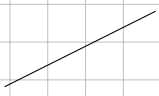
1 Basic Drawing Operations

1.1 Cartesian Coordinates

Drawing with the TikZ package must happen inside the TikZ environment. The simplest possible drawing is a straight line. Below is the code to draw a single line from (0,0) to (2,1). This line is drawn with cartesian coordinates. Note, the semicolon is necessary after each TikZ command or your document will not compile.



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

1.2 Polar Coordinates

Here's how to draw with polar coordinates. The first number is the angle above the horizontal going counterclockwise, and the second number specifies the radius. The colon tells TikZ that the ordered pair is in polar coordinates and not cartesian.

```
\draw (0:2cm) -- (90:2cm);
```

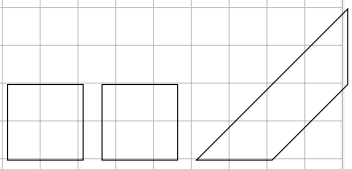
1.3 Arbitrary Polygons

You can connect multiple points in the same way but simply separating the points by double dashes. Notice the use of the keyword *cycle*. This cycles the drawing back to the original point. This is the preferred way to enclose a polygon, rather than hardcoding in the original point again.

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

1.4 Relative Coordinates

Sometimes you want to specify your next coordinate with the current coordinate as the origin of the next point.

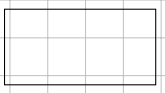


Using a plus sign before the next specified point will place that point relative to the previous specified point. If you have multiple points in a row specified with a plus, the first point will be the origin relative to the following points.

Using two plus signs will *always* use the previous point as the relative origin. The image at right shows the difference this makes.

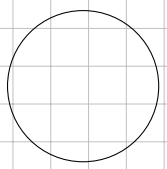
```
\draw (0,0)--(1,0)--(1,1)--(0,1)--cycle;
\draw (1.25,0)--+(1,0)--+(1,1)--+(0,1)--cycle;
\draw (3,0)--++(1,0)--++(1,1)--++(0,1)--cycle;
```

1.5 Shapes



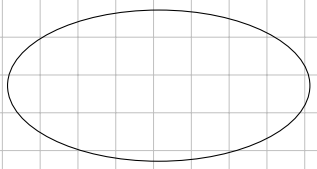
With very similar syntax, we can draw a simple rectangle by using the *rectangle* keyword instead of two dashes. The first point indicates the bottom left corner of the rectangle, and the second point indicates the top right corner of the rectangle.

```
\draw (0,0) rectangle (2,1);
```



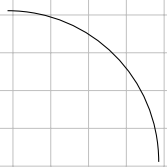
We can draw a circle using the keyword *circle*. The first point in parentheses is the center, the second number in parentheses is the radius.

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



Drawing an ellipse is draw with the ellipse keyword. Just like circle the first coordinate is the center, the second argument is the x-radius and the y-radius separated from the keyword *and*.

```
\draw (0,0) ellipse (2cm and 1cm);
```



Drawing an arc requires using the *arc* keyword. The first number is the center of the arc (like the center of a circle or ellipse). The second set of information is the starting angle, ending angle, radius, separated by colons.

```
\draw (3,0) arc (0:90:2cm);
```

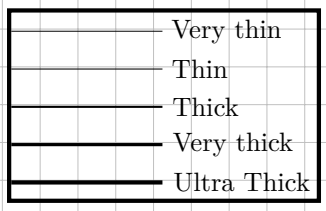
2 Styling Drawing Commands

All of the drawing commands thus far have been basic thin lines, which are pretty boring. You can easily style drawings by giving the draw command optional arguments. There are many, many styles you can apply, so we're gonna try and focus on some basic ones and show you how to learn more.

Color, stroke width, line style, decoration

2.1 Line Widths

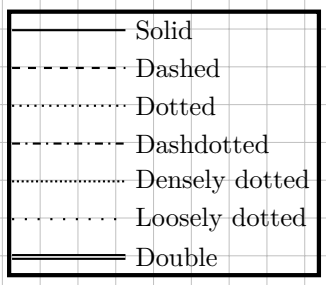
The following shows the different line widths you can use when drawing in TikZ. This should be an optional argument to the draw command. For instance, the first light, shown at right, is drawn with the following:




```
\draw[very thin] (0,0) -- (2,0);
```

2.2 Line Types

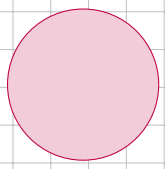
The example on the right shows different line styles you can use. Just like line widths you can use these as arguments to the draw command with the name in lowercase. The dashed line is drawn with:




```
\draw[thick,dashed] (0,-1) -- (3,-1);
```

2.3 Filled Shapes

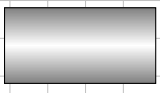
Drawing a filled object is a matter of specifying a keyword argument, which is fill, with a value of the color you want. Using filldraw also allows setting the line color using the draw parameter.



```
\filldraw[fill=purple!20,
        draw=purple]
        (0,0) circle (1cm);
```

2.4 Vertical Gradient

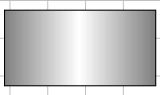
Drawing an object with a gradient is done with the *shadedraw* command instead of the draw command. Important note: the middle color should be specified last or it will not be recognized.



```
\shadedraw[top color=gray,
           bottom color=gray,
           middle color=white]
          (0,0) rectangle (2,1);
```

2.5 Horizontal Gradient

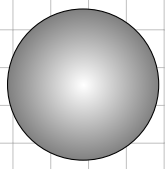
Drawing an object with a gradient is done with the *shadedraw* command instead of the draw command. Important note: the middle color should be specified last or it will not be recognized.



```
\shadedraw[left color=gray,
           right color=gray,
           middle color=white]
          (0,0) rectangle (2,1);
```

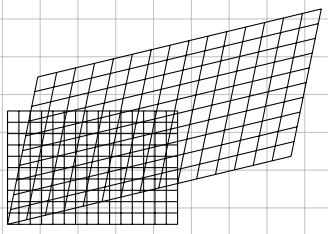
2.6 Radial Gradient

```
\shadedraw[inner color=white,
           outer color=gray,
           draw=black]
           (0,0) circle (1cm);
```



3 Transformations

Transformations can be applied to an entire tikzpicture, scope (discussed later), or a single command. There are many transformation options that can linearly transform the location of defined points.

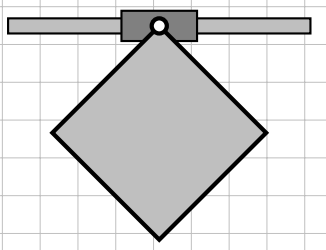


```
\draw[step=0.1]
     0,0) grid (1.5,1);
\draw[cm={1.667,0.4,0.267,1.3,(0,0)},
     step=0.1]
     (0,0) grid (1.5,1);
```

The parameter *cm* is the deformation map, while the second arguement coordinate pari of the x and y shift.

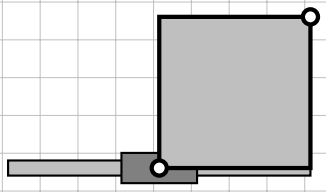
4 Drawing with Scopes

A scope is like a layer in Photoshop or AutoCAD. Scopes can have arbitrary options, the same options available when using the draw command, or optional inputs to a TikZ environment.



The figure shown at right shows how the idea of a *scope* can be used to rotate objects in that scope about a point. The plate connected to the pin is drawn inside a scope environment.

The code to draw the rotated plate is shown below. The fist picture shows the final product. The second picture shows the plate at the original position before the *rotate around* option is applied.



```
\begin{scope}[rotate around={-135:(0,0)}]
\draw[ultra thick,fill=lightgray]
     (0,0) rectangle (2,2);
\end{scope}
```

Figure 1 is a multi-spring system drawn with TikZ. The springs are rendered using the *stanli* package. The package offers a spring lifestyle.

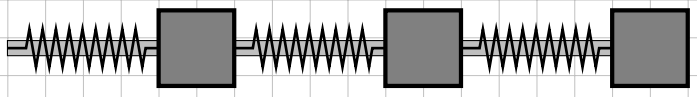


Figure 1: Drawing springs