The Big Faceless Graph Library

Welcome to the Big Faceless Graph Library example document.

The following pages show a number of graphs created with the Big Faceless Graph Library (available from **http://big.faceless.org/products/graph**), which should give you an idea of what the library is capable of. Among the highlights are:

· Full 3D engine with shading for more realistic graphs.

· Render to java.awt.Image, PNG, PDF and other formats

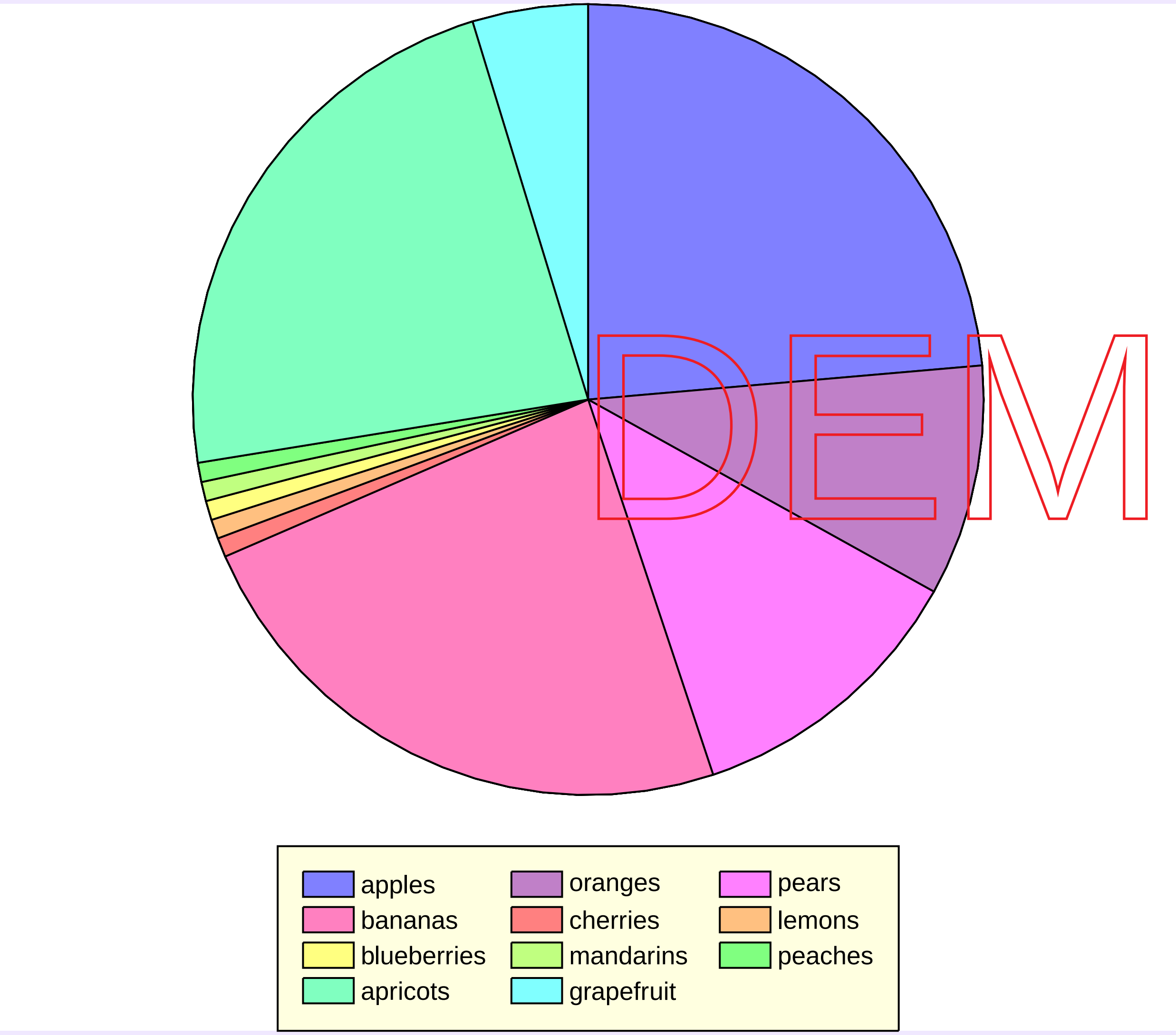
· Extensible formatting for easy graphing of currencies, dates and more.

· Curve smoothing functions for cleaner graphs

The examples that follow all show the graph, a brief description of what's being shown, and a list of all the options that were set to generate the graph.

More documentation is available at the product homepage.

This document was created with the Big Faceless PDF library, available separately from **http://big.faceless.org/products/pdf**

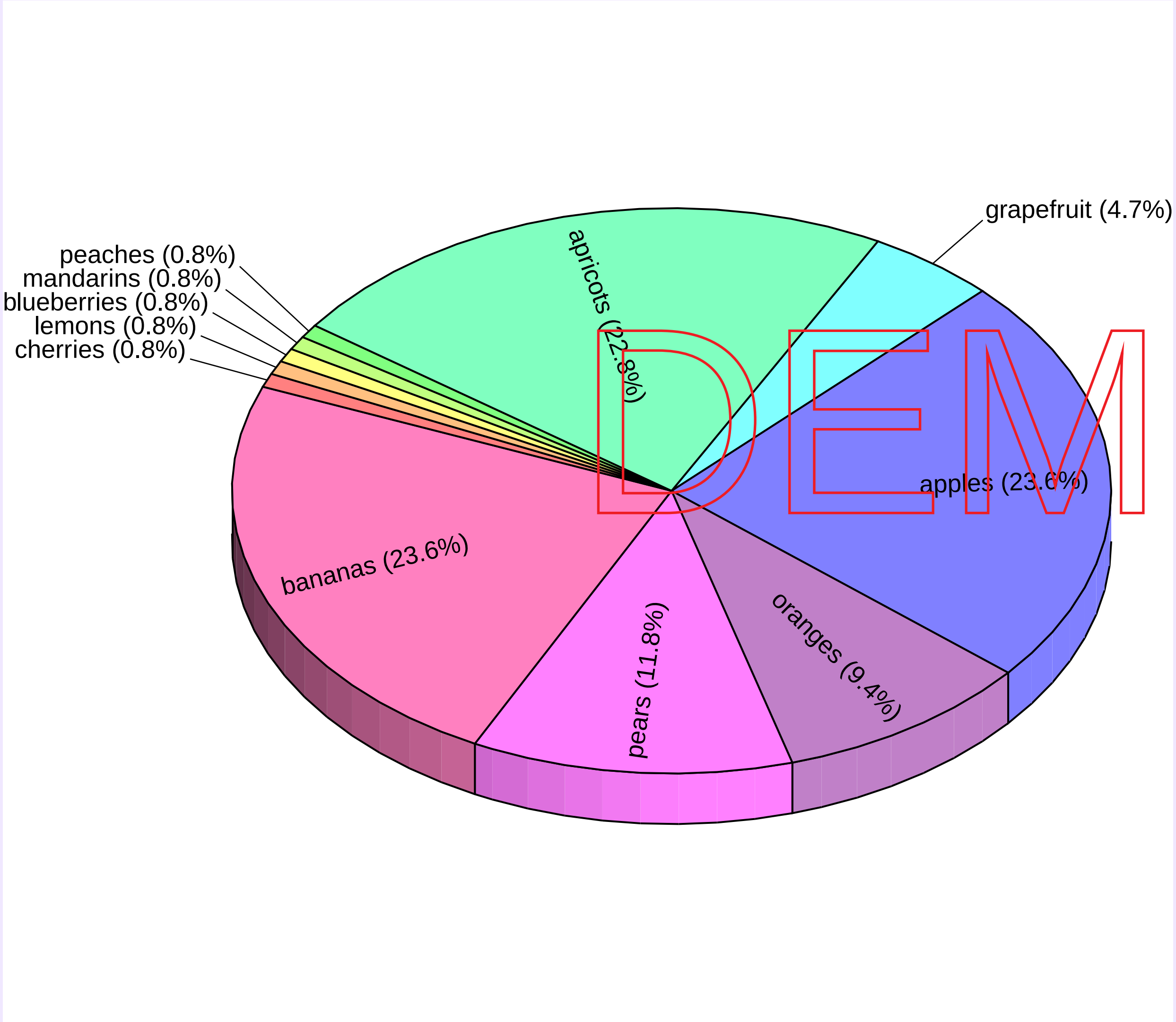


A Plain Fruit Pie

This is the default layout for a Pie Graph. The key is in the default position, below the graph. The order of the key and the slices reflects the order in which they were added to the graph - first apples, then oranges, pears, bananas, cherries lemons and so on. The slices are drawn clockwise, and by default the first slice starts at the 12 o'clock position. This can be changed using the "YRotation" option, which you'll see in the next example.

All the following Pie Graphs are drawn with exactly the same data, so the changes you see are the result of different options being set. Lets see some more interesting graphs.

```
new PieGraph()
```



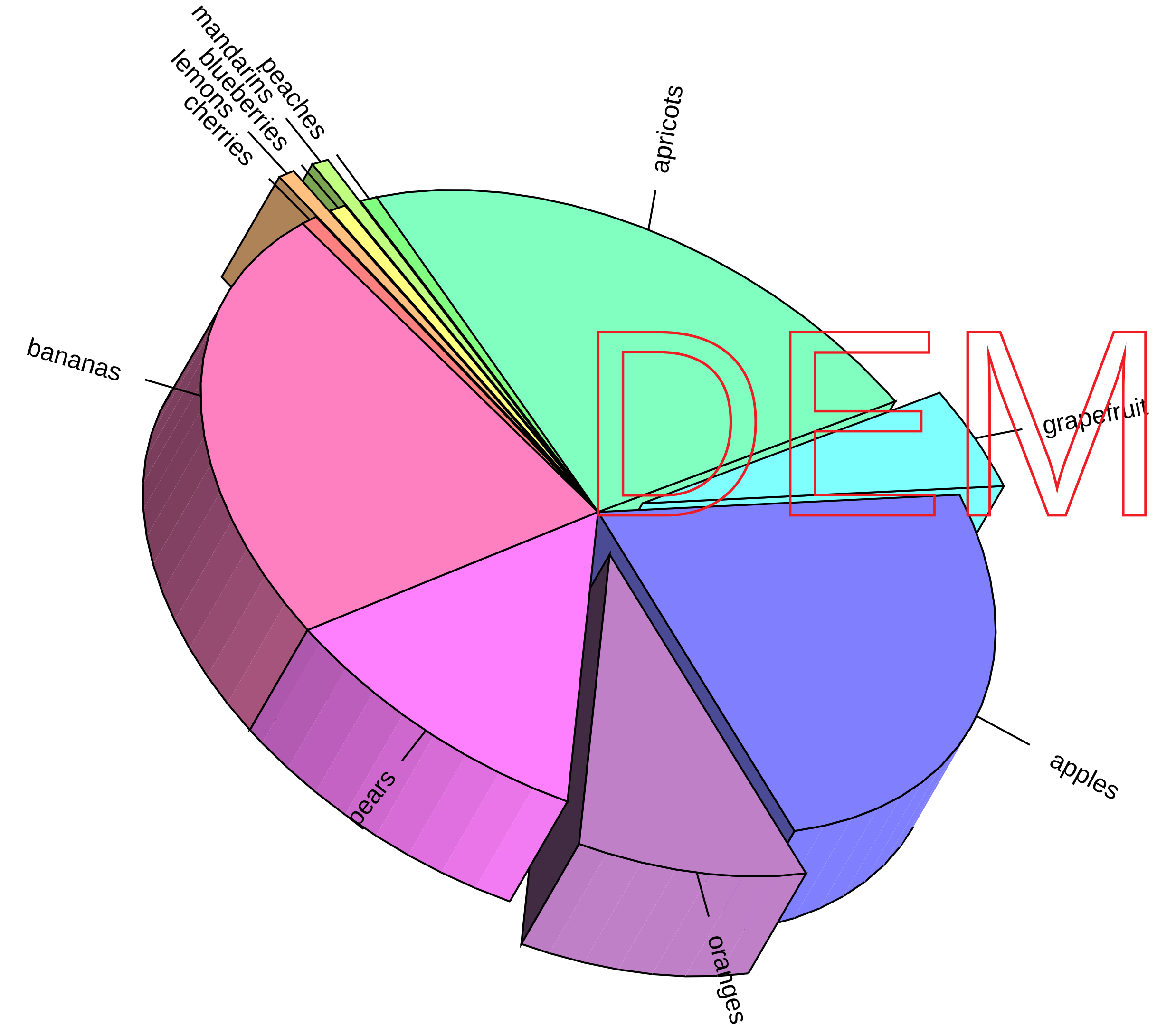
A 3D look, for a more substantial Pie.

This is a Pie graph that has been rotated to give a 3D look. The YRotation option causes the first slice to begin at a different angle from 12 o'clock. The XRotation option means that the graph is tipped backwards, so we can see the bottom edge of the pie.

We've also moved the key so that it's displayed on the slice (only if the slice is greater than 5% of the whole) and it's rotated. If the slice is less than 5% of the whole, it's displayed next to the graph and the end of a line. We're also showing the percentage for each slice.

Notice how the graph doesn't fill the entire box, but sits in the middle. This is the result of the FixedAspectRatio flag, which for Pie Graphs defaults to true. We'll show you what happens if you play with that in a later example.

```
new PieGraph()
optionYRotation(45.0)
optionXRotation(50.0)
optionDisplayKey(KEY_ROTATED_INNER_FLAT_OUTER)
optionDisplayPercentage(PERCENTAGE_WITH_KEY)
optionOuterKeyPercentage(5.0)
```



3D Pie Graph with Outer Keys

This Graph is similar to the previous graph, but with a couple of other options set.

First, we've highlighted the Citrus Fruit, by extending the relevent slices by 20%.

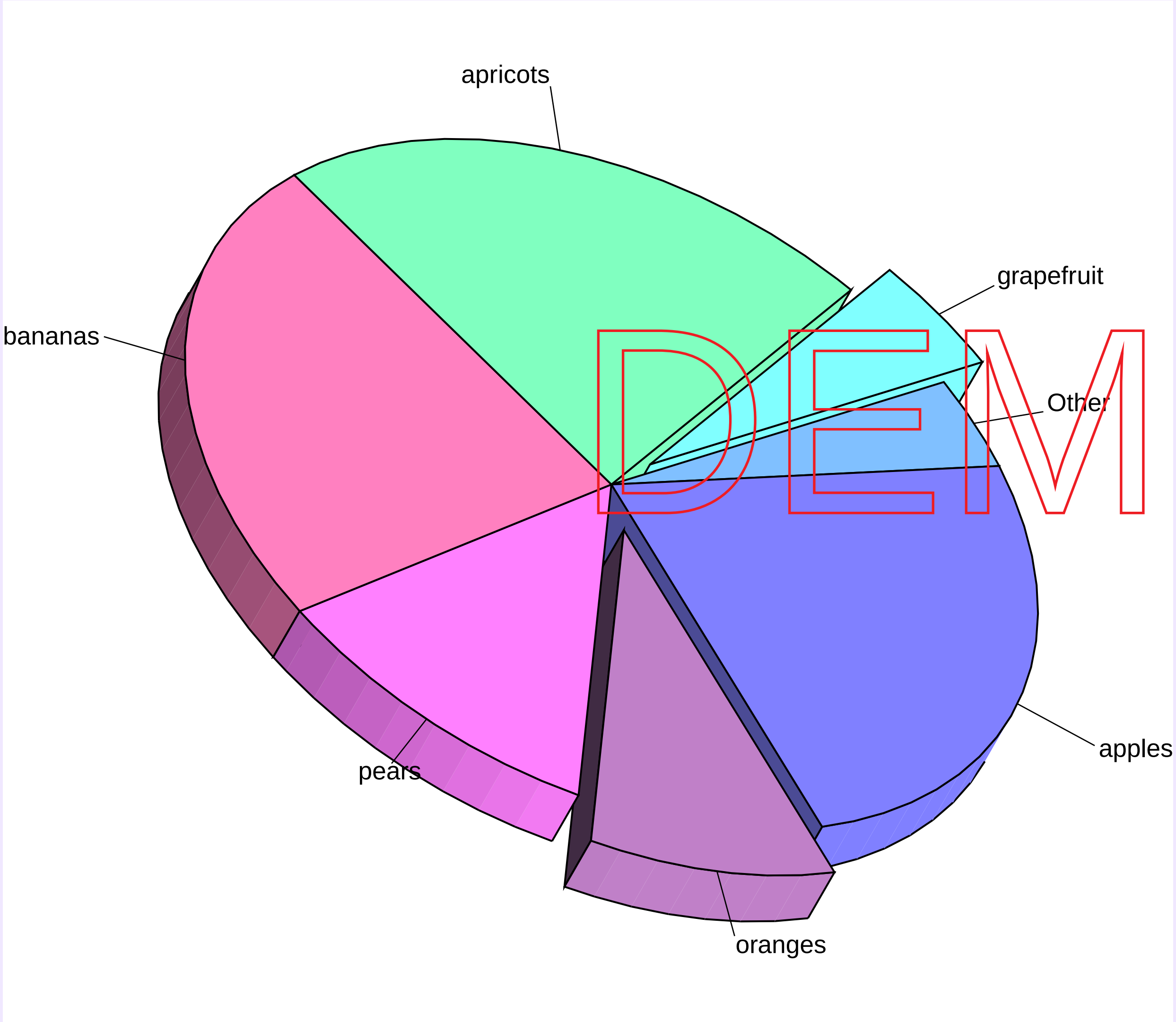
Secondly, we've added some ZRotation, which is why we can now see the left side of the graph as well as the bottom.

We've changed the Height of the graph from the default of 20, which is why all the slices are thicker.

Finally, we've changed the Key Style to ROTATED_OUTER. The results of this last one are obvious.

The data that we're using has a number of small slices, in the top left corner on this graph. In the next example we'll use the "Other Slice" feature to remove them automatically.

```
new PieGraph()
extendslice("grapefruit", 20)
extendslice("lemons", 20)
extendslice("oranges", 20)
extendslice("mandarins", 20)
optionYRotation(45.0)
optionXRotation(50.0)
optionZRotation(30.0)
optionHeight(35.0)
optionDisplayKey(KEY_ROTATED_OUTER)
```



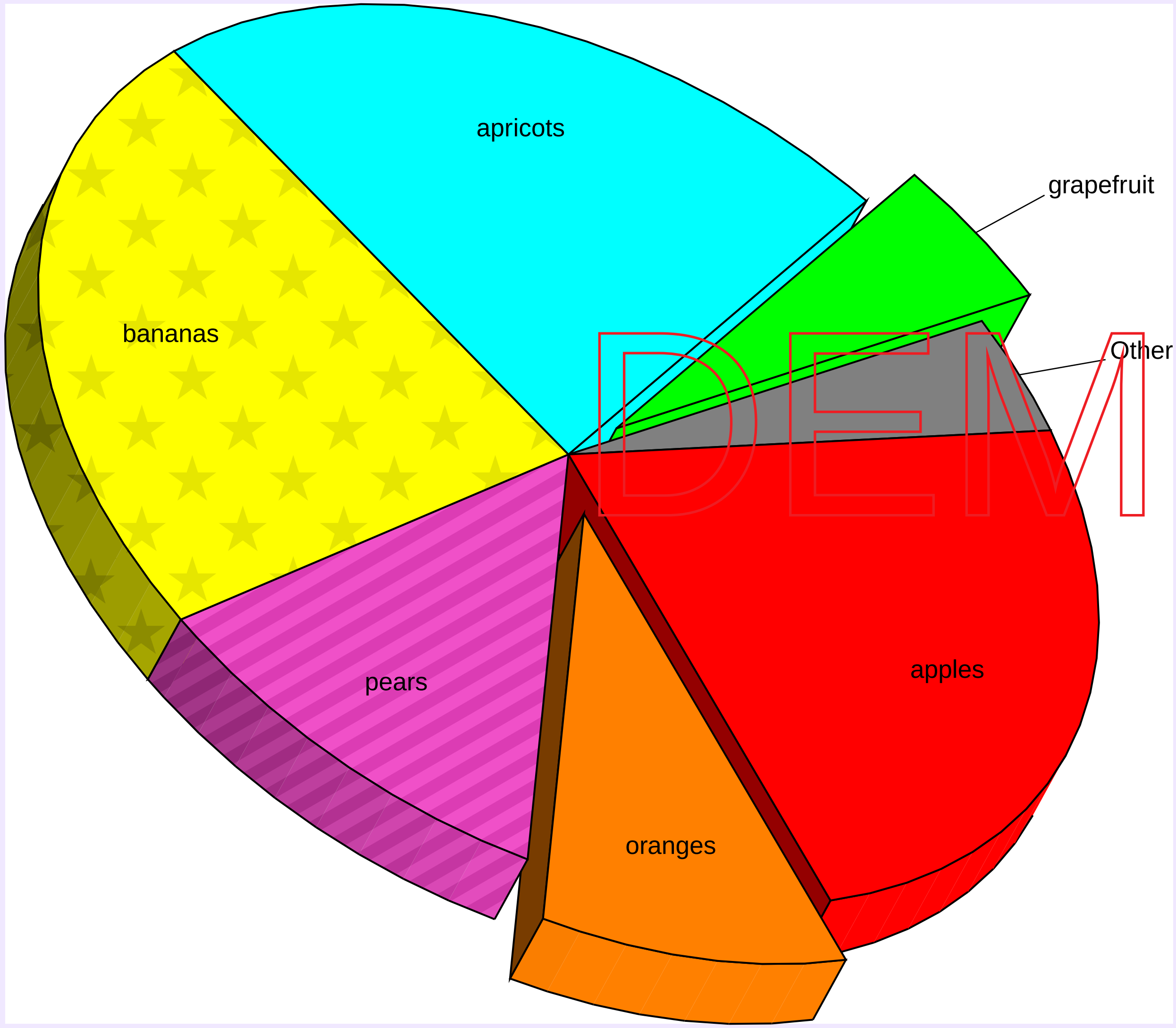
Pie Graph with an 'Other' Twist

This example shows the FLAT_OUTER style of key. It's the same data as the previous graph, but has less slices. Why?

We're using the 'OtherPercentage' option to combine the smaller slices (in this case, anything less than 3%) into a single slice, for clarity.

It's a good idea to set this to a low value by default, especially if you're generating the graphs automatically from a datasource - although we try and keep the graph looking nice, it's almost impossible with hundreds of tiny slices stacked next to eachother.

```
new PieGraph()
extendslice("grapefruit", 20)
extendslice("lemons", 20)
extendslice("oranges", 20)
extendslice("mandarins", 20)
optionYRotation(45.0)
optionXRotation(50.0)
optionZRotation(30.0)
optionOtherPercentage(3.0)
optionDisplayKey(KEY_FLAT_OUTER)
```



A Stretched, Bright Pie

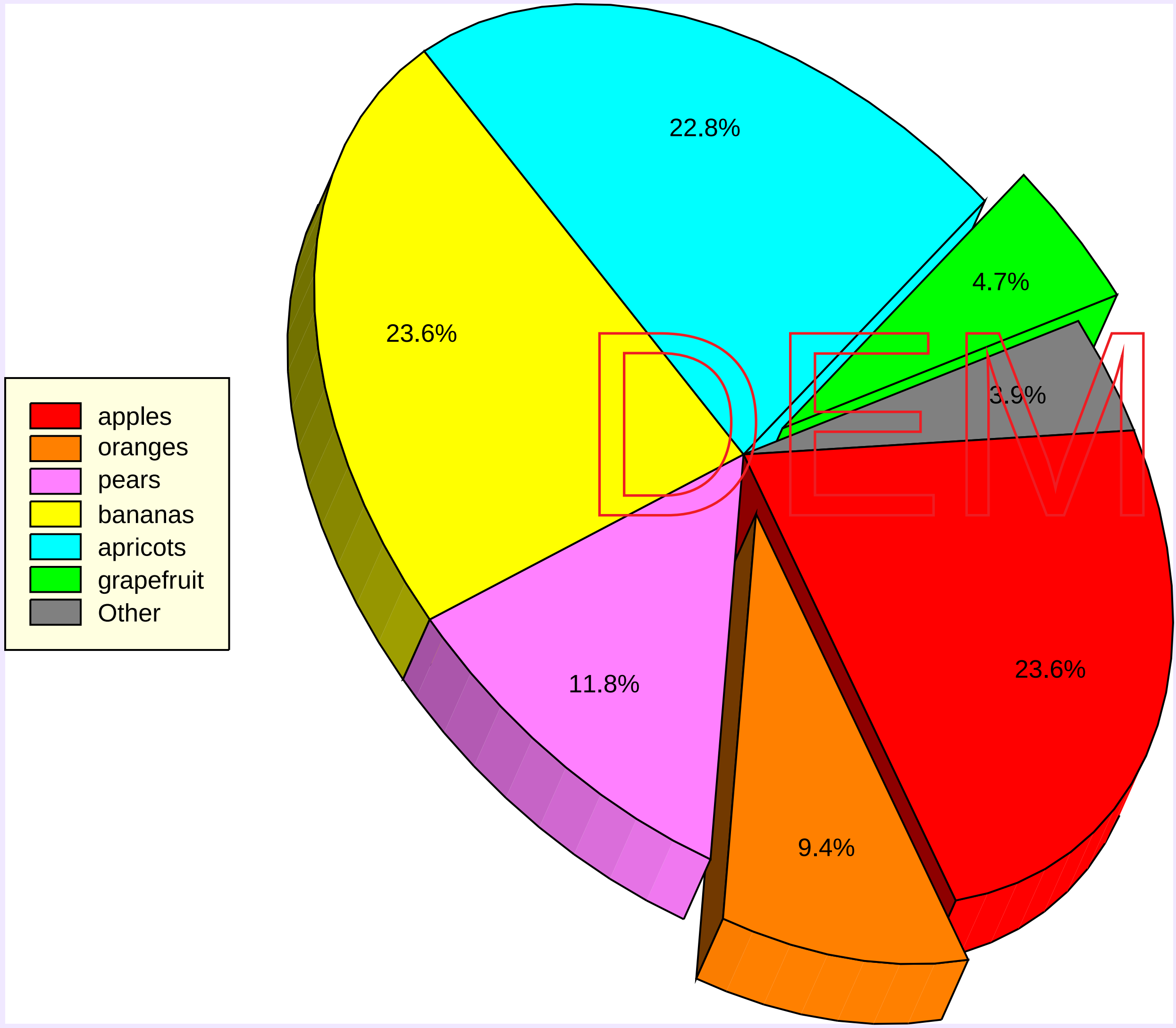
This is the same graph as the last example, with three changes.

First, we've changed the colours (all the graphs up until now have used the default colors). Two of the slices we've patterned by using the "ColorPattern" class from our companion PDF library product - this feature is ONLY available when rendering to PDF.

Second, we've changed the key style again, this time to FLAT_INNER_FLAT_OUTER. The keys will be displayed next to the slice only if they are smaller than the OuterKeyPercentage.

Finally, we've turned off the FixedAspectRatio option, so the graph fills the entire box - the penalty for this is it's no longer exactly circular. With extreme rotations and narrow output windows, this can result in a graph that is difficult to read.

```
new PieGraph()
extendslice("grapefruit", 20)
extendslice("lemons", 20)
extendslice("oranges", 20)
extendslice("mandarins", 20)
optionYRotation(45.0)
optionXRotation(50.0)
optionZRotation(30.0)
optionOtherPercentage(3.0)
optionOuterKeyPercentage(5.0)
optionDisplayKey(KEY_FLAT_INNER_FLAT_OUTER)
optionFixedAspectRatio(false)
```

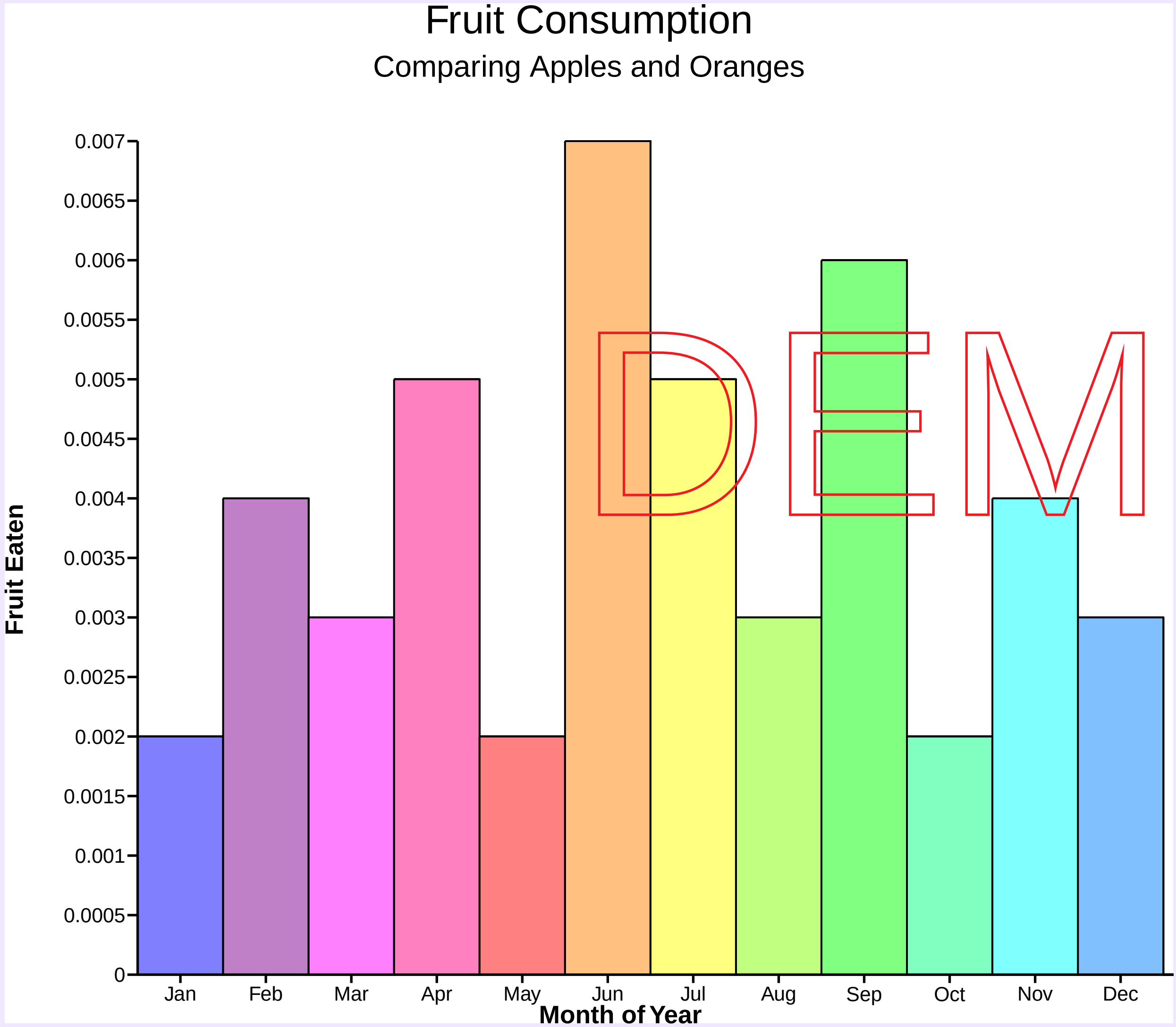


Separating the Percentage from the Key

The change in aspect ratio is more pronounced here, now that we'ved moved the key away from the Pie.

We're displaying percentages again, but by specifying PERCENTAGE_INLINE, the percentages are placed on the graph. We've forced them to be displayed on the slice by setting OuterKeyPercentage to 0. If we'd set it to 100, all the percentages would be displayed next to the slices instead.

```
new PieGraph()
extendslice("grapefruit", 20)
extendslice("lemons", 20)
extendslice("oranges", 20)
extendslice("mandarins", 20)
optionYRotation(45.0)
optionXRotation(50.0)
optionZRotation(30.0)
optionOtherPercentage(3.0)
optionOuterKeyPercentage(0.0)
optionDisplayKey(KEY_BOXED_LEFT)
optionDisplayPercentage(PERCENTAGE_INLINE)
optionFixedAspectRatio(false)
```



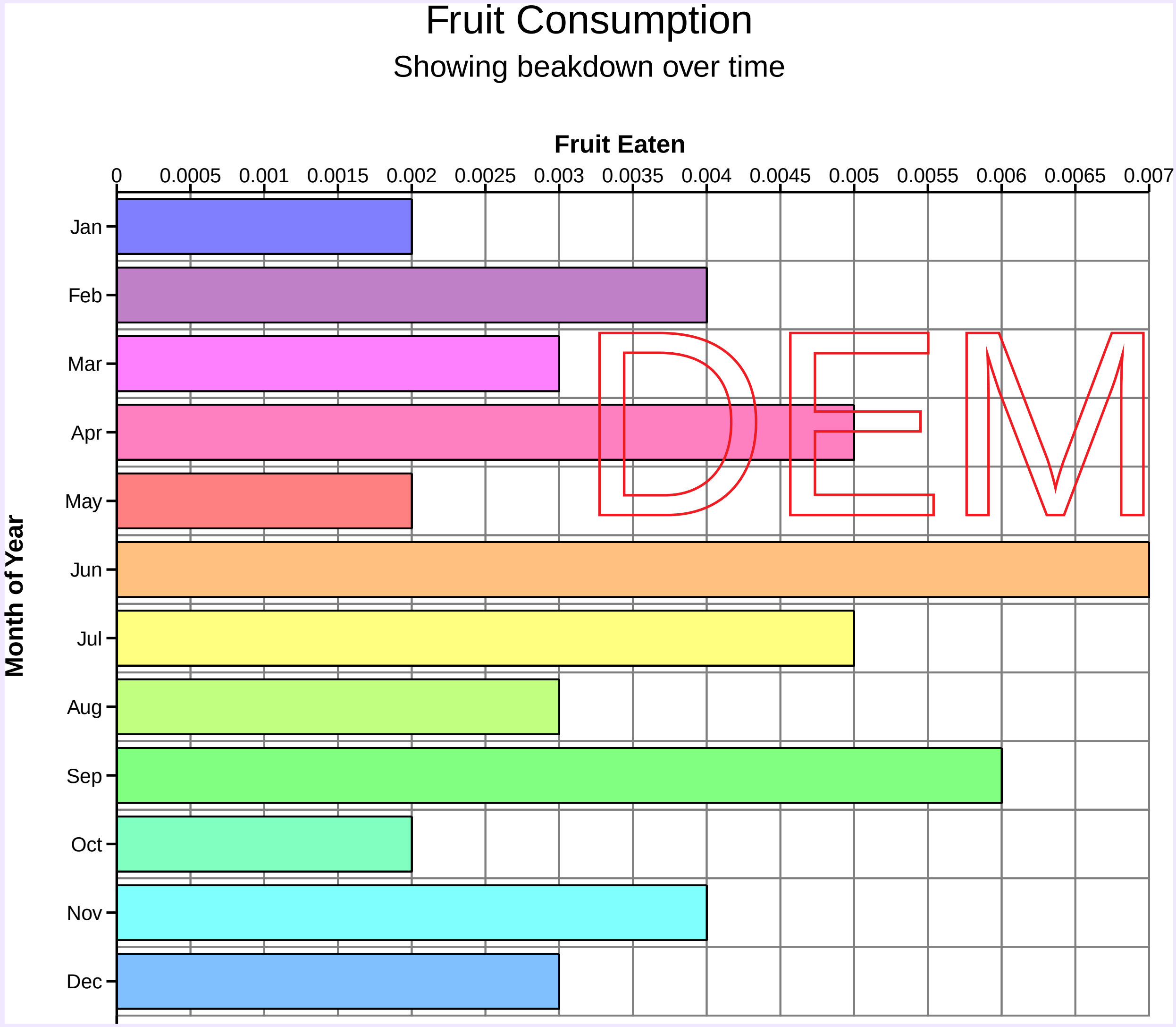
A Simple Bar Graph

This is the default layout for the simplest type of Bar Graph, which just plots single bars along the X axis.

We've played with the scale a bit here - notice that the values are tiny fractions rather than whole numbers. The library can handle any scale intelligently.

The basic bar graph looks pretty bland, so lets set some options.

```
new BarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Comparing Apples and Oranges")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
```

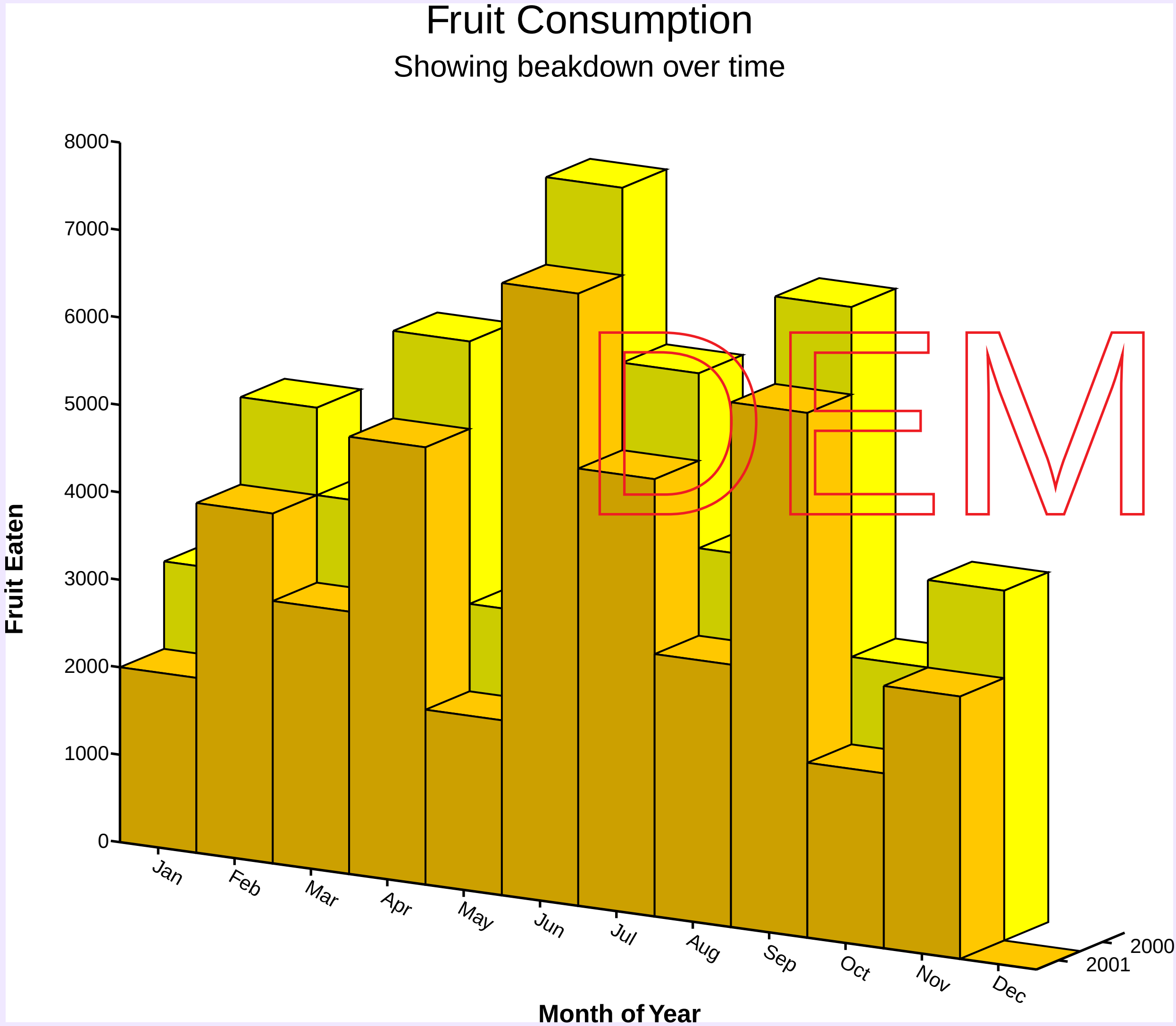


Horizontal Bars

This is the same graph as before, but spun around 90 degrees. We've set the BarWidth option to 0.8, so the bars don't take up the entire column (or row in this case). We've also added a back wall.

Horizontal graphs also work well with TowerBarGraphs

```
new BarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionZRotation(90.0)
optionZWallStyle(new Style(null, #808080))
optionBarWidth(0.8)
```



Fruit Consumption

Showing beakdown over time

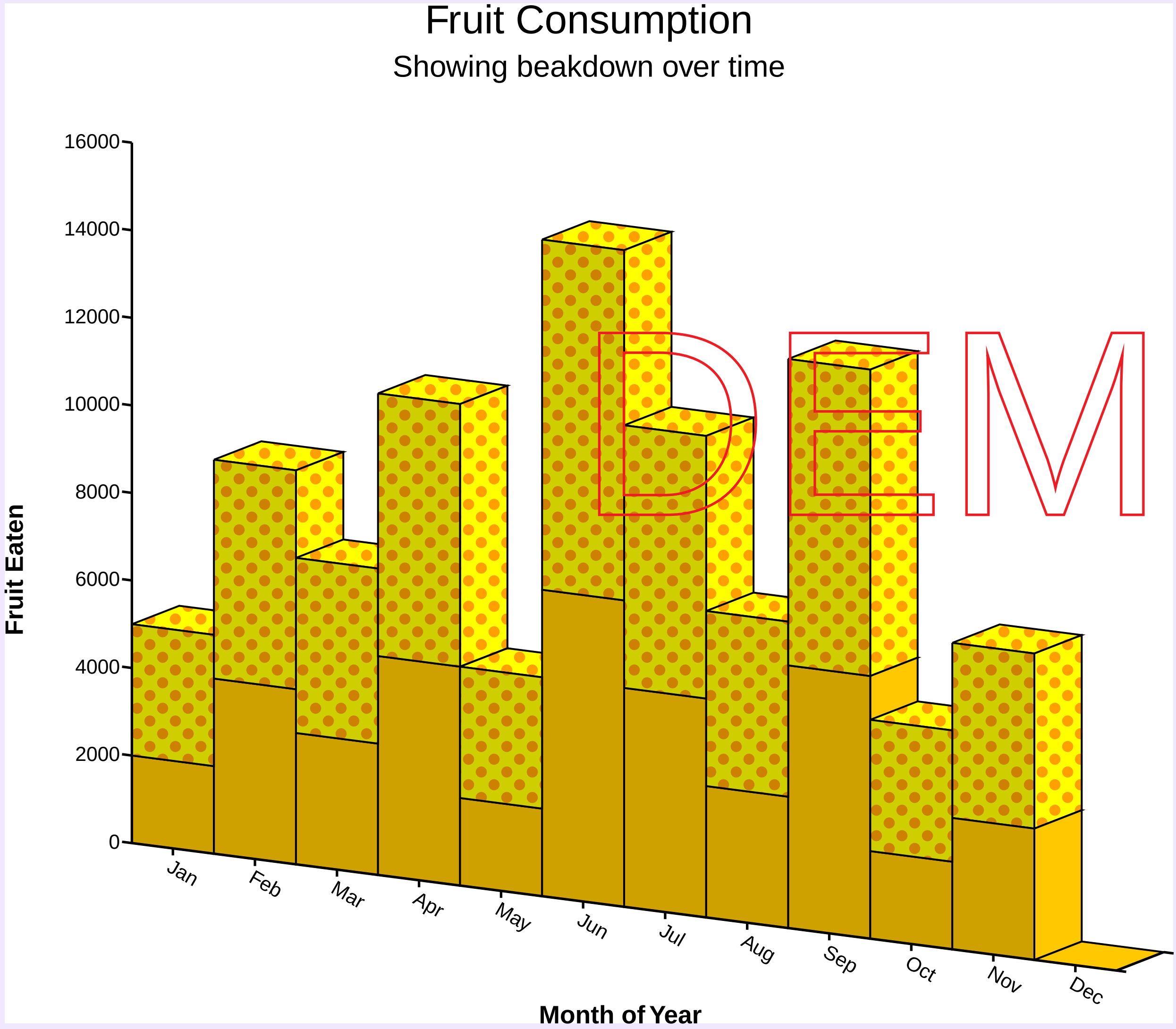
A Deeper Bar Graph

Here's another Bar Graph with a new dimension - depth.

Notice the difference between December 2001, where the value is zero, and December 2000, where there is no value at all.

Observant viewers will also notice the labels on the X-Axis have been rotated by 30 degrees - particularly useful if you've got long labels.

```
new DepthBarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionYRotation(30.0)
optionXRotation(20.0)
optionXAxisTextRotation(30.0)
```

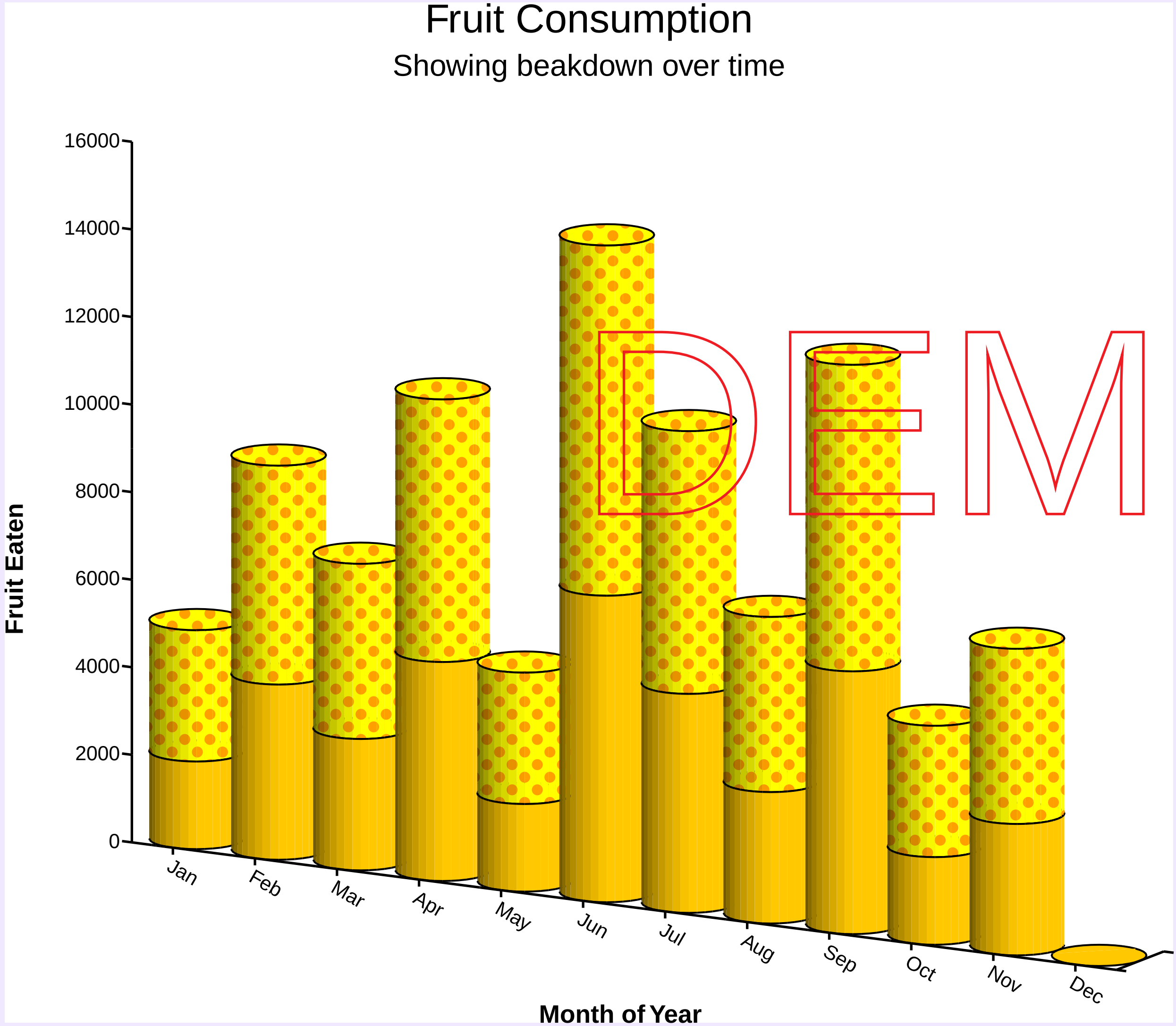


A Taller Bar Graph

Here's the third type of Bar Graph, a tower graph.

The data and all the settings are identical to the last graph, although we've once again changed the color to a ColorPattern (a feature only available with our companion PDF library product)

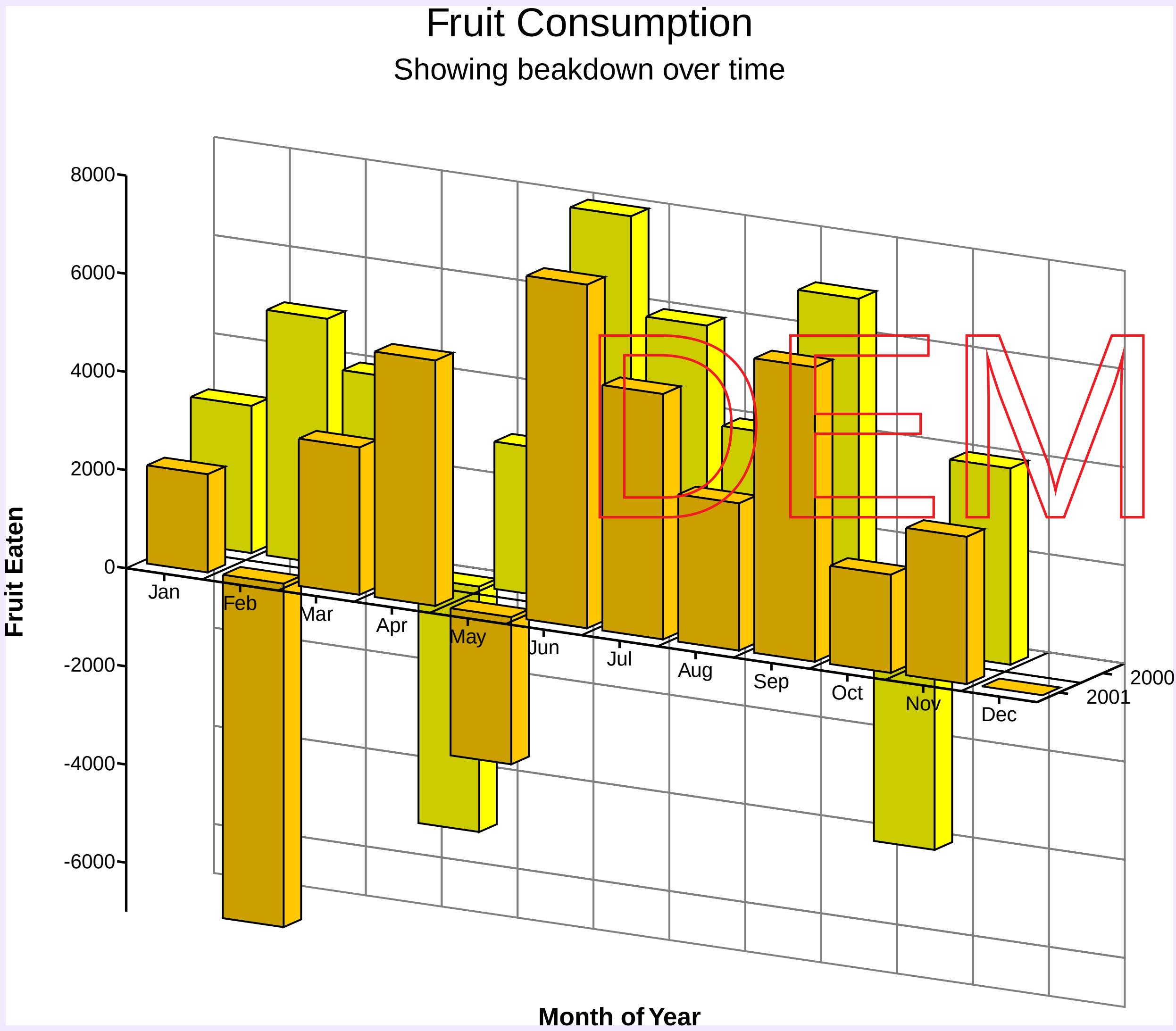
```
new TowerBarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionYRotation(30.0)
optionXRotation(20.0)
optionXAxisTextRotation(30.0)
```



A Rounder, Taller Bar Graph

This graph is identical to the previos one, except we've set the RoundBars option - the effects of which are obvious. Any bar graph except for a MultiBarGraph can be drawn with round bars - although it's not nearly as fast as using rectangles

```
new TowerBarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionYRotation(30.0)
optionXRotation(20.0)
optionRoundBars(true)
optionXAxisTextRotation(30.0)
```

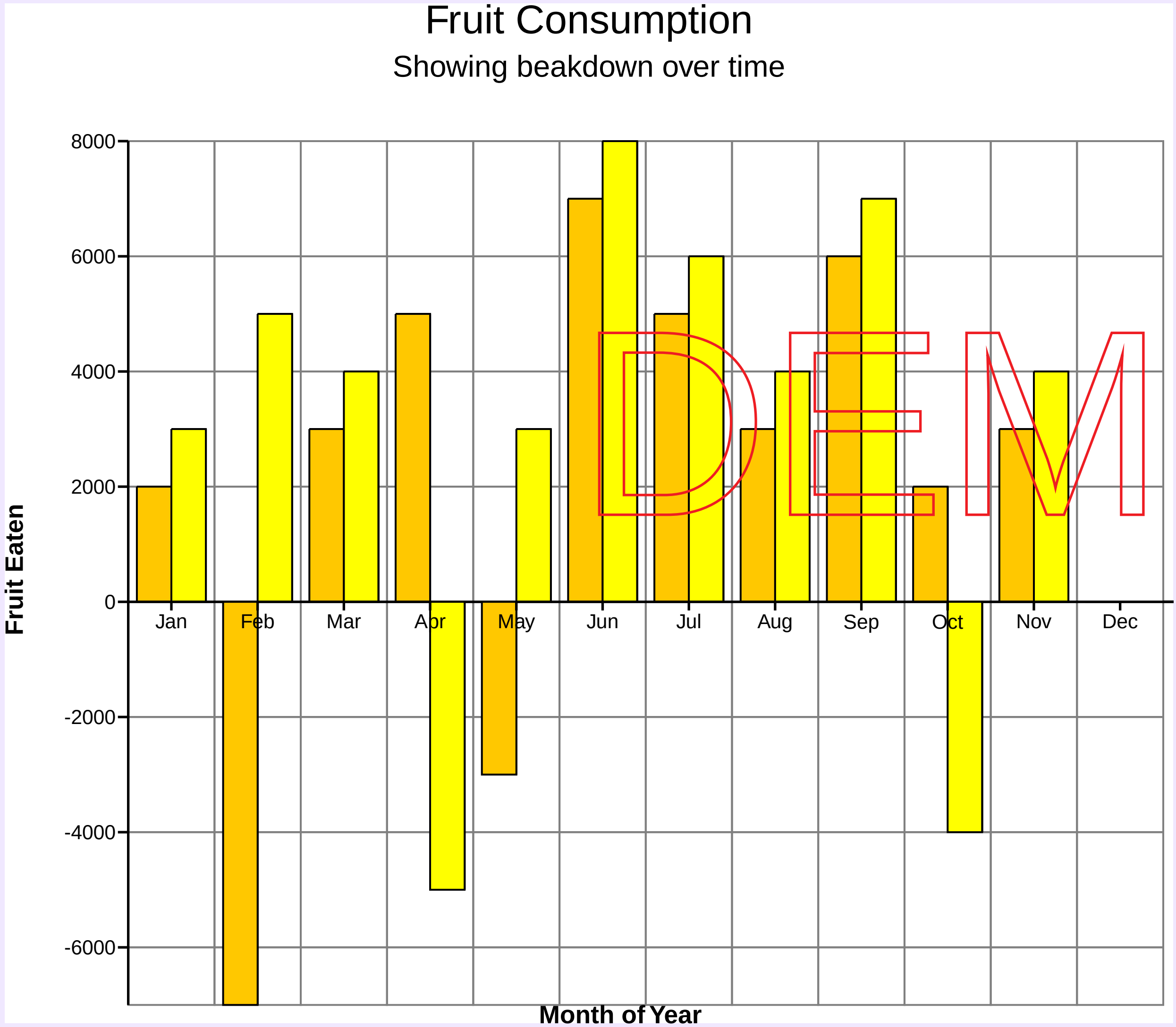


Sub-Zero Bar Graphs

Bar graphs can swing both sides of the zero axis. Most of the settings are identical to the last two graphs, but we've tweaked one or two options. First the bars have been resized to be narrower and shallower - bar width and depth can be set independently. Also notice the floor of the graph, which we've moved from the default position at the bottom of the Y axis to where y=0, by setting the YAxisAtZero option to true.

In this example the floor is transparent - optional, but useful in this situation so we can see the descending bars.

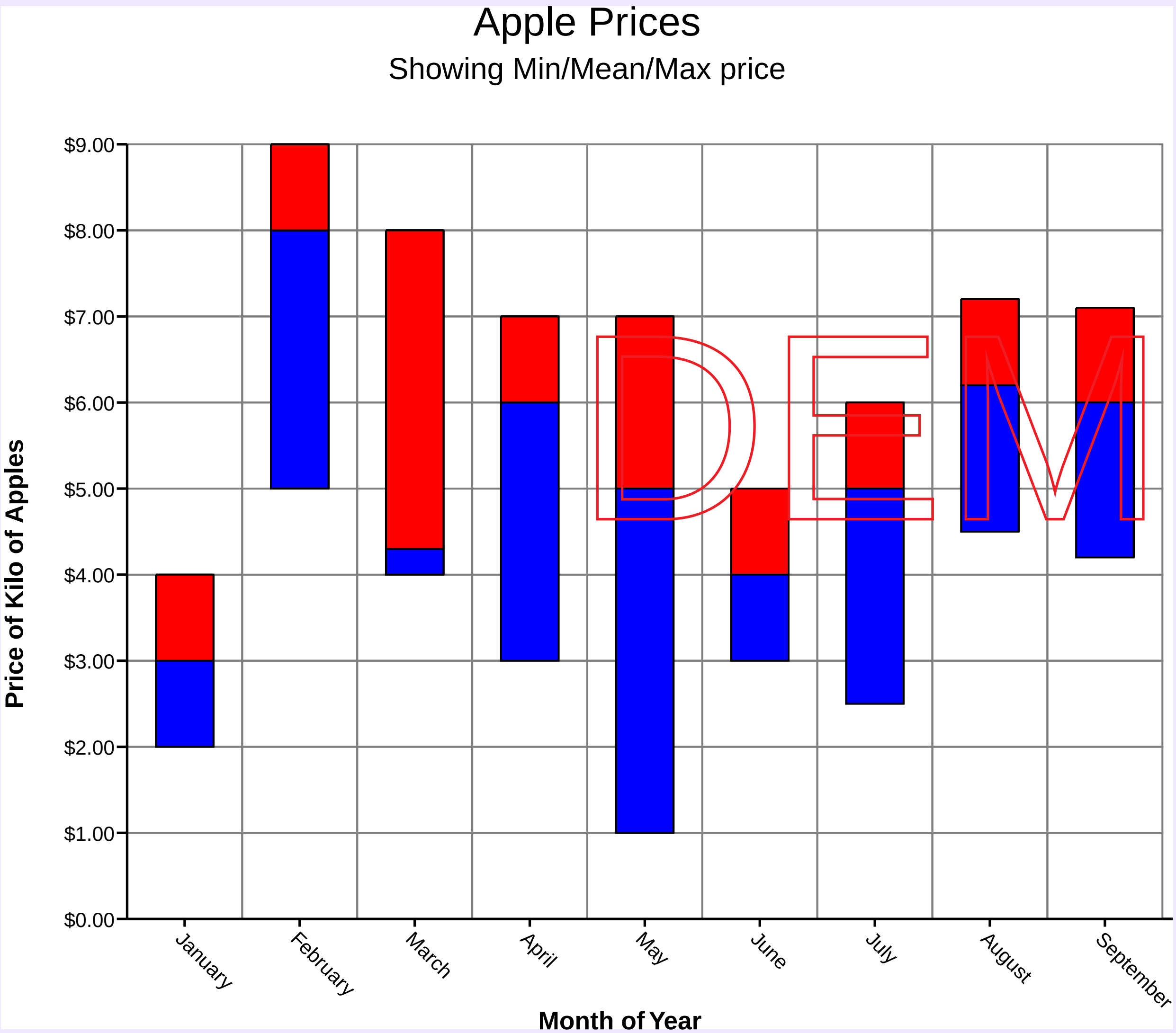
```
new DepthBarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionZWallStyle(new Style(null, #808080))
optionFloorStyle(new Style(null, #000000))
optionYRotation(30.0)
optionXRotation(20.0)
optionBarWidth(0.8)
optionBarDepth(0.4)
optionYAxisAtZero(true)
```



Multiple Sub-Zero Bars

If you have to work in two dimensions, a MultiBarGraph is a good alternative to a DepthBarGraph - plotting the same data on a single X axis instead of the X and Z axis. Here's the same graph as the previous example, minus the rotation.

```
new MultiBarGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Showing beakdown over time")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Fruit Eaten")
optionZWallStyle(new Style(null, #808080))
optionFloorStyle(new Style(null, #000000))
optionBarWidth(0.8)
optionBarDepth(0.4)
optionYAxisAtZero(true)
```

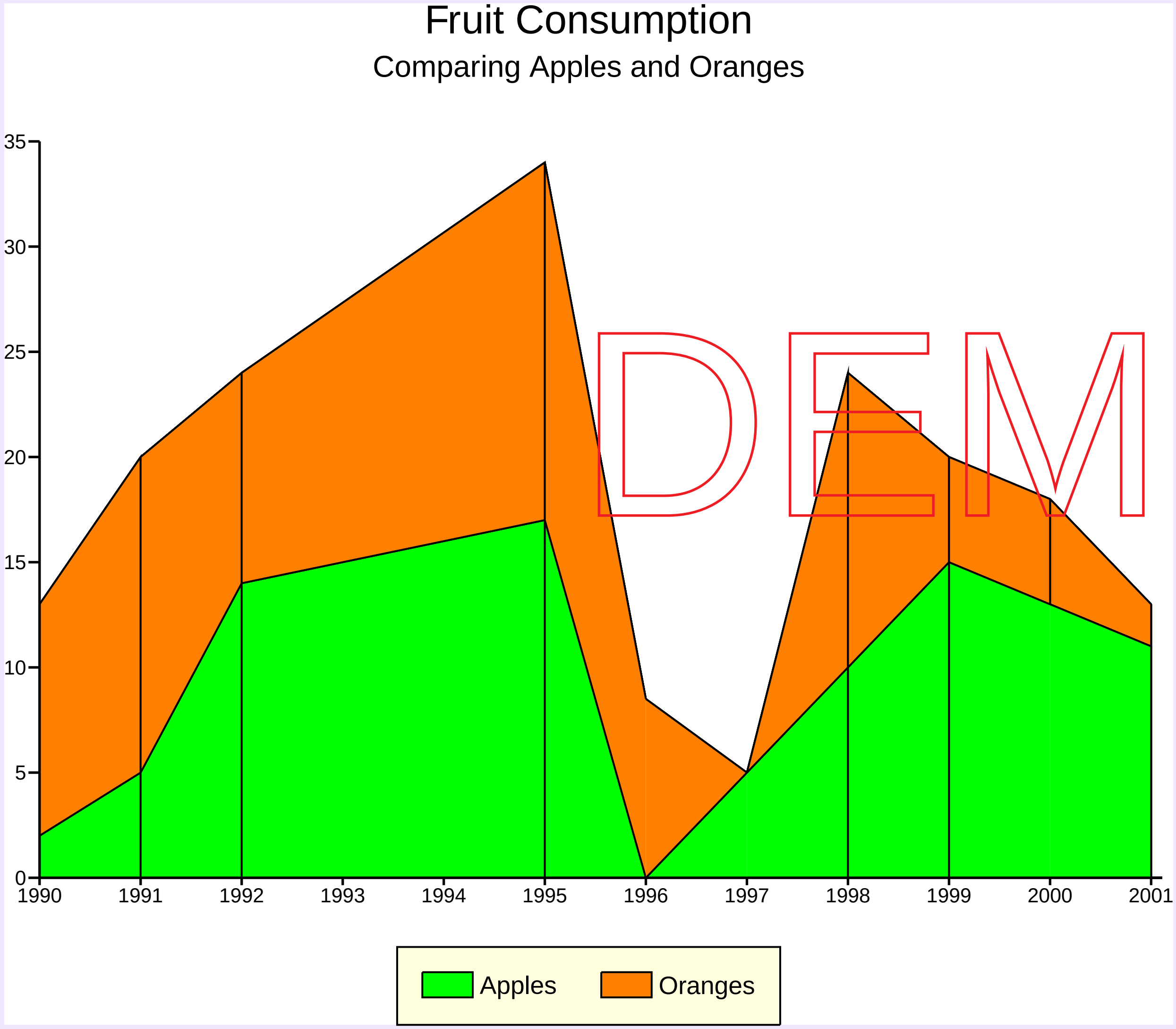


Floating Bars

A requirement of the other Bar Graphs is that all the bars grow from zero. Here's another type of Bar Graph that doesn't have that limitation - a floating bar graph. These are useful for showing a range of values, often with the center showing the average.

This is an interesting example for another reason, because it shows the use of a "Formatter", which is used to format the Y axis values as a currency.

```
new FloatingBarGraph()
optionTitle("Apple Prices")
optionSubTitle("Showing Min/Mean/Max price")
optionXAxisLabel("Month of Year")
optionYAxisLabel("Price of Kilo of Apples")
optionZWallStyle(new Style(null, #808080))
optionBarWidth(0.5)
optionXAxisTextRotation(45.0)
optionYFormatter(new CurrencyFormatter())
```

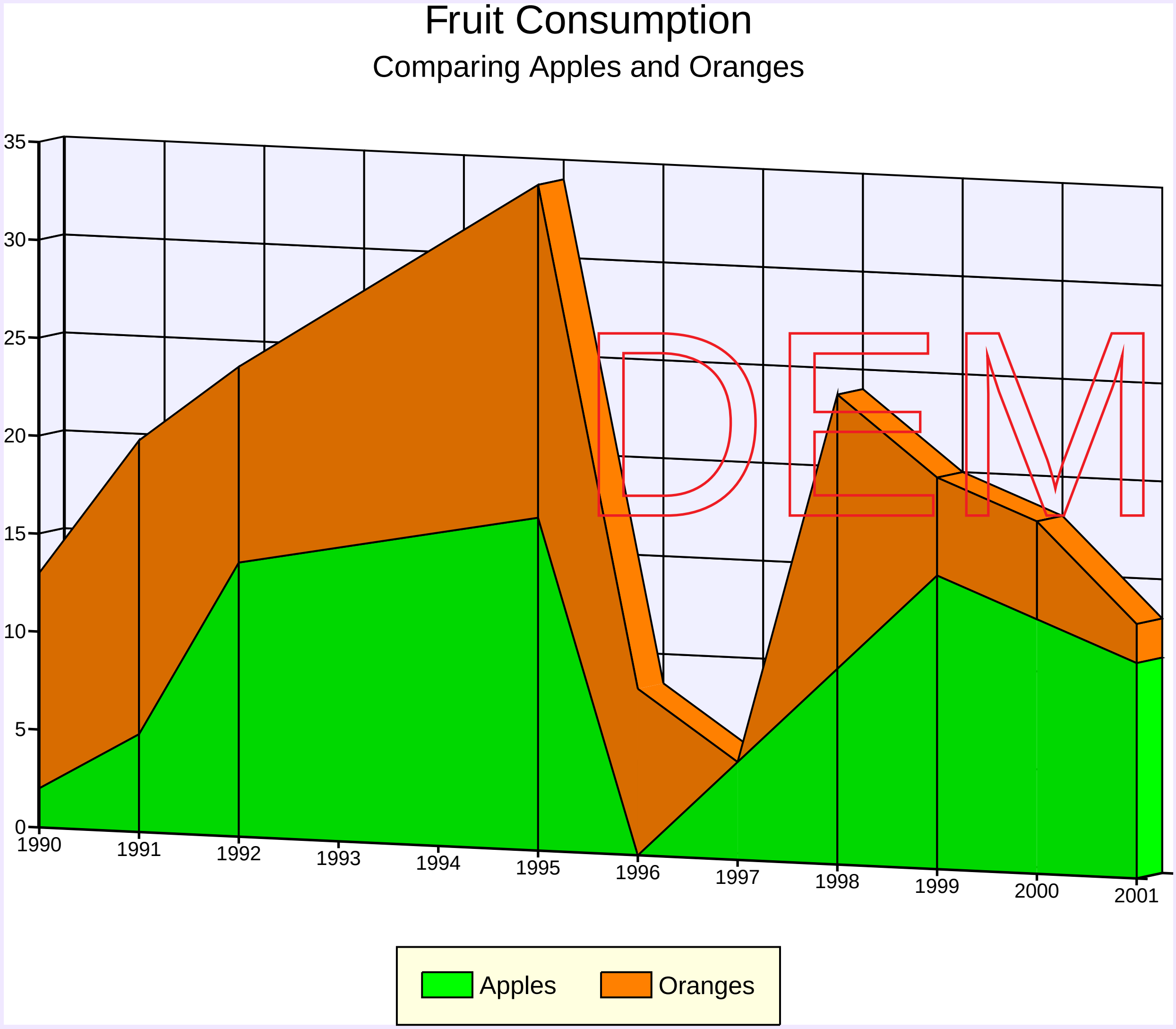


A Plain Area Graph

This is the default layout for an Area Graph. Area Graphs are a variation on a line graph, and are usually used to show cumulative data

The data shown to the left is missing readings for a couple of years - 1993,1994 and 2000 (apples only). The graph will interpolate the data for these results - to set a value to zero, do it specifically (like we've done for apples in 1996 and oranges in 1997).

```
new AreaGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Comparing Apples and Oranges")
```



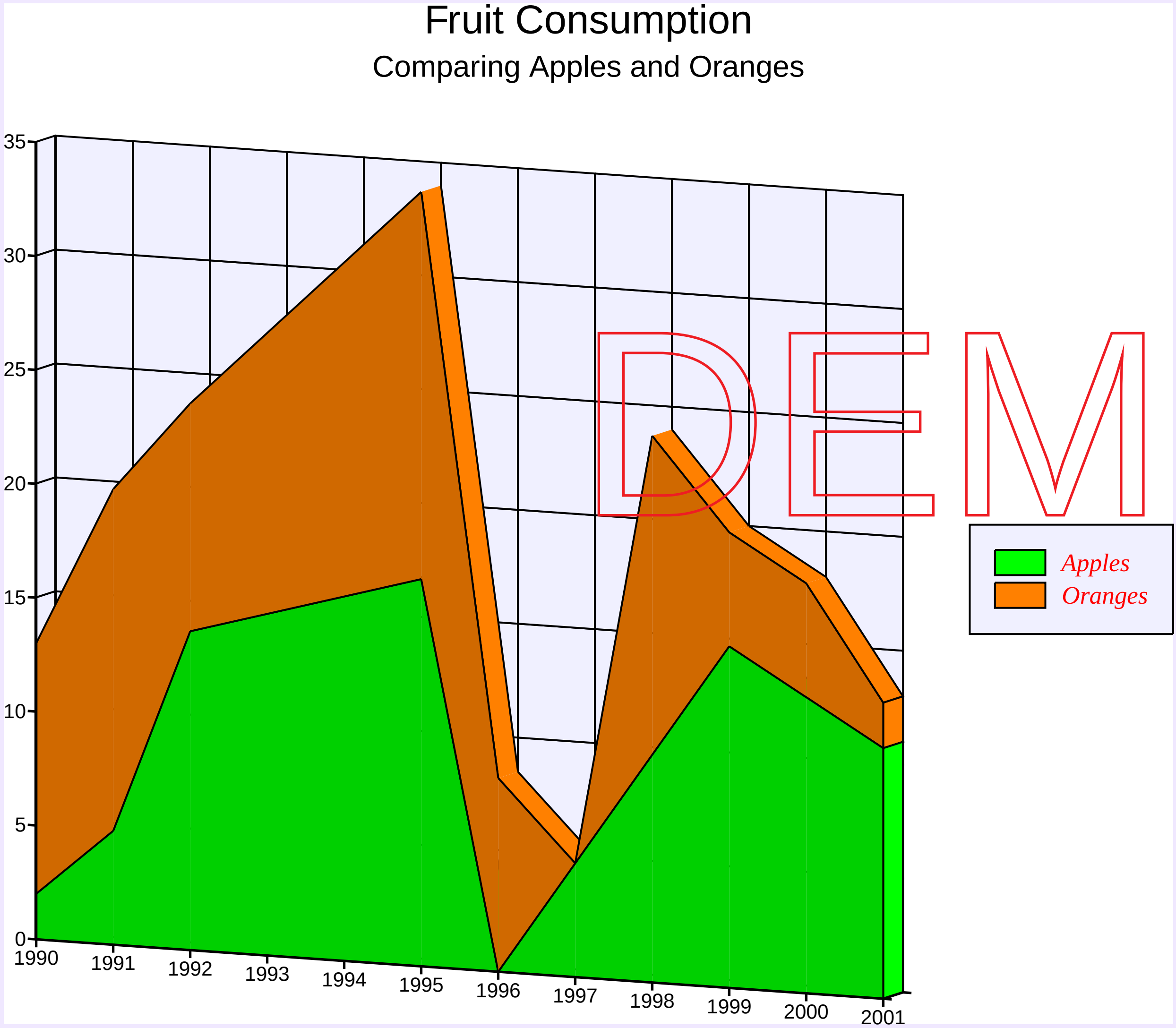
An Area Graph with depth

We've applied X and Y rotation to this graph to give it some depth - the X rotation means we can see the right of the graph, the Y rotation means we can see the top.

We've also added walls at the back (the Z-Wall) and on the Y axis (the Y-Wall), in light blue with a black border.

Notice the colors are darker than the previous graph. This is because of the lighting - by default, the light comes from the right of the graph. A single light source can come from any direction, and although the lighting model is extremely simple it's a nice way to create more realisitic looking graphs.

```
new AreaGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Comparing Apples and Oranges")
optionZWallStyle(new Style(#F0F0FF, #000000))
optionYWallStyle(new Style(#F0F0FF, #000000))
optionYRotation(25.0)
optionXRotation(10.0)
```

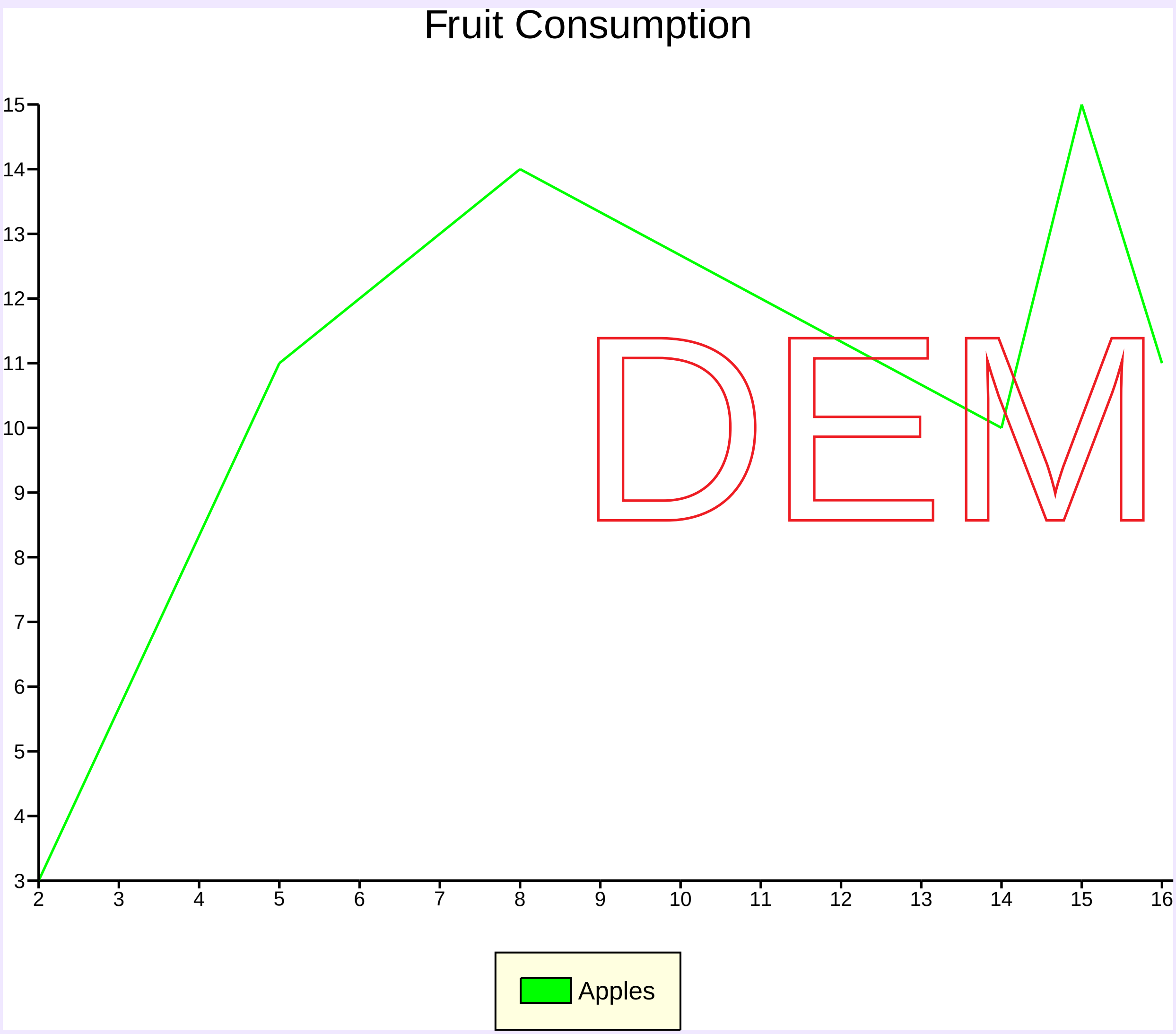


Moving to a different Key

Here's an example where the key is placed to the right of the graph. The key can go above, below, to the left or to the right of any Graph, and for Pie Graphs there are a several more options to choose from. We've also changed the font and color of the Key.

Area Graphs don't have many options specific to them, but one of them is whether to draw "Segments". Here we've turned segments off, so there are no black lines across the surface of the area graph. Compare with the previous example to see the difference.

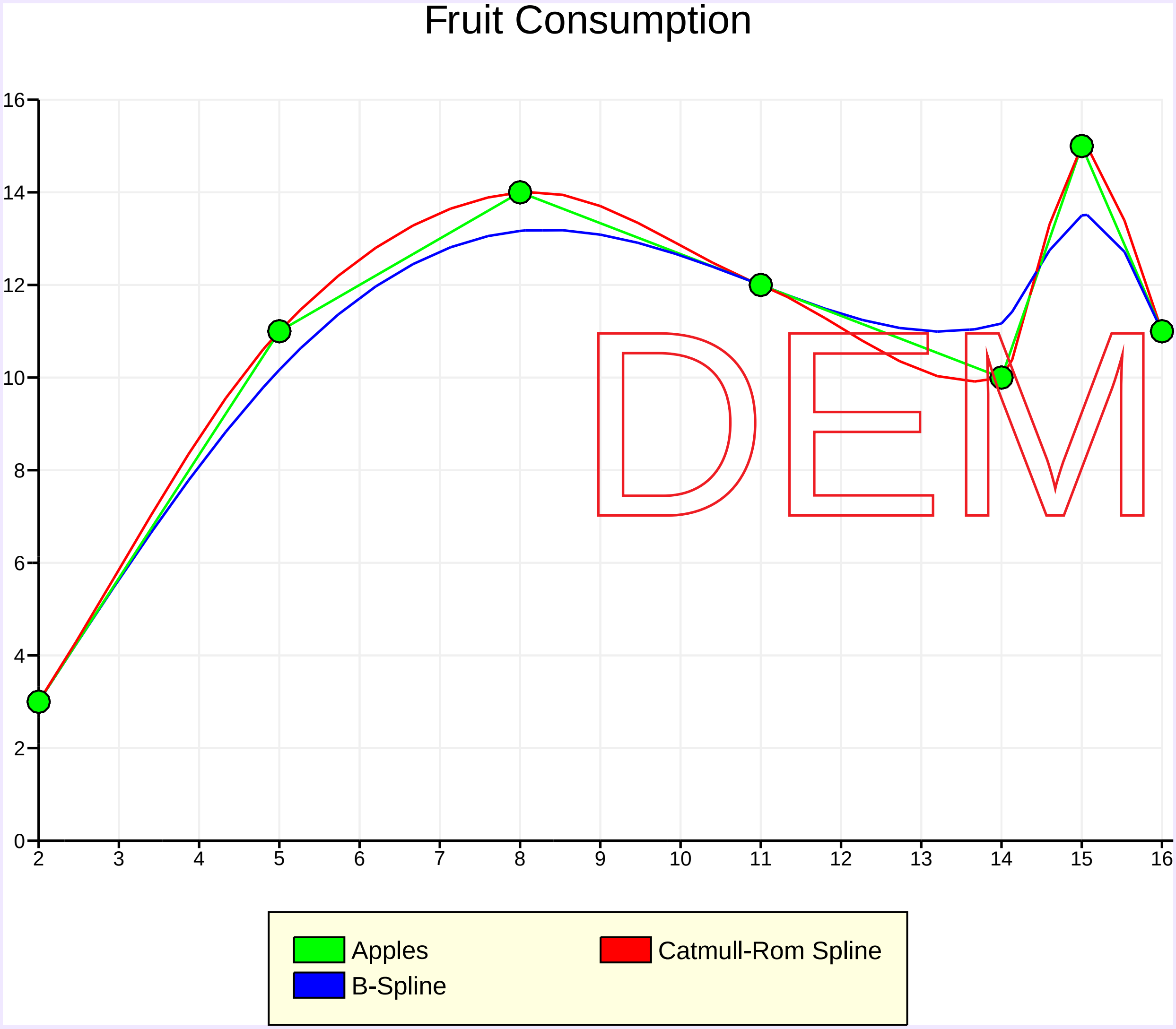
```
new AreaGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Comparing Apples and Oranges")
optionZWallStyle(new Style(#F0F0FF, #000000))
optionYWallStyle(new Style(#F0F0FF, #000000))
optionYRotation(25.0)
optionSegments(false)
optionXRotation(10.0)
optionKeyBoxStyle(new Style(#F0F0FF))
optionKeyStyle(new Style(red, serifItalic10pt))
optionDisplayKey(KEY_BOXED_RIGHT)
```



The Basic Line Graph

This is the basic Line Graph. We've plotted a single curve, which is described by a set of Data Points. Most Line Graphs will have one or more "Data Curves" plotted.

```
new LineGraph()
optionTitle("Fruit Consumption")
```



Smoother Curves

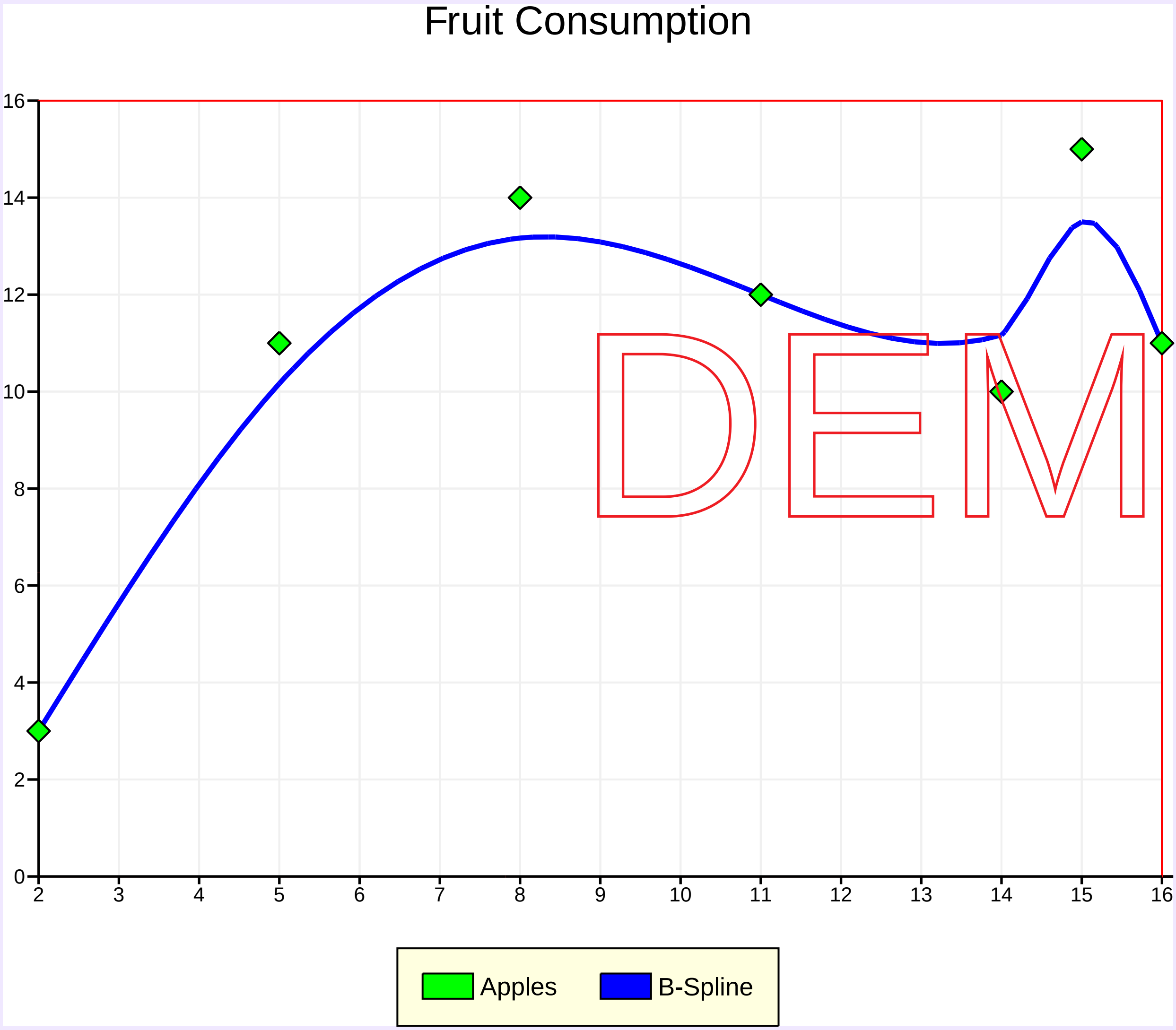
This is the same graph as before, but we're plotting three curves on the graph - the original version, and two "smoothed" versions.

Mapping a collection of points to a function is a complicated business, so we've included three different options for smoothing curves - a Polynomial (not shown here), a Catmull-Rom Spline and a B-Spline. Each has it's benefits, but the splines are probably the most useful (an explanation why is given in the class documentation). If you want to use a different smoothing function, it's easy to write your own subclass of Curve.

We've also set the YStretchToZero flag. Although the data is the same as the previous graph, the Y axis now runs to zero even though the lowest value plotted is 3.

You'll also notice there are markers on the 'Apples' curve. Markers can be Octagonal (like these), Square or Diamonds

```
new LineGraph()
apples.setMarker(MARKER_CIRCLE)
optionTitle("Fruit Consumption")
optionZWallStyle(new Style(null, #F0F0F0))
optionYStretchToZero(true)
```



No Curves at all

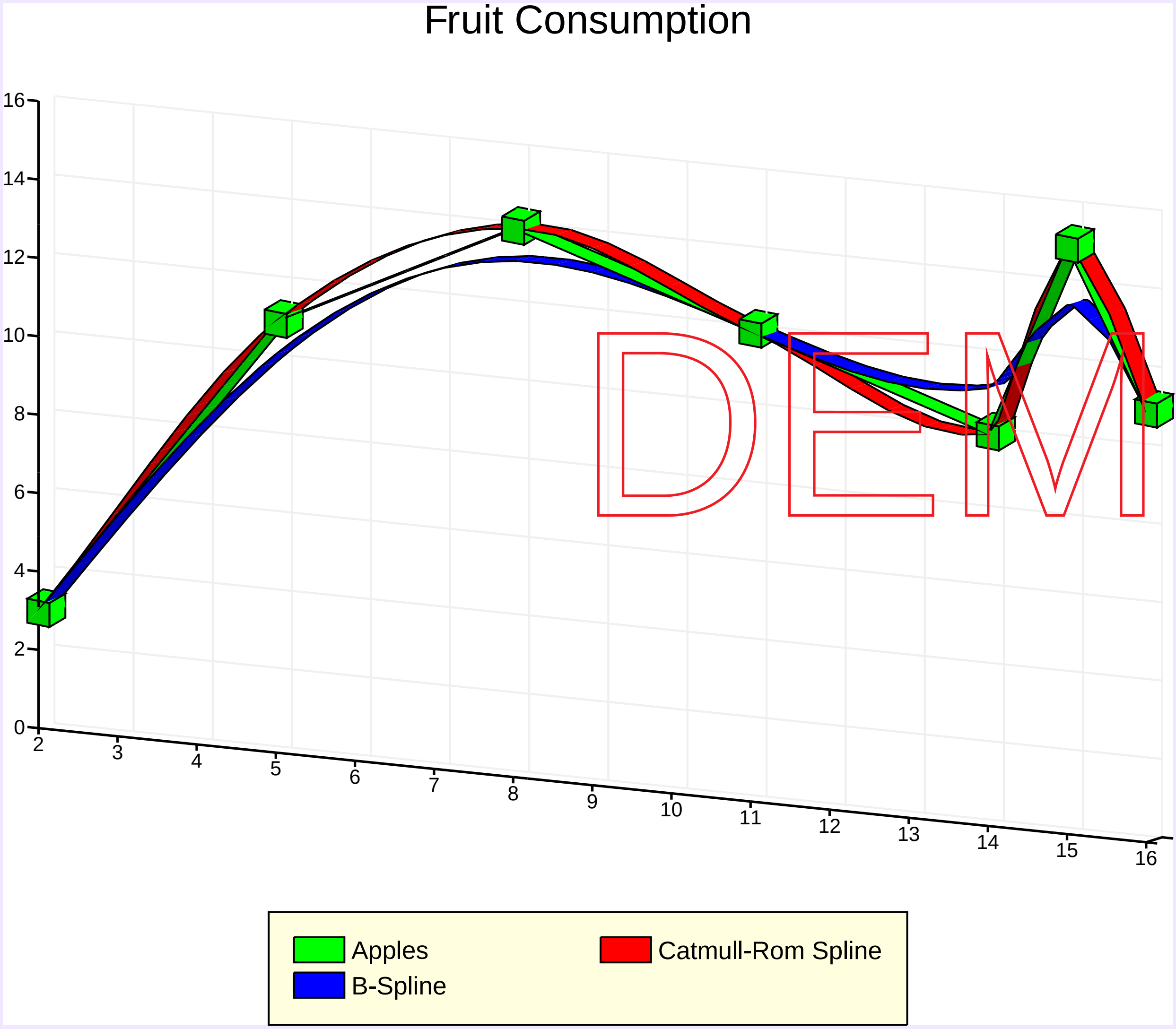
This is the same graph as before, but we're only plotting two curves - the raw data and the B-Spline.

"Two curves?" I hear you say, "I only see one!". We've turned off the lines for the raw data by using the MARKERS_ONLY type of marker. This way we just plot the markers at each point on the curve, and effectively render the line invisible.

We've also upped the FunctionSmoothness from the default of 20 to 50. This is the number of line segments used to plot any Function Curves - higher values give smoother curves but slower graphs.

The red box around the graph is the result of the BoxColor option, and the thicker line is naturally the LineThickness option. This last only has an effect if the graph isn't rotated - on high-resolution devices, the standard line thickess sometimes isn't enough.

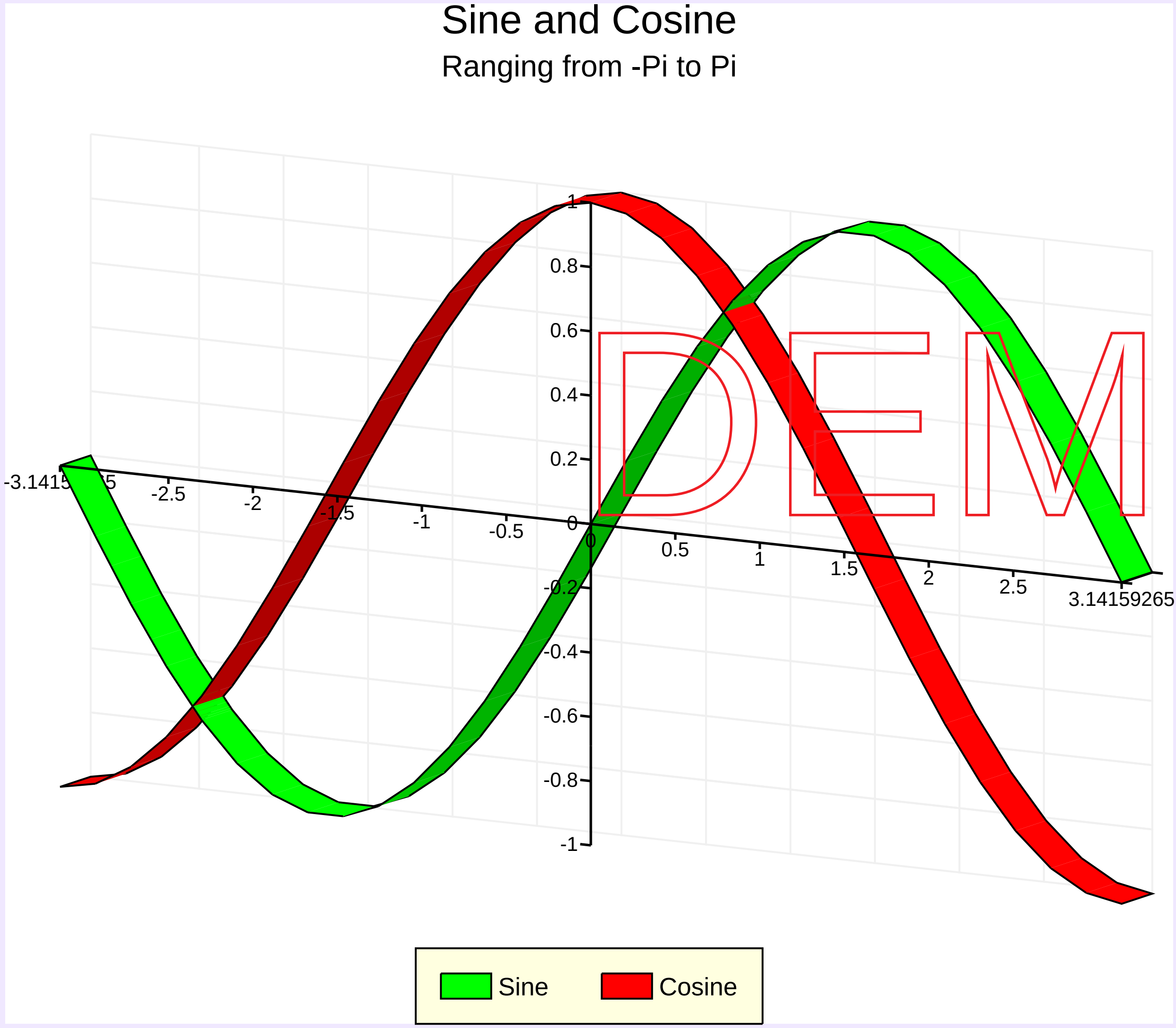
```
new LineGraph()
apples.setMarker(MARKER_DIAMOND|MARKERS_ONLY)
optionTitle("Fruit Consumption")
optionZWallStyle(new Style(null, #F0F0F0))
optionYStretchToZero(true)
optionFunctionSmoothness(50.0)
optionBoxColor(Color.red)
optionLineThickness(2.0)
```



Smoother, Deeper Curves

This graph is identical to the one before, but in 3D. This example is too complicated to be useful, but it still looks nice. We've made the lines a bit thinner to try and ease the congestion on this page

```
new LineGraph()
apples.setMarker(MARKER_SQUARE)
optionTitle("Fruit Consumption")
optionZWallStyle(new Style(null, #F0F0F0))
optionYStretchToZero(true)
optionYRotation(30.0)
optionXRotation(20.0)
optionCurveDepth(0.5)
```



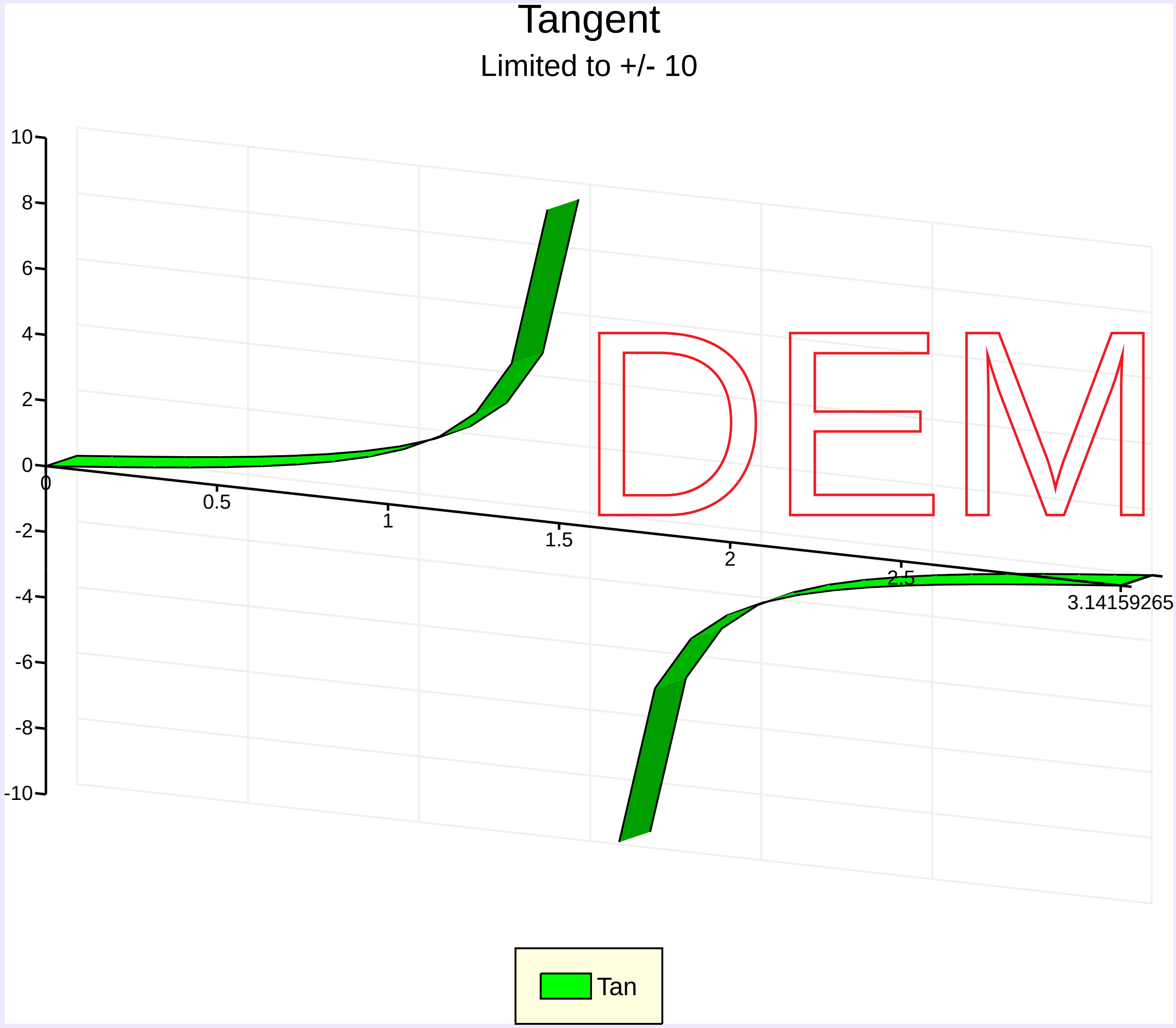
Simpler Curves

If you don't want to go to the effort of subclassing Curve, it's easy to plot simple functions on the graph using the "SimpleCurve" class. Here we're plotting A Sine and Cosine curve. Any function that takes a double and returns a double can be plotted like this, even functions returning infinite or NaN values, like Tangents.

Because we're plotting only Function Curves, we need to specify the start and end points of the graph using the MinX and MaxX options.

Incidentally, the code for a similar example to this is in the "examples" directory, but it changes the X-axis formatter to plot values of Pi on the axis (rather than the standard decimal values you see here).

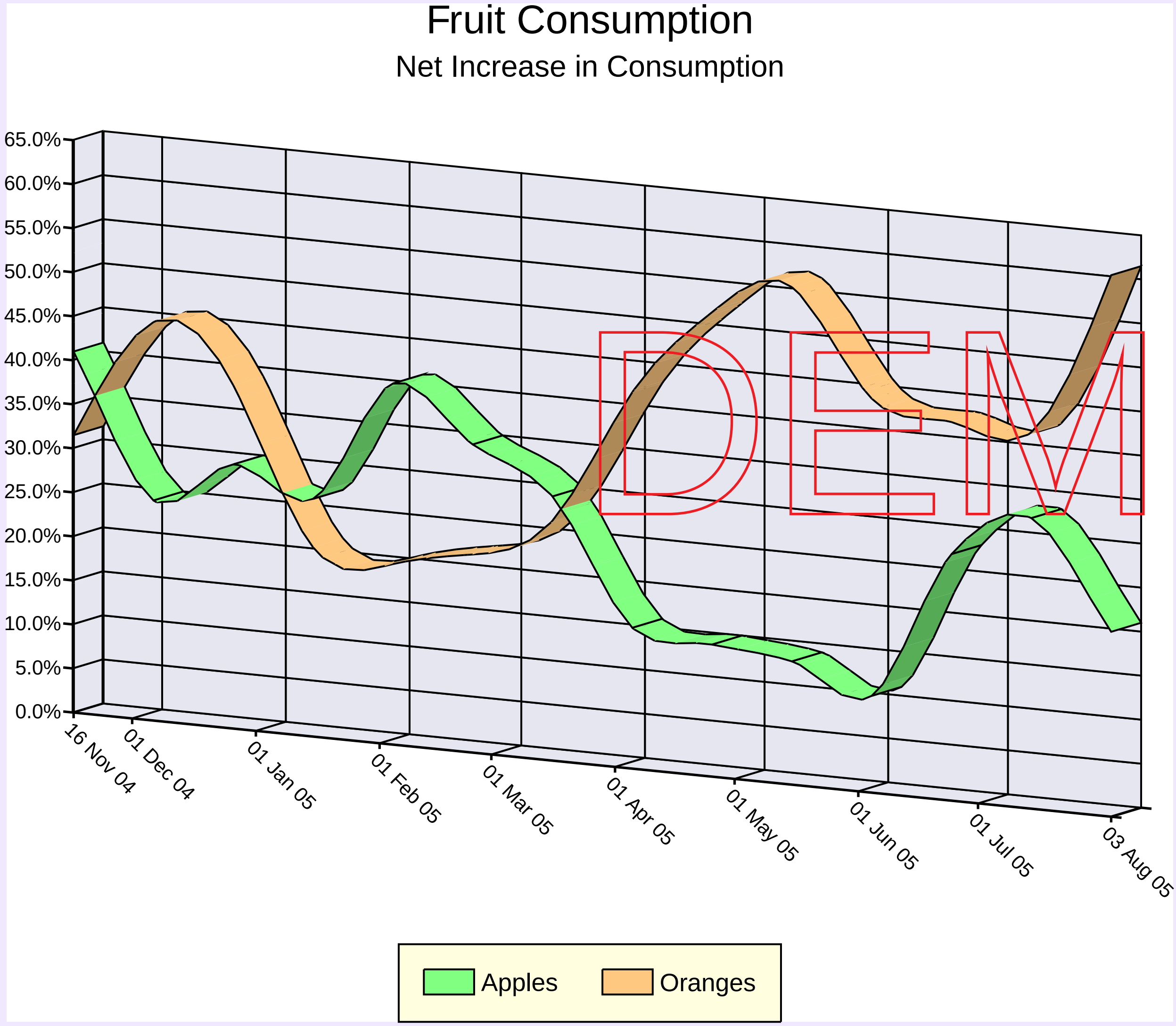
```
new LineGraph()
optionTitle("Sine and Cosine")
optionSubTitle("Ranging from -Pi to Pi")
optionZWallStyle(new Style(null, #F0F0F0))
optionYRotation(30.0)
optionXRotation(20.0)
optionMinX(-3.141592653589793)
optionMaxX(3.141592653589793)
```



Difficult Curves

Some functions return unplottable values for certain positions - the classic example is the Tangent, which returns infinite values for Pi/2. The library neatly handles these values by leaving a gap in the line. To graph a useful range for these functions, you can use the MinY and MaxY options to restrict which parts of the graph to plot

```
new LineGraph()
optionTitle("Tangent")
optionSubTitle("Limited to +/- 10")
optionZWallStyle(new Style(null, #F0F0F0))
optionYRotation(30.0)
optionXRotation(20.0)
optionMinX(0.0)
optionMaxX(3.141592653589793)
optionMinY(-10.0)
optionMaxY(10.0)
```



Format those axes

Here's a line Graph with a difference: we're using some formatters to plot percentages by date. The samples are every 20 days, but the graph makes an effort to plot "useful" values on the axis - depending on the scale of the data, anything from every day to the first of every year (in this case, it's chosen the first of each month).

The Formatter concept gives a great deal of flexibility to the graphs. Here we're plotting the first day of each month, even though there are 30 days in some months and 31 in others - points on the axis don't have to be evenly spaced.

Formatters can be used on any Axes Graph. Pre-defined formatters are available for formatting integers, floating-point, dates, percentages, currency values and discrete values, like those used on a Bar Graph. If this doesn't cover it, the Formatter class can be extended to plot just about anything.

```
new LineGraph()
optionTitle("Fruit Consumption")
optionSubTitle("Net Increase in Consumption")
optionZWallStyle(new Style(#E6E6F0, #000000))
optionYWallStyle(new Style(#E6E6F0, #000000))
optionFloorStyle(new Style(#E6E6F0, #000000))
optionYStretchToZero(true)
optionYRotation(30.0)
optionXAxisTextRotation(45.0)
optionXRotation(20.0)
optionFunctionSmoothness(50.0)
optionYFormatter(new PercentageFormatter(1))
optionXFormatter(new DateFormatter())
```