟩`)`. Here, ⟨*first object*⟩ either has the form ⟨p_1⟩`--`⟨p_2⟩ or it is just a node name. Likewise for ⟨*second object*⟩. Note that there are *no* parentheses around the p_i. Thus, you would write `(intersection of A--B and 1,2--3,0)` for the intersection of the line through the coordinates `A` and `B` and the line through the points $(1,2)$ and $(3,0)$. You would write `(intersection 2 of c_1 and c_2)` for the second intersection of the node named `c_1` and the node named `c_2`.
隐式指定此坐标系统的方法是写作 `(intersection` ⟨ 数字⟩ `of` ⟨ 第一个对象⟩ `and` ⟨ 第二个对象⟩`)`。这里，⟨ 第一个对象⟩ 的形式可以是 ⟨p_1⟩`--`⟨p_2⟩，或者只是一个节点名称。对于 ⟨ 第二个对象⟩ 也是如此。请注意，p_i 周围没有括号。因此，你可以写 `(intersection of A--B and 1,2--3,0)` 表示通过坐标 `A` 和 `B` 的线与点 $(1,2)$ 和 $(3,0)$ 的线的交点。你可以写 `(intersection 2 of c_1 and c_2)` 表示名为 `c_1` 和 `c_2` 的节点的第二个交点。

Ti*k*Z needs an explicit algorithm for computing the intersection of two shapes and such an algorithm is available only for few shapes. Currently, the following intersection will be computed correctly:
Ti*k*Z 需要明确的算法来计算两个形状的交点，而且目前只有少数形状的交点可以正确计算：

- a line and a line
  一条线和一条线
- a `circle` node and a line (in any order)
  一个 `circle` 节点和一条线（顺序可以任意）
- a `circle` and a `circle`
  一个 `circle` 和一个 `circle`



```
\begin{tikzpicture}[scale=.25]
  \coordinate [label=-135:$a$] (a) at ($ (0,0)   + (rand,rand) $);
  \coordinate [label=45:$b$]   (b) at ($ (3,2) + (rand,rand) $);

  \coordinate [label=-135:$u$] (u) at (-1,1);
  \coordinate [label=45:$v$]   (v) at (6,0);

  \draw (a) -- (b)
        (u) -- (v);

  \node (c1) at (a) [draw,circle through=(b)] {};
  \node (c2) at (b) [draw,circle through=(a)] {};

  \coordinate [label=135:$c$] (c) at (intersection 2 of c1 and c2);
  \coordinate [label=-45:$d$] (d) at (intersection of u--v and c2);
  \coordinate [label=135:$e$] (e) at (intersection of u--v and a--b);

  \foreach \p in {a,b,c,d,e,u,v}
    \fill [opacity=.5] (\p) circle (8pt);
\end{tikzpicture}
```

13.2.5 Tangent Coordinate Systems 切线坐标系统

Coordinate system `tangent`

This coordinate system, which is available only when the Ti*k*Z library `calc` is loaded, allows you to compute the point that lies tangent to a shape. In detail, consider a ⟨*node*⟩ and a ⟨*point*⟩. Now, draw a straight line from the ⟨*point*⟩ so that it "touches" the ⟨*node*⟩ (more formally, so that it is *tangent* to this ⟨*node*⟩). The point where the line touches the shape is the point referred to by the `tangent` coordinate system.

这个坐标系统仅在加载了 Ti*k*Z 库 `calc` 后才可用，它允许你计算与形状相切的点。具体而言，考虑一个 ⟨ 节点⟩ 和一个 ⟨ 点⟩。现在，从 ⟨ 点⟩ 画一条直线，使其 "接触" ⟨ 节点⟩ (更形式化地说，使其与该 ⟨ 节点⟩ *切线*)。线与形状接触的点就是由 `tangent` 坐标系统引用的点。

The following options may be given:

可以给出以下选项：

`/tikz/cs/node=`⟨*node*⟩ (no default)

This key specifies the node on whose border the tangent should lie.

指定切线应位于其边界上的节点。

`/tikz/cs/point=`⟨*point*⟩ (no default)

This key specifies the point through which the tangent should go.

指定切线应通过的点。

`/tikz/cs/solution=`⟨*number*⟩ (no default)

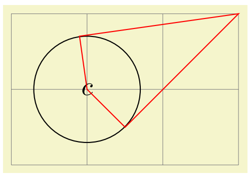
Specifies which solution should be used if there are more than one.

如果有多个解决方案，则指定要使用的解决方案。

A special algorithm is needed in order to compute the tangent for a given shape. Currently, tangents can be computed for nodes whose shape is one of the following:

为了计算给定形状的切线，需要一个特殊的算法。目前，可以计算以下形状的节点的切线：

- `coordinate`
- `circle`



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw[help lines] (0,0) grid (3,2);

  \coordinate (a) at (3,2);

  \node [circle,draw] (c) at (1,1) [minimum size=40pt] {$c$};

  \draw[red] (a)  -- (tangent cs:node=c,point={(a)},solution=1) --
       (c.center) -- (tangent cs:node=c,point={(a)},solution=2) -- cycle;
\end{tikzpicture}
```

There is no implicit syntax for this coordinate system.

这个坐标系统没有隐式语法。

13.2.6 Defining New Coordinate Systems 定义新的坐标系统

While the set of coordinate systems that Ti*k*Z can parse via their special syntax is fixed, it is possible and quite easy to define new explicitly named coordinate systems. For this, the following commands are used:

尽管 Ti*k*Z 可以通过其特殊语法解析一组坐标系统，但是定义新的显式命名的坐标系统是可能且非常容易的。为此，使用以下命令：

`\tikzdeclarecoordinatesystem{`⟨*name*⟩`}{`⟨*code*⟩`}`

This command declares a new coordinate system named ⟨*name*⟩ that can later on be used by writing (⟨*name*⟩ `cs:`⟨*arguments*⟩). When Ti*k*Z encounters a coordinate specified in this way, the ⟨*arguments*⟩ are passed to ⟨*code*⟩ as argument `#1`.

这个命令声明了一个名为 ⟨*name*⟩ 的新坐标系统，可以通过写 (⟨*name*⟩ `cs:`⟨ 参数⟩) 来使用。当 Ti*k*Z 遇到以这种方式指定的坐标时，将把 ⟨ 参数⟩ 作为参数 `#1` 传递给 ⟨*code*⟩。

It is now the job of ⟨*code*⟩ to make sense of the ⟨*arguments*⟩. At the end of ⟨*code*⟩, the two TEX dimensions `\pgf@x` and `\pgf@y` should be have the x- and y-canvas coordinate of the coordinate.

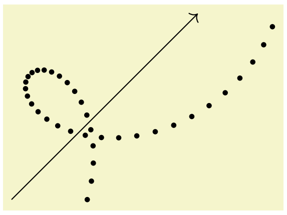
现在，⟨*code*⟩ 的任务是理解 ⟨ 参数⟩。在 ⟨*code*⟩ 的末尾，两个 TEX 尺寸 `\pgf@x` 和 `\pgf@y` 应该具有坐标的 x- 和 y-画布坐标。

It is not necessary, but customary, to parse ⟨*arguments*⟩ using the key–value syntax. However, you can also parse it in any way you like.

不必要，但习惯上，使用键-值语法解析 ⟨ 参数⟩。但你也可以以任何你喜欢的方式解析它。

In the following example, a coordinate system `cylindrical` is defined.

在下面的示例中，定义了一个坐标系统 `cylindrical`。



```
\makeatletter
\define@key{cylindricalkeys}{angle}{\def\myangle{#1}}
\define@key{cylindricalkeys}{radius}{\def\myradius{#1}}
\define@key{cylindricalkeys}{z}{\def\myz{#1}}
\tikzdeclarecoordinatesystem{cylindrical}%
{%
  \setkeys{cylindricalkeys}{#1}%
  \pgfpointadd{\pgfpointxyz{0}{0}{\myz}}{\pgfpointpolarxy{\myangle}{\myradius}}
}
\begin{tikzpicture}[z=0.2pt]
  \draw [->] (0,0,0) -- (0,0,350);
  \foreach \num in {0,10,...,350}
    \fill (cylindrical cs:angle=\num,radius=1,z=\num) circle (1pt);
\end{tikzpicture}
```

`\tikzaliascoordinatesystem{`⟨*new name*⟩`}{`⟨*old name*⟩`}`

Creates an alias of ⟨*old name*⟩.

13.3 Coordinates at Intersections 交点坐标

You will wish to compute the intersection of two paths. For the special and frequent case of two perpendicular lines, a special coordinate system called `perpendicular` is available. For more general cases, the `intersection` library can be used.

您可能希望计算两条路径的交点。对于两条垂直线的特殊且频繁的情况，有一种称为 `perpendicular` 的特殊坐标系可用。对于更一般的情况，可以使用 `intersection` 库。

13.3.1 Intersections of Perpendicular Lines 垂直线的交点

A frequent special case of path intersections is the intersection of a vertical line going through a point p and a horizontal line going through some other point q. For this situation there is a useful coordinate system.

路径交点的频繁特殊情况是通过点 p 经过的垂直线与通过另一点 q 的水平线的交点。对于这种情况，有一个有用的坐标系。

Coordinate system `perpendicular`

You can specify the two lines using the following keys:

您可以使用以下键来指定这两条线：

`/tikz/cs/horizontal line through={(`⟨*coordinate*⟩`)}` (no default)

Specifies that one line is a horizontal line that goes through the given coordinate.

指定一个经过给定坐标的水平线。

`/tikz/cs/vertical line through={(⟨coordinate⟩)}` (no default)

Specifies that the other line is vertical and goes through the given coordinate.

指定另一条是垂直线，经过给定坐标。

However, in almost all cases you should, instead, use the implicit syntax. Here, you write `(⟨p⟩ |- ⟨q⟩)` or `(⟨q⟩ -| ⟨p⟩)`.

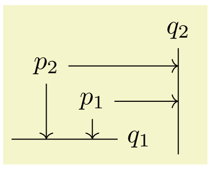
然而，在几乎所有情况下，您应该使用隐式语法。在这里，您可以写成 `(⟨p⟩ |- ⟨q⟩)` 或 `(⟨q⟩ -| ⟨p⟩)`。

For example, `(2,1 |- 3,4)` and `(3,4 -| 2,1)` both yield the same as `(2,4)` (provided the xy-coordinate system has not been modified).

例如，`(2,1 |- 3,4)` 和 `(3,4 -| 2,1)` 都可以得到与 `(2,4)` 相同的结果（前提是 xy 坐标系没有被修改）。

The most useful application of the syntax is to draw a line up to some point on a vertical or horizontal line. Here is an example:

该语法最有用的应用是在垂直或水平线上绘制一条线到达某个点。以下是一个示例：



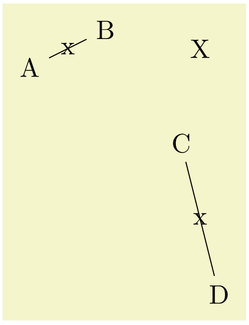
```
\begin{tikzpicture}
  \path (30:1cm) node(p1) {$p_1$}   (75:1cm) node(p2) {$p_2$};

  \draw (-0.2,0) -- (1.2,0) node(xline)[right] {$q_1$};
  \draw (2,-0.2) -- (2,1.2) node(yline)[above] {$q_2$};

  \draw[->] (p1) -- (p1 |- xline);
  \draw[->] (p2) -- (p2 |- xline);
  \draw[->] (p1) -- (p1 -| yline);
  \draw[->] (p2) -- (p2 -| yline);
\end{tikzpicture}
```

Note that in `(⟨c⟩ |- ⟨d⟩)` the coordinates ⟨c⟩ and ⟨d⟩ are *not* surrounded by parentheses. If they need to be complicated expressions (like a computation using the $-syntax), you must surround them with braces; parentheses will then be added around them.

请注意，在 `(⟨c⟩ |- ⟨d⟩)` 中，坐标 ⟨c⟩ 和 ⟨d⟩ 不用括号括起来。如果它们需要复杂的表达式（如使用 $-syntax 进行计算），则必须用花括号括起来；括号将被添加在它们周围。

As an example, let us specify a point that lies horizontally at the middle of the line from A to B and vertically at the middle of the line from C to D:

例如，让我们指定一个点，它水平地位于从 A 到 B 的线的中间，垂直地位于从 C 到 D 的线的中间：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \node (A) at (0,1)    {A};
  \node (B) at (1,1.5)  {B};
  \node (C) at (2,0)    {C};
  \node (D) at (2.5,-2) {D};

  \draw (A) -- (B) node [midway] {x};
  \draw (C) -- (D) node [midway] {x};

  \node at ({$(A)!.5!(B)$} -| {$(C)!.5!(D)$}) {X};
\end{tikzpicture}
```

13.3.2 Intersections of Arbitrary Paths
任意路径的交点

Ti*k*Z Library `intersections`

```
\usetikzlibrary{intersections} % LaTeX and plain TeX
\usetikzlibrary[intersections] % ConTeXt
```

This library enables the calculation of intersections of two arbitrary paths. However, due to the low accuracy of TeX, the paths should not be “too complicated”. In particular, you should not try to intersect paths consisting of lots of very small segments such as plots or decorated paths.

此库使得能够计算两条任意路径的交点。然而，由于 TeX 的低精度，路径不应该太过复杂。特别是，您不应该尝试计算由许多非常小的线段组成的路径的交点，比如曲线或装饰路径。

To find the intersections of two paths in Ti*k*Z, they must be "named". A "named path" is, quite simply, a path that has been named using the following key (note that this is a *different* key from the `name` key, which only attaches a hyperlink target to a path, but does not store the path in a way the is useful for the intersection computation):

要在 Ti*k*Z 中找到两条路径的交点，它们必须被"命名"。"命名路径"就是简单地使用以下键给路径命名（请注意，这是与 `name` 键不同的键，`name` 键仅为路径附加超链接目标，但不以对交点计算有用的方式存储路径）：

`/tikz/name path=`⟨*name*⟩ (no default)
`/tikz/name path global=`⟨*name*⟩ (no default)

The effect of this key is that, after the path has been constructed, just before it is used, it is associated with ⟨*name*⟩. For `name path`, this association survives beyond the final semi-colon of the path but not the end of the surrounding scope. For `name path global`, the association will survive beyond any scope as well. Handle with care.

此键的效果是，在构建路径后，在使用路径之前，将其与 ⟨*name*⟩ 关联。对于 `name path`，此关联将在路径的最后一个分号之后仍然存在，但不会超出周围作用域的末尾。对于 `name path global`，此关联将在任何作用域之外仍然存在。请谨慎使用。

Any paths created by nodes on the (main) path are ignored, unless this key is explicitly used. If the same ⟨*name*⟩ is used for the main path and the node path(s), then the paths will be added together and then associated with ⟨*name*⟩.

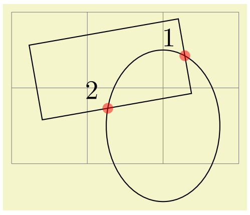
除非显式使用此键，否则将忽略路径上的节点创建的任何路径。如果主路径和节点路径使用相同的 ⟨*name*⟩，则路径将被合并，然后与 ⟨*name*⟩ 关联。

To find the intersection of named paths, the following key is used:
要找到命名路径的交点，使用以下键：

`/tikz/name intersections={`⟨*options*⟩`}` (no default)

This key changes the key path to `/tikz/intersection` and processes ⟨*options*⟩. These options determine, among other things, which paths to use for the intersection. Having processed the options, any intersections are then found. A coordinate is created at each intersection, which by default, will be named `intersection-1`, `intersection-2`, and so on. Optionally, the prefix `intersection` can be changed, and the total number of intersections stored in a TEX-macro.

此键将键路径更改为 `/tikz/intersection` 并处理 ⟨*options*⟩。这些选项确定交点要使用的路径。在处理完选项后，将找到任何交点。在每个交点处创建一个坐标，默认情况下，它们将被命名为 `intersection-1`、`intersection-2` 等。还可以选择更改前缀 `intersection`，以及将交点总数存储在 TEX 宏中。



```
\usetikzlibrary {intersections}
\begin{tikzpicture}[every node/.style={opacity=1, black, above left}]
  \draw [help lines] grid (3,2);
  \draw [name path=ellipse] (2,0.5) ellipse (0.75cm and 1cm);
  \draw [name path=rectangle, rotate=10] (0.5,0.5) rectangle +(2,1);
  \fill [red, opacity=0.5, name intersections={of=ellipse and rectangle}]
    (intersection-1) circle (2pt) node {1}
    (intersection-2) circle (2pt) node {2};
\end{tikzpicture}
```

The following keys can be used in ⟨*options*⟩:
以下键可以在⟨ *选项*⟩ 中使用：

`/tikz/intersection/of=`⟨*name path 1*⟩ `and` ⟨*name path 2*⟩ (no default)

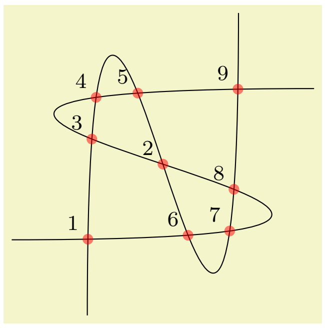
This key is used to specify the names of the paths to use for the intersection.
此键用于指定用于求交的路径的名称。

`/tikz/intersection/name=`⟨*prefix*⟩ (no default, initially `intersection`)

This key specifies the prefix name for the coordinate nodes placed at each intersection.
此键指定放置在每个交点处的坐标节点的前缀名称。

`/tikz/intersection/total=`⟨*macro*⟩ (no default)

This key means that the total number of intersections found will be stored in ⟨*macro*⟩.
此键表示找到的交点总数将存储在⟨*macro*⟩ 中。

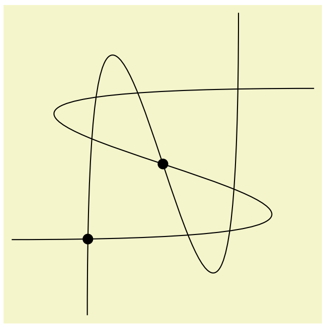


```
\usetikzlibrary {intersections}
\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);

  \fill [name intersections={of=curve 1 and curve 2, name=i, total=\t}]
        [red, opacity=0.5, every node/.style={above left, black, opacity=1}]
        \foreach \s in {1,...,\t}{(i-\s) circle (2pt) node {\footnotesize\s}};
\end{tikzpicture}
```

`/tikz/intersection/by=`⟨*comma-separated list*⟩ (no default)

This key allows you to specify a list of names for the intersection coordinates. The intersection coordinates will still be named ⟨*prefix*⟩`-`⟨*number*⟩, but additionally the first coordinate will also be named by the first element of the ⟨*comma-separated list*⟩. What happens is that the ⟨*comma-separated list*⟩ is passed to the `\foreach` statement and for ⟨*list member*⟩ a coordinate is created at the already-named intersection.

此键允许您指定交点坐标的名称列表。交点坐标仍然被命名为⟨*prefix*⟩`-`⟨*number*⟩，但是还会使用⟨ *逗号分隔的列表*⟩ 的第一个元素为第一个坐标命名。所发生的情况是⟨ *逗号分隔的列表*⟩ 被传递给`\foreach` 语句，并且对于⟨ *列表成员*⟩ 在已命名的交点处创建一个坐标。



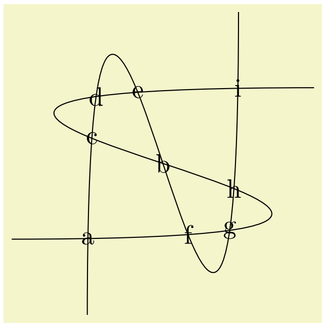
```
\usetikzlibrary {intersections}
\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);

  \fill [name intersections={of=curve 1 and curve 2, by={a,b}}]
        (a) circle (2pt)
        (b) circle (2pt);
\end{tikzpicture}
```

You can also use the `...` notation of the `\foreach` statement inside the ⟨*comma-separated list*⟩.

您还可以在⟨ *逗号分隔的列表*⟩ 中使用`\foreach` 语句的`...` 表示法。

In case an element of the ⟨*comma-separated list*⟩ starts with options in square brackets, these options are used when the coordinate is created. A coordinate name can still, but need not, follow the options. This makes it easy to add labels to intersections:

如果⟨ *逗号分隔的列表*⟩ 的元素以方括号中的选项开头，则在创建坐标时使用这些选项。坐标名称仍然可以，但不必遵循这些选项。这使得向交点添加标签变得很容易：

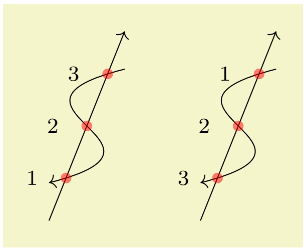


```
\usetikzlibrary {intersections}
\begin{tikzpicture}
  \clip (-2,-2) rectangle (2,2);
  \draw [name path=curve 1] (-2,-1) .. controls (8,-1) and (-8,1) .. (2,1);
  \draw [name path=curve 2] (-1,-2) .. controls (-1,8) and (1,-8) .. (1,2);

  \fill [name intersections={
           of=curve 1 and curve 2,
           by={[label=center:a],[label=center:...],[label=center:i]}}];
\end{tikzpicture}
```

`/tikz/intersection/sort by=`⟨*path name*⟩ (no default)

By default, the intersections are simply returned in the order that the intersection algorithm finds them. Unfortunately, this is not necessarily a "helpful" ordering. This key can be used to sort the intersections along the path specified by ⟨*path name*⟩, which should be one of the paths mentioned in the `/tikz/intersection/of` key.

默认情况下，交点按照交点算法发现它们的顺序简单返回。不幸的是，这不一定是一个"有用"的排序方式。此键可用于沿着由⟨ *路径名称*⟩ 指定的路径对交点进行排序，该路径应为`/tikz/intersection/of` 键中提到的路径之一。



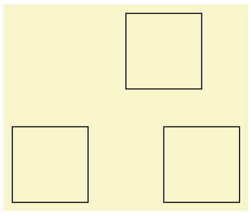
```
\usetikzlibrary {intersections}
\begin{tikzpicture}
\clip (-0.5,-0.75) rectangle (3.25,2.25);
\foreach \pathname/\shift in {line/0cm, curve/2cm}{
  \tikzset{xshift=\shift}
  \draw [->, name path=curve] (1,1.5) .. controls (-1,1) and (2,0.5) .. (0,0);
  \draw [->, name path=line]  (0,-.5) -- (1,2) ;
  \fill [name intersections={of=line and curve,sort by=\pathname, name=i}]
    [red, opacity=0.5, every node/.style={left=.25cm, black, opacity=1}]
    \foreach \s in {1,2,3}{(i-\s) circle (2pt) node {\footnotesize\s}};
}
\end{tikzpicture}
```

13.4 Relative and Incremental Coordinates 相对和增量坐标

13.4.1 Specifying Relative Coordinates 指定相对坐标

You can prefix coordinates by `++` to make them "relative". A coordinate such as `++(1cm,0pt)` means "1cm to the right of the previous position, making this the new current position". Relative coordinates are often useful in "local" contexts:

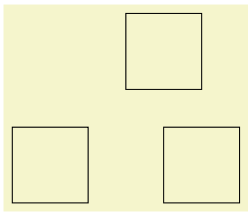
您可以在坐标前面加上`++` 使其成为"相对"坐标。例如，像`++(1cm,0pt)` 这样的坐标意味着"相对于前一个位置向右 1cm，这成为新的当前位置"。在"局部"环境中，相对坐标通常很有用：



```
\begin{tikzpicture}
  \draw (0,0)     -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
  \draw (2,0)     -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
  \draw (1.5,1.5) -- ++(1,0) -- ++(0,1) -- ++(-1,0) -- cycle;
\end{tikzpicture}
```

Instead of `++` you can also use a single `+`. This also specifies a relative coordinate, but it does not "update" the current point for subsequent usages of relative coordinates. Thus, you can use this notation to specify numerous points, all relative to the same "initial" point:

您也可以使用单个`+` 代替`++`。这也指定了一个相对坐标，但它不会"更新"当前点以供后续使用的相对坐标。因此，您可以使用此记法指定多个点，所有这些点都相对于同一个"初始"点：



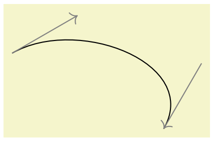
```
\begin{tikzpicture}
  \draw (0,0)     -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
  \draw (2,0)     -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
  \draw (1.5,1.5) -- +(1,0) -- +(1,1) -- +(0,1) -- cycle;
\end{tikzpicture}
```

There is a special situation, where relative coordinates are interpreted differently. If you use a relative coordinate as a control point of a Bézier curve, the following rule applies: First, a relative first control point is taken relative to the beginning of the curve. Second, a relative second control point is taken relative to the end of the curve. Third, a relative end point of a curve is taken relative to the start of the curve.

有一种特殊情况，其中相对坐标的解释方式不同。如果你将相对坐标用作贝塞尔曲线的控制点，则应遵循以下规则：首先，相对于曲线的起点取相对第一个控制点；其次，相对于曲线的终点取相对第二个控制点；第三，相对于曲线的起点取相对终点。

This special behavior makes it easy to specify that a curve should "leave or arrive from a certain direction" at the start or end. In the following example, the curve "leaves" at 30° and "arrives" at 60°:

这种特殊行为使得在起点或终点处指定曲线应该“从某个方向离开或到达”变得容易。在下面的示例中，曲线“离开”时角度为 30°，而“到达”时角度为 60°：



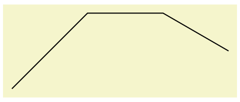
```
\begin{tikzpicture}
  \draw (1,0) .. controls +(30:1cm) and +(60:1cm) .. (3,-1);
  \draw[gray,->] (1,0) -- +(30:1cm);
  \draw[gray,<-] (3,-1) -- +(60:1cm);
\end{tikzpicture}
```

13.4.2 Rotational Relative Coordinates 旋转相对坐标

You may sometimes wish to specify points relative not only to the previous point, but additionally relative to the tangent entering the previous point. For this, the following key is useful:
有时，你可能希望指定相对于前一个点以及相对于进入前一个点的切线的点。为此，以下关键字很有用：

`/tikz/turn` (no value)

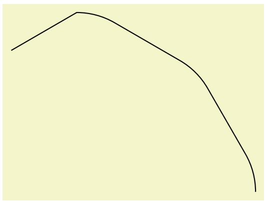
This key can be given as an option to a ⟨*coordinate*⟩ as in the following example:
你可以将该关键字作为选项给出，如下例所示：



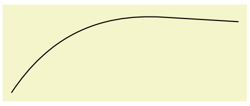
```
\tikz \draw (0,0) -- (1,1) -- ([turn]-45:1cm) -- ([turn]-30:1cm);
```

The effect of this key is to locally shift the coordinate system so that the last point reached is at the origin and the coordinate system is “turned” so that the x-axis points in the direction of a tangent entering the last point. This means, in effect, that when you use polar coordinates of the form ⟨*relative angle*⟩:⟨*distance*⟩ together with the `turn` option, you specify a point that lies at ⟨*distance*⟩ from the last point in the direction of the last tangent entering the last point, but with a rotation of ⟨*relative angle*⟩.
此关键字的效果是局部移动坐标系，使得到达的最后一个点位于原点，并且坐标系“转动”，使得 x 轴指向进入最后一个点的切线方向。这实际上意味着，当你使用形式为⟨*relative angle*⟩:⟨*distance*⟩ 的极坐标与`turn` 选项一起使用时，你指定了一个位于距离最后一个点⟨*distance*⟩、方向为进入最后一个点的切线方向、旋转角为⟨*relative angle*⟩ 的点。

This key also works with curves . . .
该关键字还适用于曲线……

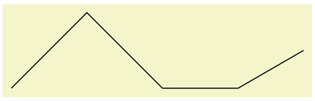


```
\tikz [delta angle=30, radius=1cm]
  \draw (0,0) arc [start angle=0]  -- ([turn]0:1cm)
              arc [start angle=30] -- ([turn]0:1cm)
              arc [start angle=60] -- ([turn]30:1cm);
```



```
\tikz \draw (0,0) to [bend left] (2,1) -- ([turn]0:1cm);
```

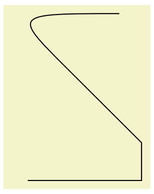
. . . and with plots . . .
……以及绘图……



```
\tikz \draw plot coordinates {(0,0) (1,1) (2,0) (3,0) } -- ([turn]30:1cm);
```

Although the above examples use polar coordinates with `turn`, you can also use any normal coordinate. For instance, `([turn]1,1)` will append a line of length $\sqrt{2}$ that is turns by 45° relative to the tangent to the last point.

虽然上面的示例使用了带有`turn` 的极坐标，但你也可以使用任何普通坐标。例如，`([turn]1,1)` 将附加一条长度为 $\sqrt{2}$ 的线，相对于最后一个点的切线旋转 45°。



```
\tikz \draw (0.5,0.5) -| (2,1) -- ([turn]1,1)
             .. controls ([turn]0:1cm) .. ([turn]-90:1cm);
```

13.4.3 Relative Coordinates and Scopes 相对坐标和作用域

An interesting question is, how do relative coordinates behave in the presence of scopes? That is, suppose we use curly braces in a path to make part of it "local", how does that affect the current position? On the one hand, the current position certainly changes since the scope only affects options, not the path itself. On the other hand, it may be useful to "temporarily escape" from the updating of the current point.

一个有趣的问题是，在作用域存在的情况下，相对坐标的行为如何？也就是说，假设我们在路径中使用花括号使其的一部分"局部化"，那么这将如何影响当前位置呢？一方面，当前位置肯定会发生变化，因为作用域只影响选项，而不影响路径本身。另一方面，暂时"逃离"当前点的更新可能是有用的。

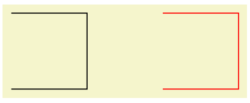
Since both interpretations of how the current point and scopes should "interact" are useful, there is a (local!) option that allows you to decide which you need.

由于当前点和作用域应如何"交互"的这两种解释都是有用的，因此有一个（局部的！）选项可以让你决定你需要哪种方式。

`/tikz/current point is local=`⟨*boolean*⟩ (no default, initially `false`)

Normally, the scope path operation has no effect on the current point. That is, curly braces on a path have no effect on the current position:

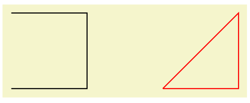
通常来说，作用域路径操作对当前点没有影响。也就是说，路径中的花括号对当前位置没有影响：



```
\begin{tikzpicture}
  \draw      (0,0) -- ++(1,0)   -- ++(0,1)   -- ++(-1,0);
  \draw[red] (2,0) -- ++(1,0) { -- ++(0,1) } -- ++(-1,0);
\end{tikzpicture}
```

If you set this key to `true`, this behavior changes. In this case, at the end of a group created on a path, the last current position reverts to whatever value it had at the beginning of the scope. More precisely, when Ti*k*Z encounters `}` on a path, it checks whether at this particular moment the key is set to `true`. If so, the current position reverts to the value it had when the matching `{` was read.

如果将此关键字设置为 `true`，则此行为会发生变化。在这种情况下，在路径上的一个组的末尾，当前位置将恢复为与读取相匹配的 `{` 时的值。更准确地说，当 Ti*k*Z 在路径上遇到 `}` 时，它会检查此时该关键字是否设置为 `true`。如果是，则当前位置将恢复为读取匹配的 `{` 时的值。



```
\begin{tikzpicture}
  \draw      (0,0) -- ++(1,0)   -- ++(0,1)   -- ++(-1,0);
  \draw[red] (2,0) -- ++(1,0)
     { [current point is local] -- ++(0,1) } -- ++(-1,0);
\end{tikzpicture}
```

In the above example, we could also have given the option outside the scope, for instance as a parameter to the whole scope.

在上面的示例中，我们也可以在作用域之外给出选项，例如作为整个作用域的参数。

13.5 Coordinate Calculations 坐标计算

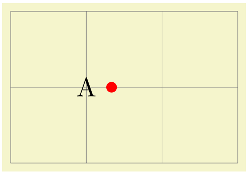
Ti*k*Z Library `calc`

```
\usetikzlibrary{calc} % LaTeX and plain TeX
\usetikzlibrary[calc] % ConTeXt
```

You need to load this library in order to use the coordinate calculation functions described in the present section.

在使用本节中描述的坐标计算函数之前，您需要加载此库。

It is possible to do some basic calculations that involve coordinates. In essence, you can add and subtract coordinates, scale them, compute midpoints, and do projections. For instance, `($(a) + 1/3*(1cm,0)$)` is the coordinate that is 1/3cm to the right of the point `a`:
可以进行涉及坐标的基本计算。基本上，可以添加和减去坐标，对其进行缩放，计算中点和投影。例如，`($(a) + 1/3*(1cm,0)$)` 是点 `a` 右侧 1/3cm 处的坐标：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \node (a) at (1,1) {A};
  \fill [red] ($(a) + 1/3*(1cm,0)$) circle (2pt);
\end{tikzpicture}
```

13.5.1 The General Syntax
通用语法

The general syntax is the following:
通用语法如下：

`([`⟨*options*⟩`]$`⟨*coordinate computation*⟩`$)`.

As you can see, the syntax uses the TeX math symbol `$` to indicate that a "mathematical computation" is involved. However, the `$` has no other effect, in particular, no mathematical text is typeset.
如您所见，语法使用 TeX 数学符号 `$` 表示涉及"数学计算"。但是，`$` 没有其他作用，特别是没有进行数学文本排版。
The ⟨*coordinate computation*⟩ has the following structure:
⟨*坐标计算*⟩ 具有以下结构：

1. It starts with

   以如下形式开始：

   ⟨*factor*⟩`*`⟨*coordinate*⟩⟨*modifiers*⟩

2. This is optionally followed by `+` or `-` and then another

   可选地后跟 `+` 或 `-`，然后是另一个

   ⟨*factor*⟩`*`⟨*coordinate*⟩⟨*modifiers*⟩

3. This is once more followed by `+` or `-` and another of the above modified coordinate; and so on.

   再次后跟 `+` 或 `-` 和上述修饰过的坐标；以此类推。

In the following, the syntax of factors and of the different modifiers is explained in detail.
接下来，详细解释因子和不同修饰符的语法。

13.5.2 The Syntax of Factors
因子的语法

The ⟨*factor*⟩s are optional and detected by checking whether the ⟨*coordinate computation*⟩ starts with a `(`. Also, after each ± a ⟨*factor*⟩ is present if, and only if, the `+` or `-` sign is not directly followed by `(`.
⟨*因子*⟩ 是可选的，通过检查 ⟨*坐标计算*⟩ 是否以 `(` 开头来确定。此外，在每个 ± 后，只有在 `+` 或 `-` 符号直接后面没有跟随 `(` 时才出现 ⟨*因子*⟩。
If a ⟨*factor*⟩ is present, it is evaluated using the `\pgfmathparse` macro. This means that you can use pretty complicated computations inside a factor. A ⟨*factor*⟩ may even contain opening parentheses, which creates a complication: How does Ti*k*Z know where a ⟨*factor*⟩ ends and where a coordinate starts? For

instance, if the beginning of a ⟨*coordinate computation*⟩ is `2*(3+4...`, it is not clear whether `3+4` is part of a ⟨*coordinate*⟩ or part of a ⟨*factor*⟩. Because of this, the following rule is used: Once it has been determined, that a ⟨*factor*⟩ is present, in principle, the ⟨*factor*⟩ contains everything up to the next occurrence of `*(`. Note that there is no space between the asterisk and the parenthesis.

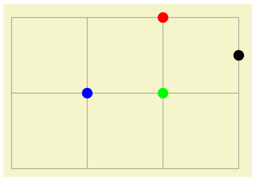
如果存在 ⟨ *因子*⟩，则使用 `\pgfmathparse` 宏对其进行求值。这意味着您可以在因子内使用相当复杂的计算。一个 ⟨ *因子*⟩ 甚至可以包含括号，这会导致一个问题：Ti*k*Z 如何知道 ⟨ *因子*⟩ 在哪里结束，坐标从哪里开始？例如，如果 ⟨ *坐标计算*⟩ 的开头是 `2*(3+4...`，不清楚 `3+4` 是 ⟨ *坐标*⟩ 的一部分还是 ⟨ *因子*⟩ 的一部分。因此，采用以下规则：一旦确定存在 ⟨ *因子*⟩，原则上，⟨ *因子*⟩ 包含直到下一个出现的 `*(` 为止。请注意，星号和括号之间没有空格。

It is permissible to put the ⟨*factor*⟩ in curly braces. This can be used whenever it is unclear where the ⟨*factor*⟩ would end.

可以将 ⟨ *因子*⟩ 放在花括号中。这可以在不清楚 ⟨ *因子*⟩ 结束的地方使用。

Here are some examples of coordinate specifications that consist of exactly one ⟨*factor*⟩ and one ⟨*coordinate*⟩:

以下是由一个 ⟨ *因子*⟩ 和一个 ⟨ *坐标*⟩ 组成的坐标规范的示例：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \fill [red] ($2*(1,1)$) circle (2pt);
  \fill [green] (${1+1}*(1,.5)$) circle (2pt);
  \fill [blue] ($cos(0)*sin(90)*(1,1)$) circle (2pt);
  \fill [black] (${3*(4-3)}*(1,0.5)$) circle (2pt);
\end{tikzpicture}
```

13.5.3 The Syntax of Partway Modifiers 部分修饰符的语法

A ⟨*coordinate*⟩ can be followed by different ⟨*modifiers*⟩. The first kind of modifier is the *partway modifier*. The syntax (which is loosely inspired by Uwe Kern's `xcolor` package) is the following:

⟨ 坐标⟩ 可以后跟不同的⟨ *修饰符*⟩。第一种修饰符是*部分修饰符*。其语法（受 Uwe Kern 的`xcolor` 宏包的启发，但略有不同）如下：

⟨*coordinate*⟩`!`⟨*number*⟩`!`⟨*angle*⟩`:`⟨*second coordinate*⟩

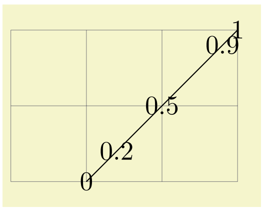
One could write for instance

例如，可以写作：

```
(1,2)!.75!(3,4)
```

The meaning of this is: "Use the coordinate that is three quarters on the way from `(1,2)` to `(3,4)`." In general, ⟨*coordinate x*⟩`!`⟨*number*⟩`!`⟨*coordinate y*⟩ yields the coordinate $(1 - \langle number\rangle)\langle coordinate\ x\rangle + \langle number\rangle\langle coordinate\ y\rangle$. Note that this is a bit different from the way the ⟨*number*⟩ is interpreted in the `xcolor` package: First, you use a factor between 0 and 1, not a percentage, and, second, as the ⟨*number*⟩ approaches 1, we approach the second coordinate, not the first. It is permissible to use a ⟨*number*⟩ that is smaller than 0 or larger than 1. The ⟨*number*⟩ is evaluated using the `\pgfmathparse` command and, thus, it can involve complicated computations.

它的含义是："使用从`(1,2)` 到`(3,4)` 的路径的四分之三的位置上的坐标。" 一般而言，⟨ *坐标 x*⟩`!`⟨ *数值*⟩`!`⟨ *坐标 y*⟩ 将生成坐标 $(1 - \langle 数值\rangle)\langle 坐标\ x\rangle + \langle 数值\rangle\langle 坐标\ y\rangle$。需要注意的是，这与`xcolor` 宏包中对⟨ *数值*⟩ 的解释有所不同：首先，使用的是 0 到 1 之间的因子，而不是百分比；其次，当⟨ *数值*⟩ 趋近于 1 时，接近第二个坐标，而不是第一个坐标。可以使用小于 0 或大于 1 的⟨ *数值*⟩。⟨ *数值*⟩ 通过`\pgfmathparse` 命令计算，因此可以涉及复杂的计算。



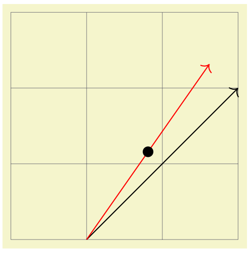
```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \draw (1,0) -- (3,2);

  \foreach \i in {0,0.2,0.5,0.9,1}
    \node at ($(1,0)!\i!(3,2)$) {\i};
\end{tikzpicture}
```

The ⟨*second coordinate*⟩ may be prefixed by an ⟨*angle*⟩, separated with a colon, as in `(1,1)!.5!60:(2,2)`. The general meaning of ⟨*a*⟩`!`⟨*factor*⟩`!`⟨*angle*⟩`:`⟨*b*⟩ is: "First, consider the line from ⟨*a*⟩ to ⟨*b*⟩. Then rotate this line by ⟨*angle*⟩ *around the point* ⟨*a*⟩. Then the two endpoints of this line will be ⟨*a*⟩ and some point ⟨*c*⟩. Use this point ⟨*c*⟩ for the subsequent computation, namely the partway computation."

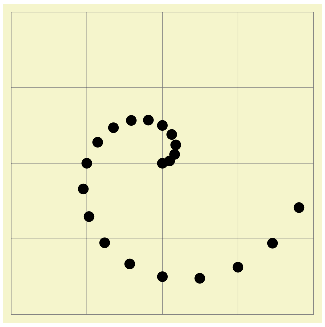
⟨ *第二个坐标*⟩ 可以带有前缀⟨ *角度*⟩，用冒号分隔，例如 `(1,1)!.5!60:(2,2)`。⟨*a*⟩`!`⟨ *因子*⟩`!`⟨ *角度*⟩`:`⟨*b*⟩ 的一般含义是："首先，考虑从⟨*a*⟩ 到⟨*b*⟩ 的直线。然后以⟨*a*⟩ 为中心，将这条直线旋转⟨ *角度*⟩。这条直线的两个端点将是⟨*a*⟩ 和某个点⟨*c*⟩。使用该点⟨*c*⟩ 进行后续计算，即部分计算。"

Here are two examples:

以下是两个示例：



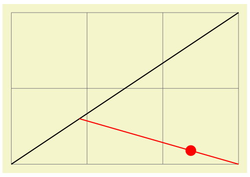
```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,3);

  \coordinate (a) at (1,0);
  \coordinate (b) at (3,2);

  \draw[->] (a) -- (b);

  \coordinate (c) at ($ (a)!1! 10:(b) $);

  \draw[->,red] (a) -- (c);

  \fill ($ (a)!.5! 10:(b) $) circle (2pt);
\end{tikzpicture}
```



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (4,4);

  \foreach \i in {0,0.1,...,2}
    \fill ($(2,2) !\i! \i*180:(3,2)$) circle (2pt);
\end{tikzpicture}
```

You can repeatedly apply modifiers. That is, after any modifier you can add another (possibly different) modifier.

可以重复应用修饰符。也就是说，在任何修饰符之后，可以添加另一个（可能不同的）修饰符。



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \draw (0,0) -- (3,2);
  \draw[red] ($(0,0)!.3!(3,2)$) -- (3,0);
  \fill[red] ($(0,0)!.3!(3,2)!.7!(3,0)$) circle (2pt);
\end{tikzpicture}
```

13.5.4 The Syntax of Distance Modifiers
部分修饰符的语法

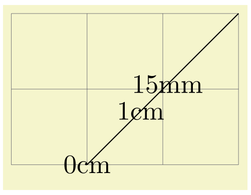
A *distance modifier* has nearly the same syntax as a partway modifier, only you use a ⟨*dimension*⟩ (something like `1cm`) instead of a ⟨*factor*⟩ (something like `0.5`):

一个*距离修饰符*的语法几乎与*部分修饰符*相同，只是你使用一个⟨*dimension*⟩（类似于`1cm` 的东西）而不是⟨*factor*⟩（类似于`0.5`）：

⟨*coordinate*⟩`!`⟨*dimension*⟩`!`⟨*angle*⟩`:`⟨*second coordinate*⟩

When you write ⟨*a*⟩`!`⟨*dimension*⟩`!`⟨*b*⟩, this means the following: Use the point that is distanced ⟨*dimension*⟩ from ⟨*a*⟩ on the straight line from ⟨*a*⟩ to ⟨*b*⟩. Here is an example:

当你写下⟨*a*⟩`!`⟨*dimension*⟩`!`⟨*b*⟩ 时，意味着以下内容：在从⟨*a*⟩ 到⟨*b*⟩ 的直线上，距离⟨*a*⟩ ⟨*dimension*⟩ 的点。以下是一个示例：



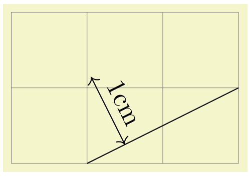
```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \draw (1,0) -- (3,2);

  \foreach \i in {0cm,1cm,15mm}
    \node at ($(1,0)!\i!(3,2)$) {\i};
\end{tikzpicture}
```

As before, if you use a ⟨*angle*⟩, the ⟨*second coordinate*⟩ is rotated by this much around the ⟨*coordinate*⟩ before it is used.

与之前一样，如果使用⟨*angle*⟩，⟨*second coordinate*⟩ 在使用之前会旋转这么多。

The combination of an ⟨*angle*⟩ of `90` degrees with a distance can be used to "offset" a point relative to a line. Suppose, for instance, that you have computed a point `(c)` that lies somewhere on a line from `(a)` to `(b)` and you now wish to offset this point by `1cm` so that the distance from this offset point to the line is `1cm`. This can be achieved as follows:

将⟨*angle*⟩ 为`90` 度与一个距离结合起来，可以用于相对于一条直线进行"偏移"一个点。例如，假设你计算出一个点`(c)` 位于从`(a)` 到`(b)` 的某条线上，现在你希望将此点偏移`1cm`，使得该偏移点到该线的距离为`1cm`。可以按如下方式实现：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \coordinate (a) at (1,0);
  \coordinate (b) at (3,1);

  \draw (a) -- (b);

  \coordinate (c) at ($ (a)!.25!(b) $);
  \coordinate (d) at ($ (c)!1cm!90:(b) $);

  \draw [<->] (c) -- (d) node [sloped,midway,above] {1cm};
\end{tikzpicture}
```

13.5.5 The Syntax of Projection Modifiers
投影修饰符的语法

The projection modifier is also similar to the above modifiers: It also gives a point on a line from the ⟨*coordinate*⟩ to the ⟨*second coordinate*⟩. However, the ⟨*number*⟩ or ⟨*dimension*⟩ is replaced by a ⟨*projection coordinate*⟩:

投影修饰符与上述修饰符类似：它也会给出从⟨ 坐标⟩ 到⟨ 第二个坐标⟩ 的线上的一个点。然而，⟨ 数字⟩ 或⟨ 尺寸⟩ 被替换为⟨ 投影坐标⟩：

⟨*coordinate*⟩`!`⟨*projection coordinate*⟩`!`⟨*angle*⟩`:`⟨*second coordinate*⟩

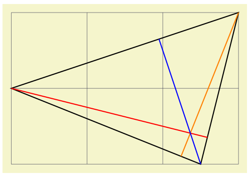
Here is an example:

下面是一个示例：

```
(1,2)!(0,5)!(3,4)
```

The effect is the following: We project the ⟨*projection coordinate*⟩ orthogonally onto the line from ⟨*coordinate*⟩ to ⟨*second coordinate*⟩. This makes it easy to compute projected points:

效果如下：我们将⟨ 投影坐标⟩ 在垂直于从⟨ 坐标⟩ 到⟨ 第二个坐标⟩ 的直线上进行投影。这使得计算投影点变得容易：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (3,2);

  \coordinate (a) at (0,1);
  \coordinate (b) at (3,2);
  \coordinate (c) at (2.5,0);

  \draw (a) -- (b) -- (c) -- cycle;

  \draw[red]    (a) -- ($(b)!(a)!(c)$);
  \draw[orange] (b) -- ($(a)!(b)!(c)$);
  \draw[blue]   (c) -- ($(a)!(c)!(b)$);
\end{tikzpicture}
```

```
\setcounter{section}{13}
\setcounter{subsection}{5}
\setcounter{subsubsection}{5}
```