Render dynamic graphs in SVG
Create SVG graphs that scale dynamically with their content

Brian Venn

October 29, 2004

Scalable Vector Graphics (SVG) is an XML-based language for drawing two-dimensional graphics. The ability to render graphics on the fly lends itself naturally to using it for representing data such as graphs. But suppose the data being represented varies in its magnitude. You may want to graph values between 0 and 10 today, and between 0 and 100,000 tomorrow. Plotting these values on the same scales would be useless -- ideally, you want the ability to scale SVG graphs depending on their content. Author Brian Venn shows you how.

This is the exact problem I came across during testing and development here at IBM Hursley UK. I had already written code that could draw bar charts, line graphs, and scatter plot graphs in SVG very nicely. But the data I was dealing with in each test run varied across several orders of magnitude. Using the same scale on each graph was useless as many graphs were either too small or too big, and the values were either all crunched down in one corner, or plotted off the page.

What I needed was a means of automatically determining the best scale to use for each set of data.

Note: To view the SVG files in this article, you need an SVG viewer which you can find (along with a .zip file that includes all associated files) in Resources.

A little bit of high school mathematics

Looking at the problem, I found that the best means of automatically scaling the graphs was to look at the set of data that needed plotting, determine the largest value contained in the data set, and use that value as the scale against which all the other values would be plotted.

My technique is illustrated better with a working example. Suppose you have a test that generates three values: A, B, and C. Following each test run you need to plot these values onto a bar chart, but the range of values changes dramatically with each test run.
From the first test run, the values generated were:

- A=100
- B=50
- C=25

So for this test run, A had the highest reading with 100. The first (and most important) task is to calculate the **scaling factor** for the graph. This is simply how many pixels in height each value will be worth when it is plotted. The scaling factor is obtained by dividing the height of the graph’s axis by the highest value in the data. This will become clearer when I work through the mathematics for this example, hence.

- For clarity the Y-axis is always set to 1000 pixels high. This simply makes it easier to comprehend the maths involved.
- The highest value in the data set is 100 (for A).
- For this graph. The scaling factor will be: \( \frac{1000}{100} = 10 \).
- So for this bar chart, Each value is worth 10 pixels in height.
- To get the height of each bar, multiply the scaling factor by the data set value. This means the height of each bar will be:

  **Figure 1. Using the scaling factor, highest value is 100 (for A)**

\[
A = 100 \times 10 = 1000 \text{ pixels} \\
B = 50 \times 10 = 500 \text{ pixels} \\
C = 25 \times 10 = 250 \text{ pixels}
\]

Now, suppose on the next test run the following values are generated:

- A=2000
- B=5000
- C=800

Remember, the Y-axis is still 1000 pixels high. The highest value this time is the B column, with a value of 5000. The scaling factor is therefore \( \frac{1000}{5000}=0.2 \).

So this time the height of each bar will be:
Figure 2. Using the scaling factor, highest value is 5000 (for B)

A=2000x0.2 = 400 pixels
B=5000x0.2 = 1000 pixels
C =800x0.2 = 200 pixels

As you can see, because the heights of the bars are calculated with respect to the highest value, the bars do not exceed 1000 pixels in height and do not end up being plotted too high. It is this scaling factor that is the key to SVG dynamic scaling. Remember, it is calculated based on the highest value; all other values are scaled accordingly.

Now that I have introduced the key concept of the scaling factor, I'll show you how to put it to use. The following example implementation is a small Java-language program that takes a set of values on the command line and draws an SVG bar graph in the fashion described in the above example. Although written in Java code, it is the mathematics that is the important factor. If necessary, you should be able to rewrite it in any language you wish by simply looking at the Java code.

Rather than reproduce the whole code here in the article, I'll just cover the main points. I recommend that you download the examples (see Resources) and open up the SVG_barchart.java file in an editor with a line numbering facility.

First the highest value for the passed in values is determined in Lines 38-48. The most important line is line 51, where the Y-axis scaling factor is calculated:

Listing 1. Calculate the Y-axis scaling factor

51. YAxisScalingFactor = 1000/(double)largestNumber;

The SVG output file is created on lines 58 and 59. The SVG is written out to this file using SVGout.write(-SVG-). Note that various escape characters must be used such as \" for double quotes. This can make the SVG difficult to read at times, but is necessary as otherwise the Java compiler interprets it incorrectly and the code won't compile. Also, be aware that each SVG text line starts with \n -- this makes the SVG easier to read when using the View Source function of an SVG viewer.

Lines 70-79 draw the X and Y axes. The X-axis is a simple line from [x=0, y=1000] to [x=1000, y=1000], and the Y-axis runs from [x=0, y=0] to [x=0, y=1000]. This provides a simple Cartesian set of axes to work with.
Lines 82 and 83 draw a dotted line at the top and halfway down the Y-axis. This simply makes the graph easier to read.

On lines 86 and 87, the Y-axis is labeled at the top, and halfway down. Notice that you are using the largest number value calculated earlier to label the Y-axis.

**Listing 2. Label the Y-axis**

86. SVGout.write("<text style="fill:black; stroke:none"
  x="-10" y="0" >" + largestNumber + "</text>);
87. SVGout.write("<text style="fill:black; stroke:none"
  x="-10" y="500" >" + (largestNumber/2) + "</text>");

Note that the values are dynamic, as this uses the `largestNumber` value to label the axis. If necessary, you can easily add extra guidelines and labels by dividing by other values.

The actual drawing of the bars starts on line 90:

**Listing 3. Draw the bars**

90. // The graph is ready to be rendered with the values.
91. for(i=0;i<barChartValues.length;i++) {
  92.  // Calculate the Y position. First work out how high the bar
  93.  // will be by multiplying the value by the scaling factor.
  94.  // calculated earlier
  95.  double barHeight =
  96.    Integer.parseInt(barChartValues[i]) * YAxisScalingFactor;
  97.  System.out.println("Bar Height is =" + barHeight);
  98.  double YStart = 1000 - barHeight;
  99.  double XPosition = (i*110);
100.  double YPosition = (i*10); // Generate some random numbers for your bar colours
101.  int randomRed = random.nextInt(255);
102.  int randomGreen = random.nextInt(255);
103.  int randomBlue = random.nextInt(255);
104.  SVGout.write("<rect x=""+XPosition+"" y=""+YStart+"" width =""+100+"" height=""+barHeight+"" style="fill:rgb("+randomRed+""
  +randomGreen+","+randomBlue+"));
105. }
106. }
Try downloading and running the grapher yourself. Pass in various values ranging across different orders of magnitude and see how the graph changes and scales itself with the values. Figure 3 shows some example output.

**Figure 3. Sample dynamically generated SVG bar graphs**

![Sample dynamically generated SVG bar graphs](image)

Line graphs  
So far, I've shown you how to dynamically scale bar charts. But these aren't the only types of graphs; some forms of data, especially time-based data such as share prices or seismographic data are best plotted on line graphs. You can also scale line graphs with a similar approach to that used with a bar graph.

Suppose you are monitoring the number of rows in a database. A reading of the number of rows is taken once every 30 seconds and is recorded in an array. At the end of the test run you have an array with 10 values:

10, 20, 30, 50, 90, 25, 45, 60, 70, 10

Once again, you need to calculate a scaling factor for each test run. However, you don't want to restrict yourself so that you are only able to draw graphs if you have an exact number of readings. This time you are going to have to scale on both the X-axis and the Y-axis.

The best way to draw a line graph is to use an SVG element called the **polyline**. The polyline takes in pairs of values as X and Y points and draws a connecting line between the points. For example:

```xml
<polyline points="0,0, 10,10, 20,20"/>
```

draws a line with 3 points at [X=0,Y=0], [X=10,Y=10], and [X=20,Y=20]. These are the points that you need to scale and calculate from the data set.

As before, a working Java example program is supplied that scales and renders polyline graphs from the supplied data (see Related topics).

Again, I'll just cover the main points of the code.
First, calculate the X-axis scaling factor. This is determined by the number of readings you are rendering, so you need to divide the number of pixels in the X-axis by the number of readings. For example, if you have taken 50 readings, the X-axis scaling factor would be 1000/50 = 20. Hence each reading is plotted at 20-pixel intervals along the X-axis. As the readings are passed in as an array, you simply need to divide 1000 by the number of elements in the array.

**Listing 4. Calculate the X-axis scaling factor**

```java
40. XAxisScalingFactor = 1000/(double)valuesToPlot.length;
```

Lines 41-49 find the highest value in the passed-in array.

Lines 54-100 get the output file ready and draw the X and Y axes so they're ready for plotting.

The Y-axis scaling factor is calculated on line 104 in the exact same way as in the bar chart example (see **Listing 2**).

Lines 108 to 125 are where the line gets rendered:

**Listing 5. Draw the polyline**

```java
108. // Render the line
109. SVGout.write("\n<polyline points="0 1000,");
110.
111. for(i=0;i<valuesToPlot.length;i++){ 
112. 
113.  // Calculate the X position by determining which
114.  //value in the array you are dealing with.
115.  XValue = ((i+1)*XAxisScalingFactor);
116. 
117.  YValue = Integer.parseInt(valuesToPlot[i]);
118.  YValue = YValue*YAxisScalingFactor;
119.  YValue = 1000-YValue;
120. 
121.  // You now have your polyline point.
122.  SVGout.write( " " + XValue + " " + YValue +",\n"); 
123. 
124. }
125. // Close off the polyline.
126. SVGout.write("\" style="stroke:red; stroke-width: 3; fill :
127. none;"/>");
```

In line 109, the polyline element is started, and an initial point is drawn at the origin of the axis.

Next, start to work your way through the values in the array. First, calculate the X value. This is the position of the element in the array, multiplied by the X-axis scaling factor. For example, if you have 50 elements in your array, the X-axis scaling factor would be:

- \( \text{XAxisScalingFactor} = \frac{1000}{\text{valuesToPlot.length}} \)
- \( \text{XAxisScalingFactor} = \frac{1000}{50} \)
- \( \text{XAxisScalingFactor} = 20 \)

So each value is plotted at 20-pixel intervals, hence working through the array:
• 1st point : x=20,<1st value>
• 2nd point : x=40,<2nd value>
• 3rd point : x=60,<3rd value>
• ...
• ...
• 49th point : x=980,<49th value>
• 50th point : x=1000,<50th value>

Note that you use \((i+1)\) because in the array, the counter value of the first element is zero; as you want to start with the first element, the 1 is added.

Next, the Y-value is calculated:

1. Line 117 extracts the value from the array and converts to an integer.
2. Line 118 multiplies it by the Y-axis scaling factor to determine its position.
3. Finally, subtract it from 1000. This is done because the Y-values run positively down the page.

The X-axis is drawn horizontally at a Y height of 1000 pixels, so to determine how high this calculated Y position is above the X-axis, you subtract it from 1000.

You now have your calculated X and Y positions, so add this point into the polyline inline.

Once you have worked through all the values in the array, the polyline is closed off and a style is applied to the line.

Try running the example and passing in various lists of numbers across different orders of magnitude. Again, you will see how the graph scales itself depending on the highest value that is passed in. Also, try passing in different numbers of values, such as 10 or even 1000. The X-axis will scale itself based on the number of values passed in. Figure 4 shows some example output.

**Figure 4. Sample dynamically generated SVG line graphs**

![](image)

**Scatter plots**

The final type of graph covered here is the scatter plot graph. These graphs are good for data that varies across two dimensions. Here, you'll use the exact same approach as in the previous examples, although this time you have a set of values for the X and Y axes to plot.
In the example, the data passes in the form (X-value), (Y-value). For example, [1,1], [3,5], [50,2], and [10,34].

Once again, I provide a Java sample program that you can download (see Related topics). Here's a look at the code.

Lines 31-57 determine the highest values for the X values and the Y values.

Lines 60-61 calculate the scaling factor for both the X and the Y axes.

Lines 67-119 draw the axes. Note that this time you are drawing four dotted guide lines across the graph and labeling them accordingly.

Lines 124-157 plot the data:

**Listing 6. Calculate and draw the scatter plot points**

122. // The axis and the guide lines are ready; now draw the data.
123. SVGout.write("\n    <g style="fill:none; stroke:red; stroke-width:3">\n124. for(i=0;i<dataToPlot.length;i++){\n125.   // Get the value out of the array.
126.   String value = dataToPlot[i];
127.   index = value.indexOf (',');
128.   String X_Pos = value.substring(0,index);
129.   String Y_Pos = value.substring(index+1);
130.   XValue = Integer.parseInt(X_Pos);
131.   YValue = Integer.parseInt(Y_Pos);
132.   // Calculate the point's position by using the scaling
133.   XValue = XValue*XAxisScalingFactor;
134.   YValue = YValue*YAxisScalingFactor;
135.   YValue = 1000-YValue;
136.   // You now have your point. As it's a scatter plot, it
137.   // would look nice with an X, so use the point to draw
138.   // a line from the top left to the bottom right, and from
139.   // the top right to the bottom left.
140.   \n141.   SVGout.write("line x1="+(int)(XValue-5) +
142.    " y1="+(int)(YValue+5)+ " x2="+(int)(XValue+5)+
143.    " y2="+(int)(YValue-5)+" />";)
144. }

First, define a style for this group (line 123).

Then, begin working through the data. In line 129, you obtain the value.
Next, separate the X and Y values out by looking for the comma separator (lines 133-135) and convert the values to numbers (lines 138-139).

Then calculate the position of the point (lines 143 - 145).

The point is now ready to plot. Instead of just plotting a single dot, this time try marking the spot with an X. This is a simple case of drawing two lines, one starting 5 pixels to the left and 5 pixels up and ending 5 pixels to the right and 5 pixels down, and another starting 5 pixels to the right and 5 up and ending 5 pixels to the left and 5 pixels down. This will result in an X, centred on the calculated point.

Download and run the scatter plot grapher contained in the examples.

**Figure 5. Sample dynamically generated scatter plot graphs**

Further improvements

For simplicity and to make the code as readable as possible, I have used a minimal amount of Java code and generated SVG in the examples shown here. You can, of course, take these examples and improve on them further by adding extra colours, effects, and information. Included in the samples are two bar chart graphers, one of them draws the simple rectangles as described earlier (see Figure 3). The other is slightly more complicated and uses gradients and drop shadows, but the method of calculating the bar heights remains the same. Figure 6 shows an example.
You can see an example of one of my own line graphs in Figure 7. This graph shows the number of messages that were in the database I was running a test on, plotted over time. I added in a logo, a drop shadow effect, and a stats table of the test run.
Figure 7. Improved SVG line graph

Figure 8 shows an example of one of my scatter plots. During this test run, I wanted to compare the performance times for the deleting of messages from a database. One set of data was based on using JDBC for the operation, the other on using prepared statements. Again, this has a logo, drop shadows, and a stats table.
Figure 8. Improved SVG scatter plot

The full SVG versions of both these graphs are included with the samples.

Summary

This article demonstrated by example how to dynamically scale SVG graphs according to the data. Through these techniques and with the sample code, you can now render your own graphs and tailor them to your own requirements.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code and SVG example files</td>
<td>x-svggrph_examples.zip</td>
<td>51 KB</td>
</tr>
</tbody>
</table>
Related topics

- **Download** the sample code used in this article.
- Find the **Scalable Vector Graphics (SVG) specification** at the W3C site.
- Read Brian Venn's other developerWorks articles, "**Bring Scalable Vector Graphics to life with built-in animation elements**" (June 2003).
- Check out Adobe's **SVG Zone** for a wide range of SVG resources, including an area where you can download an **SVG viewer**.