# Reprint from
# The *Mathematica* Conference

*June, 1992   Boston, MA*

880.21.10.1992

# Chapter 5 Coordinate Systems



This chapter discusses the different coordinate systems which are used by *Mathematica* graphics. We see how *Mathematica* specifies coordinates and how the viewing region is chosen. Then we look at the various transformations which are carried out to map primitives onto screen or paper. Finally considerations specific to three-dimensional rendering are examined.

# 5.1 Coordinate Specification

| | |
|---|---|
| `{`$x$`, `$y$`}` | two-dimensional coordinates |
| `Scaled[{`$x$`, `$y$`}]` | |
| `Scaled[{`$dx$`, `$dy$`}, {`$x$`, `$y$`}]` | |
| `{`$x$`, `$y$`, `$z$`}` | three-dimensional coordinates |
| `Scaled[{`$x$`, `$y$`, `$z$`}]` | |
| `Scaled[{`$dx$`, `$dy$`, `$dz$`}, {`$x$`, `$y$`, `$z$`}]` | |

Types of coordinates for two and three dimensions.

The coordinates for a primitive such as `Line` or `Polygon` can be given in a number of ways. The simplest and most obvious is to give pairs of numbers for two-dimensional graphics and triples for three-dimensional graphics. These are called user coordinates; they are the initial points which we wish to represent. An alternative way to represent points, in a particular part of the picture, is to use `Scaled` coordinates. These refer to display coordinates. For example, they allow us to draw something in the middle of the image without having to know anything about the user coordinates being displayed.

## 5.1.1 PlotRange

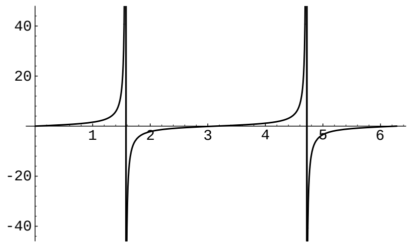
As an object is rendered the list of primitives and directives is scanned to determine the extent of the coordinates which are shown. This is controlled by the option `PlotRange` which provides considerable control of the range which will be seen.

| | |
|---|---|
| `PlotRange -> Automatic` | only the most interesting part |
| `PlotRange -> All` | the full range of coordinates |
| `PlotRange -> {`*min*`, `*max*`}` | the range `{`*min* `->` *max*`}` |

Settings for the `PlotRange` option.

This shows a plot of `Tan[x]` with the default value `PlotRange -> Automatic`.

```
In[1]:= plot = Plot[ Tan[x], {x,0,2Pi}]
```



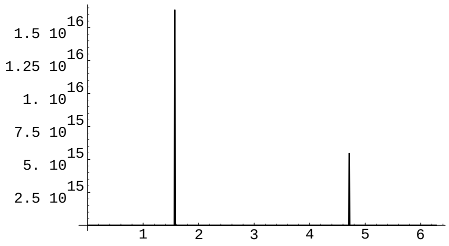

This shows the actual numbers which were used.

```
In[2]:= FullOptions[ %, PlotRange]

Out[2]= {{-0.15708, 6.44026}, {-45.9115, 47.9542}}
```

This plots all the primitives. Notice that this shows less about the `Tan` function than the previous plot.

```
In[3]:= Show[ plot, PlotRange -> All]
```



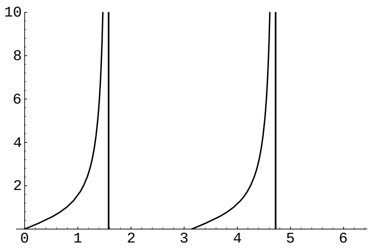

`FullOptions` is a very useful way to find out what `PlotRange` was actually used.

```
In[4]:= FullOptions[ %, PlotRange]
```

Out[4]= {{-0.15708, 6.44026}, {$-4.08294\ 10^{14}$, $1.67401\ 10^{16}$}}

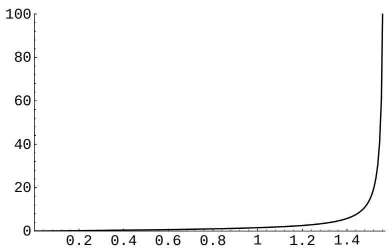
This applies to the $y$ direction only. The $x$ `PlotRange` is determined by the default.

```
In[5]:= Show[ plot, PlotRange -> {0,10}]
```




This works on both dimensions.

```
In[6]:= Show[ plot, PlotRange -> {{0, Pi/2}, {0,100}}]
```




In general, a wide range of settings and combinations for the `PlotRange` option can be used.

| | |
|---|---|
| `PlotRange -> {Automatic, All}` | two dimensions |
| `PlotRange -> {All, {5., 10.}}` | two dimensions |
| `PlotRange -> {Automatic, {3.,5.}, All}` | three dimensions |

Example settings for `PlotRange`.

## ■ 5.1.2 Scaled

The examples of primitives which we have seen all showed the coordinates being given in one system, the original or user coordinate system. Sometimes you wish to choose points described in the final or display coordinate system. This is done with the `Scaled` command. Here I describe the two-dimensional form. There is an equivalent three-dimensional version.

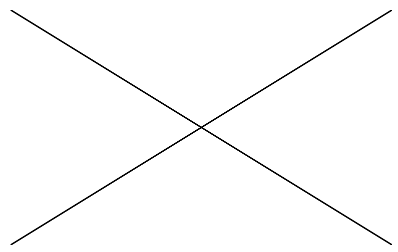
| | |
|---|---|
| `{`*x*`,` *y*`}` | `{`*x*`,` *y*`}` in user coordinates |
| `Scaled[{`*sx*`,` *sy*`}]` | `{`*sx*`,` *sy*`}` in display coordinates |
| `Scaled[{`*dx*`,` *dy*`}, {`*x*`,`*y*`}]` | `{`*dx*`,` *dy*`}` in display coordinates relative to `{`*x*`,` *y*`}` in user coordinates |

Coordinates which can be used to build two-dimensional primitives.

`Scaled` coordinates are chosen to run from 0 to 1 in both dimensions and to have their origin at the lower-left corner of the picture. Thus `Scaled[{0,0}]` refers to the minimum of the `PlotRange` and `Scaled[{1,1}]` to the maximum.

This always draws lines from corner to corner.

```
In[7]:= Show[ Graphics[{
          Line[ {Scaled[ {0,0}], Scaled[{1,1}]}],
          Line[ {Scaled[ {0,1}], Scaled[{1,0}]}]}]]
```



This is a *Mathematica* function which uses the offset version of `Scaled`.

```
In[8]:= errorbar[ {x_, y_}] :=
                {Line[{Scaled[ {-.01, -.02}, {x,y}],
                       Scaled[ {.01, -.02}, {x,y}]}],
                 Line[{Scaled[ {0, -.02}, {x,y}],
                       Scaled[ {0, .02}, {x,y}]}],
                 Line[{Scaled[ {-.01, .02}, {x,y}],
                       Scaled[ {.01, .02}, {x,y}]}]}
```

```
In[9]:= Show[ Graphics[ {
          errorbar[ {.1, .1}], errorbar[ {.9, .9}]}]]
```

## ■ 5.1.3 AspectRatio

The aspect ratio of the graphics object is set by the option `AspectRatio`.

| | |
|---|---|
| `AspectRatio -> ` *num* | show picture with a height/width = *num* |
| `AspectRatio -> Automatic` | show picture with ratio taken from data |

Settings for the `AspectRatio` option.

This uses the default `AspectRatio` of `1/GoldenRatio`, which is equal to 0.618034.

*In[10]:=* **`Show[ Graphics[ Rectangle[ {0,0}, {4,1}]]]`**

This will produce a square image.

*In[11]:=* **`Show[ %, AspectRatio -> 1]`**

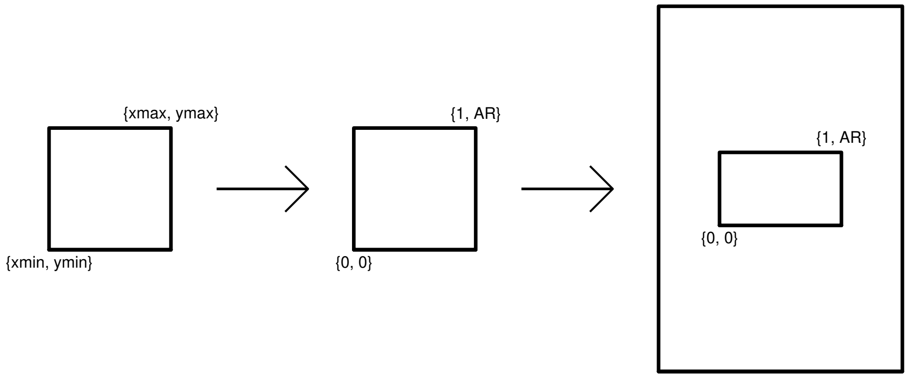
This shows the real $x$ to $y$ proportions.

*In[12]:=* **`Show[ %, AspectRatio -> Automatic]`**

Even though `AspectRatio -> Automatic` maintains the $x$ to $y$ scaling, this is often not very useful. An example is `Plot[ Tan[x], {x,0,2Pi}, AspectRatio -> Automatic]`.

## 5.2 The Coordinate Transformations

Once the `PlotRange` and `AspectRatio` have been determined the actual mapping onto the screen or paper can be made. There are two transformations in this process by which a *Mathematica* graphics object is converted into a sequence of painting instructions on a screen or paper. The first involves a transformation of user coordinates, set by `PlotRange`, into display coordinates such that $\{xmin, ymin\}$ and $\{xmax, ymax\}$ map to $\{0,0\}$ and $\{1,$ `AspectRatio`$\}$ respectively. After this the display coordinates are mapped onto the paper so as to leave room for axes, labels and margins.



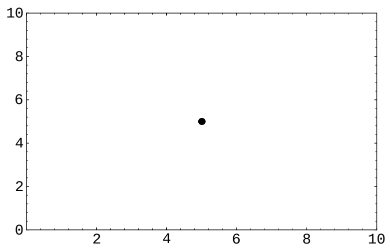

**Coordinate system transformations.**

This second transformation is quite complex and often leads to confusion. It makes the coordinate area, into which user coordinates are drawn, to be rectangular, to have a given aspect ratio and to be as large as possible. This is done by emitting all of the axes and labels, for which space must be left, before any other objects. This initial part of the stream is called the critical region. All objects which are emitted in the critical region will be included in the calculations which map the coordinate area onto the window. Text which is associated with axes, such as tick and axes labels, will be included in the critical region.

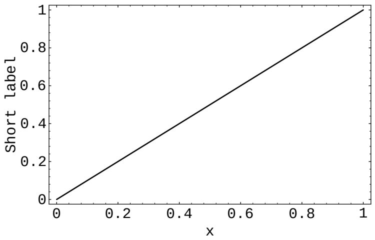
Eventually the critical region is closed and objects which are emitted will no longer be used for scaling. This is usually after the axes have been drawn. The objects which follow will be clipped against the coordinate area.

The point is mapped into the middle of the display coordinate space. Notice the frame axes that mark the bounds of the coordinate area. These are emitted in the critical region.

```
In[13]:= Show[ Graphics[ {
             PointSize[0.02],
             Point[{5,5}]},
             PlotRange -> {{0,10}, {0,10}},
             Frame -> True]]
```

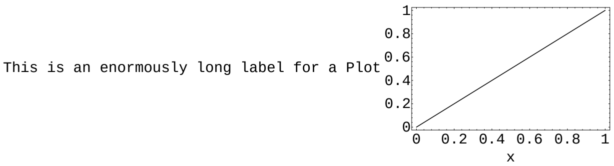


```
In[14]:= Plot[ x, {x,0,1}, Frame -> True,
            FrameLabel -> {"x", "Short label"}]
```



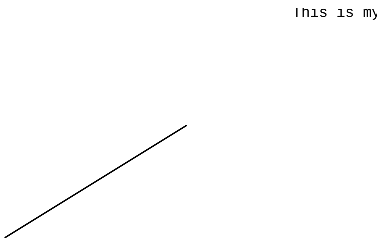

Notice that the plot has shrunk to include the label.

```
In[15]:= Show[%,
             FrameLabel -> {"x",
             "This is an enormously long label for a Plot"},
             RotateLabel -> False]
```



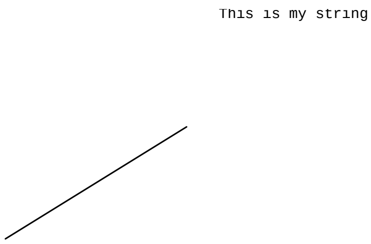

The text is cut at the point {2,2} since this is the edge of the coordinate area.

```
In[16]:= Show[ Graphics[{
          Line[{{0,0}, {1,1}}],
          Text["This is my string", {2,2}]}]]
```



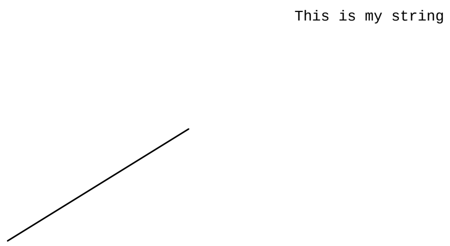
The text is complete since the offset of the `Text` primitive has moved it into the coordinate area.

```
In[17]:=
         Show[ Graphics[{
                 Line[{{0,0}, {1,1}}],
                 Text["This is my string", {2,2}, {1,0}]}]]
```



The string is allowed to spill out of the coordinate area since the text was emitted in the critical region.

```
In[18]:=
        Show[ Graphics[{
                Line[{{0,0}, {1,1}}],
                Text["This is my string", {2,2}]},
                PlotRange -> All]]
```

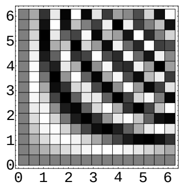



## 5.2.1 PlotRegion

The option `PlotRegion` allows one to add margins around a plot.

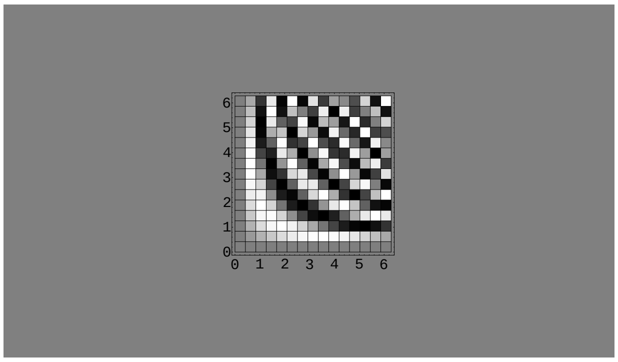
The coordinates for `PlotRegion` are `Scaled` coordinates which run from 0 to 1 in each dimension.

```
In[19]:= DensityPlot[ Sin[x y],
                        {x, 0,2Pi},
                        {y,0,2Pi},
                        PlotRegion -> {{0.25,0.75},
                                       {0.25,0.75}}]
```




Showing the background shows how the image has been shrunk.

```
In[20]:= Show[%, Background -> GrayLevel[0.5]]
```



# 5.3 Three-Dimensional Coordinates

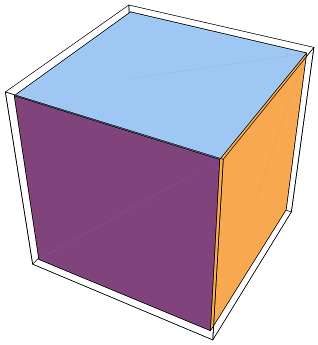
Now we shall look at the parts of the coordinate system transformation which are particular to rendering of three-dimensional objects.

## 5.3.1 BoxRatios

The `PlotRange` is determined as the first stage of processing a list of primitives. This determines the box into which the primitives will be drawn. This also is the box drawn with the `Boxed` option. The ratios of the lengths of the sides of this box are controlled with the option `BoxRatios`. This is the three-dimensional analog of the `AspectRatio`.

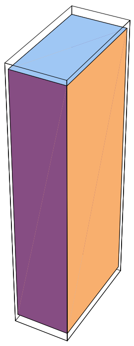
This shows a cubical image since the ratios of the box are all the same.

```
In[21]:= Show[ Graphics3D[
              Cuboid[{0,0,0}, {1,2,5}],
              BoxRatios -> {1,1,1}]]
```



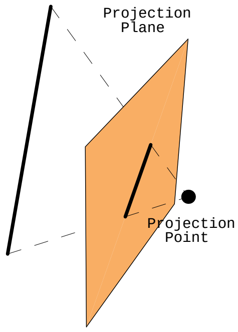
This shows the actual ratios of the `Cuboid` dimensions.

```
In[22]:=
     Show[ Graphics3D[ Cuboid[{0,0,0}, {1,2,5}],
              BoxRatios -> Automatic]]
```



## 5.3.2 Projection

The process by which a coordinate system of dimension $n$ is transformed into one of dimension less than $n$ is called a projection. In order to view a three-dimensional object on a computer screen or a printed page it is necessary to project it from three to two dimensions.

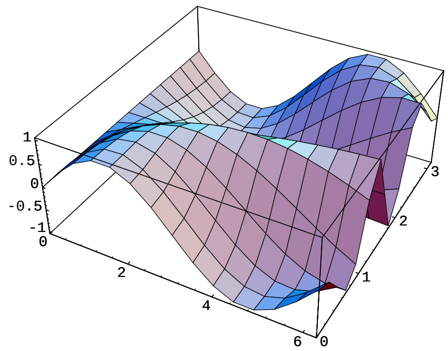



**Projection of a line.**

The components which define a projection are a projection plane, which defines the two-dimensional space to which we wish to move, and a projection point. The projection is made by forming the straight line from the projection point to the object we wish to project. Where this line cuts the projection plane is the projection of that object.

In *Mathematica* the projection point is given by the option `ViewPoint` and the projection plane is defined by the plane which contains the `ViewCenter` and is perpendicular to the vector from the `ViewPoint` to the `ViewCenter`, the line-of-sight vector.

```
In[23]:= Plot3D[ Sin[ x  Cos[y]], {x,0,2Pi}, {y,0,Pi}]
```



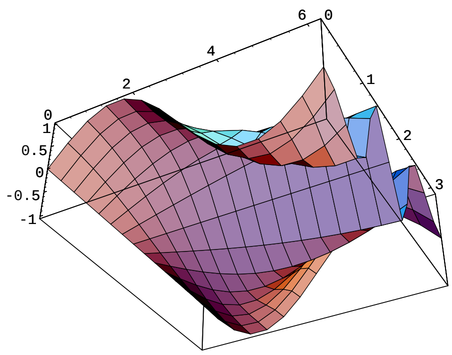

This is the default `ViewPoint` which was used.

```
In[24]:= FullOptions[%, ViewPoint]

Out[24]= {1.3, -2.4, 2.}
```

This reflects the viewpoint in the $z$ direction to show us the underside of the surface.

```
In[25]:= Show[ %%, ViewPoint -> % {1,1,-1}]
```

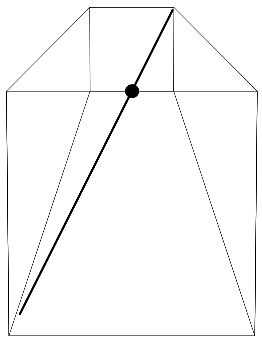


ViewPoint is given in a special scaled coordinate system in which the origin is the center of the box. The units are scaled so that the longest side of the box is unity and the other dimensions follow their BoxRatios. Note that the ViewPoint must be outside of the box. This choice of coordinate system allows us to gain an interesting view of a wide variety of three-dimensional objects from one default ViewPoint.

ViewCenter is used to set the projection plane as is specified above. It is also used to specify which part of the projected image will appear in the center of the window. The coordinate system of ViewCenter is Scaled coordinates, the lower-left corner of the box is {0,0,0} and the upper-right corner is {1,1,1}. With the default setting ViewCenter -> Automatic *Mathematica* chooses the projection plane to pass through the center of the box but places the center of the projection so that the image is as large as possible. ViewCenter -> {0.5, 0.5, 0.5} uses the same projection plane but puts the point Scaled[{0.5, 0.5, 0.5}] at the center of the drawing region.

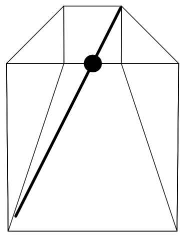
This uses the default ViewCenter -> Automatic.

```
In[26]:= Show[ Graphics3D[ {
            PointSize[0.05],
            Point[{.5,5,.5}],
            Thickness[0.008],
            Line[{{0,0,0}, {1,10,1}}]},
            ViewPoint -> {0, -1, .1}]]
```



See how the image has shrunk since the center of the image is at the center of the drawing region here.

```
In[27]:= Show[ %, ViewCenter -> {0.5, 0.5, 0.5}]
```



### SphericalRegion

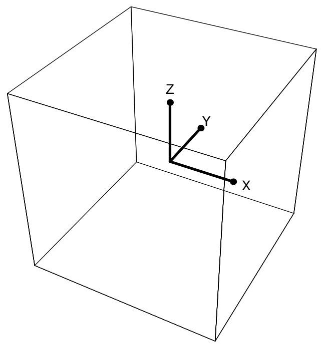
With the default setting of SphericalRegion -> False the image is scaled so that the projection of the box fits into the window. Setting SphericalRegion -> True scales so that the sphere which can be transcribed around the box fits the window. This is desirable if one wishes to animate the rotation of a graphic by changing the ViewPoint.

### ViewVertical

The option ViewVertical is provided to allow the user to specify which direction in the image will point up in the projected image. The default value of this is ViewVertical -> {0,0,1} so the *z* direction (at the ViewCenter) points up.

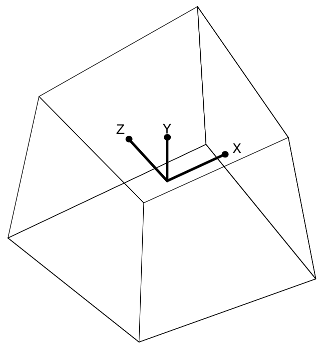
The Z points up here.

```
In[28]:= Show[
           Graphics3D[ {PointSize[.02],
             Thickness[.008], Line[ {{0,0,0}, {1,0,0}}],
             Point[{1,0,0}], Text[ "X", {1.2,0,0}],
             Line[ {{0,0,0}, {0,1,0}}], Point[{0,1,0}],
             Text[ "Y", {0,1.2,0}], Line[ {{0,0,0}, {0,0,1}}],
             Point[{0,0,1}], Text[ "Z", {0,0,1.2}]},
             PlotRange -> {{-1.5,1.5}, {-1.5,1.5},
                                          {-1.5,1.5}}]]
```




This makes Y point up.

```
In[29]:= Show[ %, ViewVertical -> {0,1,0}]
```




### AspectRatio

With the specification of `BoxRatios`, `ViewPoint`, `ViewCenter`, `ViewVertical` and `SphericalRegion` the projection from three-dimensional coordinates is complete. However, the relative scaling of $x$ and $y$ in the projection plane can be altered with the option `AspectRatio`. This defaults to `Automatic`, which maintains the relative proportions which were initially present. Under most circumstances this is desirable.