Paper 282-2011

Can SAS® Do Canvas? Creating Graphs Using HTML 5 Canvas Elements

Edwin J. van Stein, Astellas Pharma Global Development Europe, The Netherlands

ABSTRACT

With the introduction of the canvas element in the HTML 5 specifications, an interesting alternative for creating nice-looking graphs that are compatible with most Web browsers has become available. The purpose of this paper is to provide an introduction to the canvas element and how to let SAS® draw on it. The examples that are given have been tested in SAS® on Microsoft Windows and UNIX with different Web browsers on Windows, Linux, and Mac OS X as client.

INTRODUCTION

The canvas element is part of HTML5 and allows for dynamic, scriptable rendering of 2D shapes and bitmap images (to quote Wikipedia). Canvas was first introduced in Apple's Max OS X WebKit component (Safari) and was later added to the Gecko browsers (Mozilla Firefox amongst others) and Opera. The specifications, as part of the HTML 5 specifications, are currently being maintained by WHATWG (Web Hypertext Application Technology Working Group, founded by representatives from Apple, the Mozilla Foundation and Opera Software).

COMPATIBILITY

The canvas element is natively supported by the latest versions of Mozilla Firefox, Google Chrome, Safari and Opera. It is not natively supported in Internet Explorer (support is in development for Internet Explorer 9). However, including a JavaScript (JS) library like ExplorerCanvas in your HTML adds support in Internet Explorer without installing a plug-in on the client machine. Support for the canvas element can also be added to Internet Explorer using Google or Mozilla plug-ins, but this requires installation of plug-ins, which a lot of companies do not allow their staff to do.

CANVAS VERSUS SVG

An obvious competitor to the canvas element is SVG (Scalable Vector Graphics). SVG is an open standard for static and dynamic vector graphics under development by the W3C (World Wide Web Consortium) and has been around for quite some time now. In SAS® 9.2 an SVG graph device has been added to allow the creation of SVG graphs directly from SAS®. Admittedly SVG has a number of advantages over canvas. For instance SVG is a vector format whereas canvas creates a bitmap from the instructions received, SVG has a scene graph which is very useful for re-rendering, SVG consists of objects you can add events to and SVG is resolution independent. However, SVG is not natively supported by Internet Explorer (support is in development for Internet Explorer 9). Support can be added to Internet Explorer using a plug-in, which needs to be installed on the client machine. Note that support for the SVG Viewer by Adobe has been discontinued on the 1st of January 2009. Support in other browsers is incomplete in most cases and Mozilla Firefox does not support embedding via an <img> element. In addition I have found the code to create canvas graphs from scratch a lot easier then the code behind SVG if you stay away from animations.

SAS® AND CANVAS

The code that drives canvas graphs is very similar to the functions used when annotating graphs. This paper will first give an introduction on the canvas element and then some examples on how to draw on a canvas from a DATA step to produce graphs.

SETTING UP THE CANVAS

The canvas itself has no contents or border. In its most basic form a canvas has an id, a width and a height. Using the id the canvas can be identified and contents can be added using JS. In most cases the graph needs to be displayed when the page is loaded. This can be done by adding a script to the header of the HTML output and calling that script when the body of the HTML output is loaded. With Internet Explorer still being the most used web browser it's a good idea to include a JS framework like ExplorerCanvas so that the output is displayed correctly on all major browsers. This results in the following basic set-up of the output.

```
<html>
<head>
```

```
<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->
<script type="text/javascript">
function drawit() {
// JavaScript to draw in the canvas goes here
}
</script>
</head>
<body onLoad="drawit();">
<div id="container">
<canvas id="graph" width="401" height="301"></canvas>
</div>
</body>
</html>
```

In case of multiple graphs each canvas element needs to have a unique id. The resulting HTML output is just a blank page with a blank canvas on which to draw. The coordinate system within the canvas is slightly different from the coordinate system using annotation, because the y dimension is inverted as illustrated below.

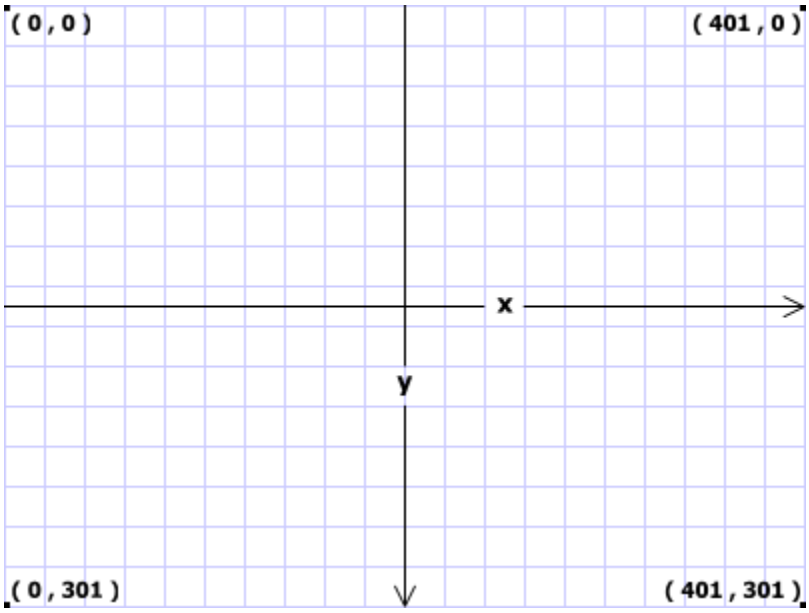

Figure 1 Coordinate system in canvas

To work around the inverted y-axis you can use the %scalet macro provided with SAS® (part of %annomac) or create something similar. For the examples in this paper a %canvas_scale macro is used that converts the value &val from the scale &minval - &maxval to the scale &mincoord - &maxcoord. Macros like these work fine with inverted minimum and maximum coordinates.

Both in the HTML example and the figure you might have noticed that the size of the graph is 401 by 301 pixels instead of a seemingly more logical 400 by 300 pixels. Imagine if you zoom in on the graph far enough so that pixels become squares. The coordinate (0,0) is not the center of the top left pixel, but instead the top left corner of the top left pixel. Your browser can only draw entire pixels, so drawing a line of 1 pixel wide from (1,0) to (1,4) will result in a line of 2 pixels wide. Drawing the same line from (3.5,0) to (3.5,4) will result in a line with a width of 1 pixel as illustrated below.

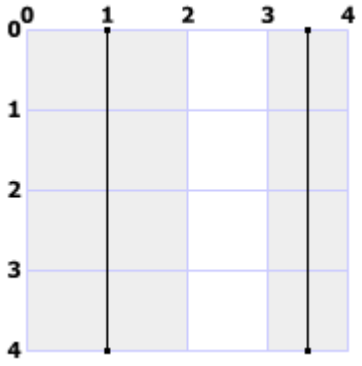

Figure 2 Drawing lines with a width of 1 pixel

For things like axes you're very likely to draw lines at logical locations (for instance x-axis from (10,290) to (390,290) and y-axis from (10,290) to (10,10)). If you want your lines to be 1 pixel wide then it helps to consistently add 0.5 to every coordinate (if you prefer lines with a width of 2 pixels then using integers for coordinates is better).

SOME MORE PREPARATIONS

The JS that is going to draw on the canvas needs to know which canvas to draw on and which context of that canvas. Currently only 2d is available as context (3d will probably be added in a future version of HTML 5 according to WHATWG). To identify the canvas and its context the following lines of code should be the first thing in the script that draws on the canvas:

```
var canvas = document.getElementById("graph");
var context = canvas.getContext("2d");
```

Graph is the id of the canvas element. Any statements after this can use the JS variable context to reference the context instead of having to repeat document.getElementById("graph").getContext("2d") every time. Please note that getElementById will fail if the element (in this case the canvas) does not exist yet. So the JS that draws on the canvas should not run before the canvas is loaded. This is done by adding it to the onLoad event of your output HTML's body tag.

At this point it's also a good idea to set some compositing and style attributes that you are unlikely to set per individual shape to draw. The globalCompositeOperation attribute determines what to do when elements are overlapping. A value of 'source-over' will simply put the last shape to draw on top of the previous one. The value 'lighter' displays both overlapping shapes and the part that overlaps has a lighter color. This attribute is not supported by ExplorerCanvas, so the output in Internet Explorer will always look as if it is set to 'source-over' (the default in the other browsers).

The lineCap attribute determines what to do with the end of a line. Available values are 'butt' (default), 'round' and 'square'. The lineJoin attribute determines what to do when 2 lines from the same shape come together. Available values are 'round', 'bevel' and 'miter' (default). In case of miter a limit can be set with the miterLimit attribute (default value 10) so that the miter does not become too long if the angle between the lines is very small. To set these attributes add the following lines to the JS function:

```
context.globalCompositeOperation="lighter";
context.lineCap="butt";
context.lineJoin="miter";
context.miterLimit=10;
```

CODING IT

As example a simple line plot will be drawn on a canvas which is 801 by 601 pixels. Lines of 1 pixel wide will be drawn so 0.5 will be added to all coordinates. All canvas related drawing will be done using macros to make it reusable. These macros can be found on sasCommunity.org. The basic DATA step we will be expending upon looks like this:

```
data canvas;
   set wins end=eof;
```

```
   file "C:\Personal_Local_Data\wins.html";
   if _n_=1 then do;
      %canvas_init(xpix=800,ypix=600,gco=source-over);
   end;
   if eof then do;
      %canvas_term;
   end;
run;
```

The %canvas_init macro sets up the beginning of the HTML output and the JS. The %canvas_term macro finalizes the HTML output including the canvas and the onLoad event. The input data comes from the State Gaming Control Board in Nevada and contains the statewide total win amount for the games Twenty-One, Craps, Roulette, 3-Card Poker and Baccarat per month for 2010.

DRAWING STRAIGHT LINES

Drawing a straight line on canvas can be done by defining the start of a path with the beginPath() method, moving to a start coordinate (__x1,__y1) using the moveTo() method, drawing the line with the drawTo() method to the end coordinate (__x2,__y2), setting the width and color using the lineWidth and strokeStyle attributes and then actually drawing the line with the stroke() method.

```
put "   context.beginPath();";
put "   context.moveTo(.5+" __x1 +(-1) ",.5+" __y1 +(-1) ");";
put "   context.lineTo(.5+" __x2 +(-1) ",.5+" __y2 +(-1) ");";
put "   context.lineWidth=" __line_width +(-1) ";";
put "   context.strokeStyle=""" __scolor +(-1) """;";
put "   context.stroke();";
```

The coordinates are the coordinates following the system canvas uses. The strokeStyle attribute requires a color that is recognized by a web browser like, ‘white’, ‘fff’ or ‘#ffffff’.

DRAWING LABELS

Writing on canvas can be done by using the strokeText() or fillText() methods. Both of these methods use the following attributes:

- font: needs to be a valid font style like ‘normal 8px sans-serif’;
- textBaseline: the baseline of the text, this can be ‘top’, ‘hanging’, ‘middle’, ‘alphabetic’, ‘ideographic’ or ‘bottom’;
- textAlign: the alignment of the text, this can be ‘left’, ‘center’, ‘right’, ‘start’ or ‘end’.

Text can be filled and/or outlined. FillText() uses the fillStyle attribute to determine the color. The fillStyle attribute requires a color that is recognized by a web browser similar to strokeStyle. StrokeText() uses the strokeStyle and lineWidth attributes similar to drawing lines.

```
put "   context.font=""" __font +(-1) """;";
put "   context.textBaseline=""" __base +(-1) """;";
put "   context.textAlign=""" __align +(-1) """;";
put "   context.lineWidth=" __line_width +(-1) ";";
put "   context.strokeStyle=""" __scolor +(-1) """;";
put "   context.strokeText(""" __text +(-1) """,.5+" __x1 +(-1) "," @;
put ".5+" __y1 +(-1) ");";
put "   context.fillStyle=""" __fcolor +(-1) """;";
put "   context.fillText(""" __text +(-1) """,.5+" __x1 +(-1) "," @;
put ".5+" __y1 +(-1) ");";
```

DRAWING LABELS AT AN ANGLE

Writing at an angle or drawing any other shapes at an angle can be done, but works differently from what you might expect. The easiest way of doing it is:

1. save the current state of the canvas using the save() method, this stores the current origin and rotation of the canvas;

2. move the origin of the canvas (normally the (0,0) coordinate) to the coordinates you want to rotate around using the translate() method;
3. rotate around the new origin using the rotate() method, the rotate() method requires the angle expressed in radians, so if you know the degrees you can convert to radians in JS using '*Math.PI/180';
4. write the text using the strokeText() and/or fillText() methods;
5. restore the state of the canvas using the restore() method, this will ensure that subsequent text and shapes is drawn without rotation.

```
put "   context.save();";
put "   context.translate(.5+" __x1 +(-1) ",.5+" __y1 +(-1) ");";
put "   context.rotate(" __ang +(-1) "*Math.PI/180);";
* add methods and attributes to write/draw;
put "   context.restore();";
```

DRAWING RECTANGLES

Drawing a rectangle (or any other shape with straight lines) in canvas is quite similar to using the POLY and POLYCONT annotate functions. First tell the canvas that a new path should be started using the beginPath() method. Then moveTo() the first coordinate and draw lines to the subsequent coordinates using the lineTo() method. Using closePath() the path will be closed so you don't have to return to the first coordinate using lineTo(). After that you can choose to either fill the shape that was drawn using the fill() method and/or outline the shape using the stroke() method. This once again uses the fillStyle, lineWidth and fillStyle attributes.

```
put "   context.beginPath();";
put "   context.moveTo(.5+" __x1 +(-1) ",.5+" __y1 +(-1) ");";
put "   context.lineTo(.5+" __x1 +(-1) ",.5+" __y2 +(-1) ");";
put "   context.lineTo(.5+" __x2 +(-1) ",.5+" __y2 +(-1) ");";
put "   context.lineTo(.5+" __x2 +(-1) ",.5+" __y1 +(-1) ");";
put "   context.closePath();";
put "   context.lineWidth=" __line_width +(-1) ";";
put "   context.strokeStyle=""" __scolor +(-1) """;";
put "   context.stroke();";
put "   context.fillStyle=""" __fcolor +(-1) """;";
put "   context.fill();";
```

Rectangles can also be drawn using the rect() method, but using the moveTo() and lineTo() methods comes in handy when drawing other symbols with straight lines or combining straight lines with curved lines.

DRAWING SYMBOLS

Similar to drawing rectangles any symbol consisting of straight lines can be drawn using the methods described above. For instance a triangle can be drawn using the code below.

```
put "   context.beginPath();";
put "   context.moveTo(.5+" __x1 +(-1) ",.5+" __y1 +(-1) "-" @;
put __size +(-1) "*Math.sqrt(3)/4);";
put "   context.lineTo(.5+" __x1 +(-1) "+" __size +(-1) "/2,.5+" @;
put __y1 +(-1) "+" __size +(-1) "*Math.sqrt(3)/4);";
put "   context.lineTo(.5+" __x1 +(-1) "-" __size +(-1) "/2,.5+" @;
put __y1 +(-1) "+" __size +(-1) "*Math.sqrt(3)/4);";
put "   context.lineTo(.5+" __x1 +(-1) ",.5+" __y1 +(-1) "-" @;
put __size +(-1) "*Math.sqrt(3)/4);";
put "   context.closePath();";
put "   context.strokeStyle=""" __scolor +(-1) """;";
put "   context.stroke();";
put "   context.fillStyle=""" __fcolor +(-1) """;";
put "   context.fill();";
```

LINE PLOT

Using the methods described above we can now draw a line plot by drawing lines, labels, a rectangle and symbols. The code to generate the plot below including the macros used can be found on sasCommunity.org.

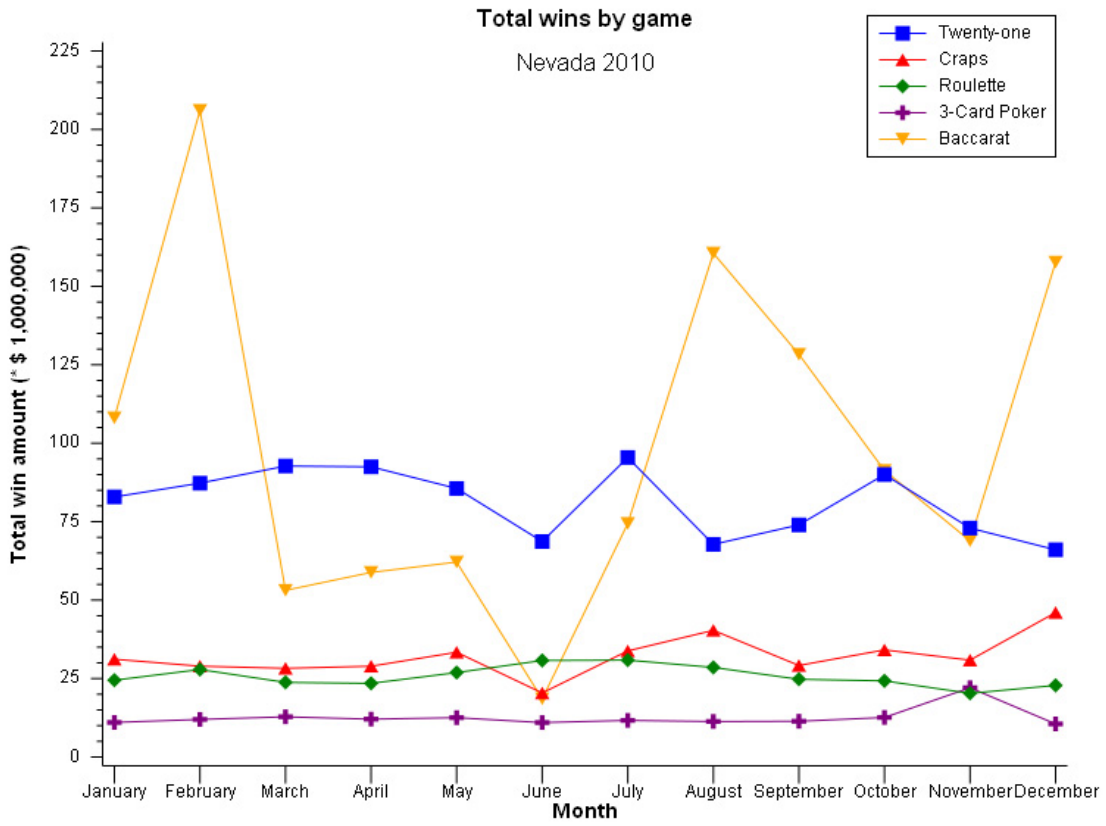

Figure 3 Line plot (Mozilla Firefox 3)

There are slight differences in what the plot looks like between browsers. The main difference is in rendering of text as shown below.

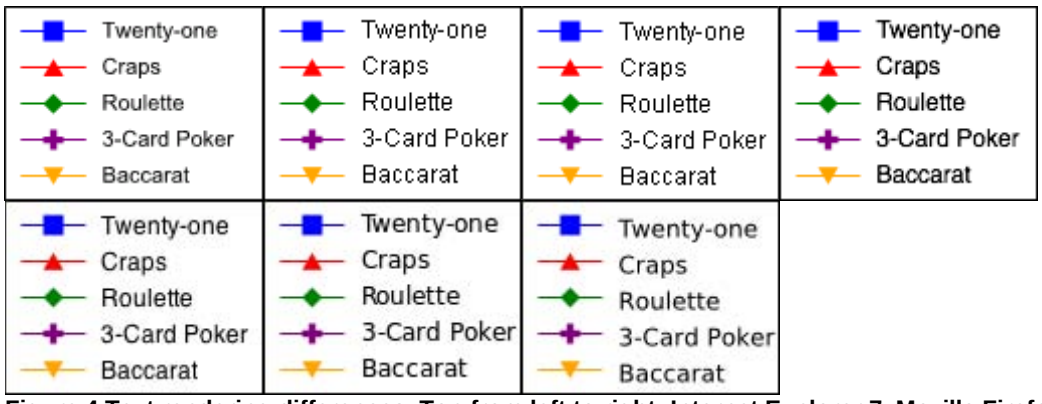

Figure 4 Text rendering differences. Top from left to right: Internet Explorer 7, Mozilla Firefox 3 and Google Chrome 6 on Windows XP and Mozilla Firefox 3 on Mac OS X. Bottom from left to right: Safari 5 on Mac OS X and Firefox 3 and Opera 10 on Linux.

DRAWING CIRCLES

Drawing circles on canvas is done using the arc() method. This method has quite some parameters that need to be specified:

- x and y coordinates to use as center;
- the radius of the circle;
- the start and end angle, since we want a full circle the start angle is 0 and the end angle is 2*π (2*Math.PI in JS);
- a boolean value that determines whether to draw counter clockwise, which is irrelevant when drawing a full circle, but nevertheless should be specified.

```
put "   context.beginPath();";
put "   context.moveTo(.5+" __x1 +(-1) "+" __size +(-1) "/2,.5+" @;
put __y1 +(-1) ");";
put "   context.arc(.5+" __x1 +(-1) ",.5+" __y1 +(-1) "," __size +(-1) @;
put "/2,0,Math.PI*2,true); ";
put "   context.closePath();";
put "   context.strokeStyle=""" __scolor +(-1) """;";
put "   context.stroke();";
put "   context.fillStyle=""" __fcolor +(-1) """;";
put "   context.fill();";
```

Using the arc() method combined with the methods described for the line plot a bubble plot can be created. The input data once again comes from the State Gaming Control Board in Nevada and contains the number of locations and units for slot machines per denomination for December 2010.

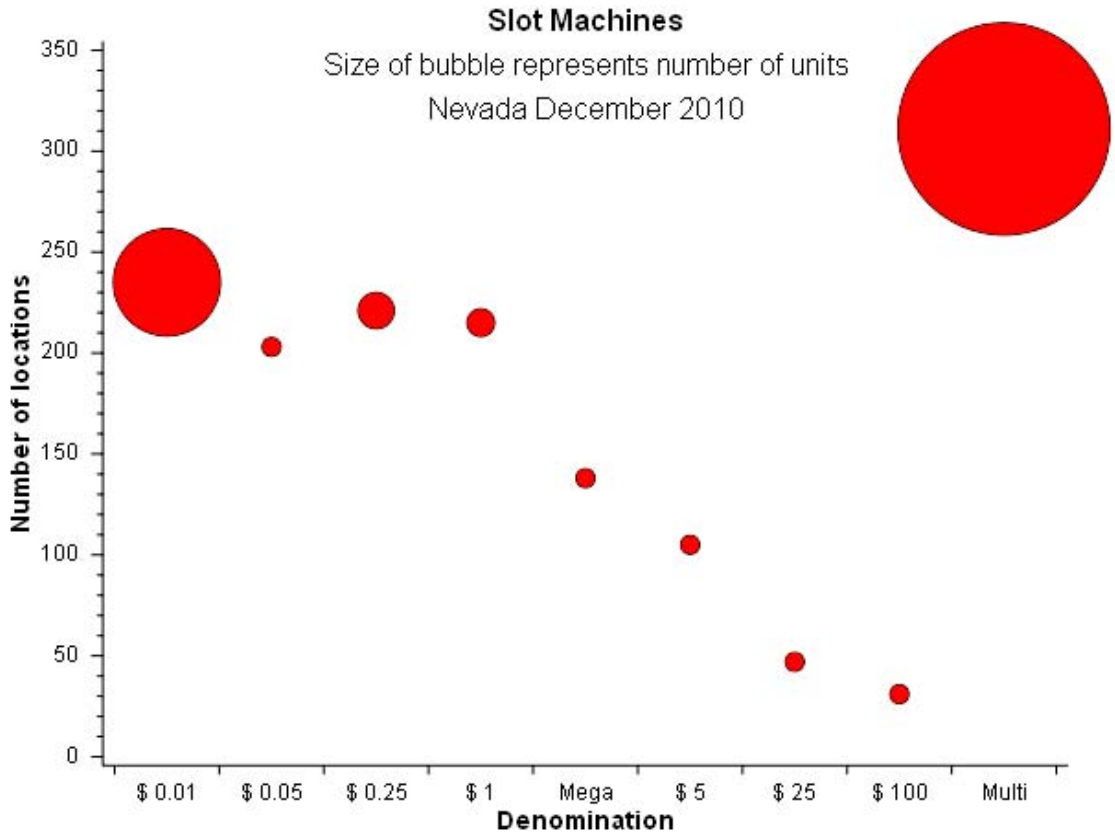

Figure 5 Bubble plot (Mozilla Firefox 3)

In case of bubble plots with overlapping bubbles the globalCompositeOperation attribute becomes interesting. Specifying 'lighter' as value will show overlap as opposed to showing the topmost shape. Note that this will not work using ExplorerCanvas as globalCompositeOperation is not supported.

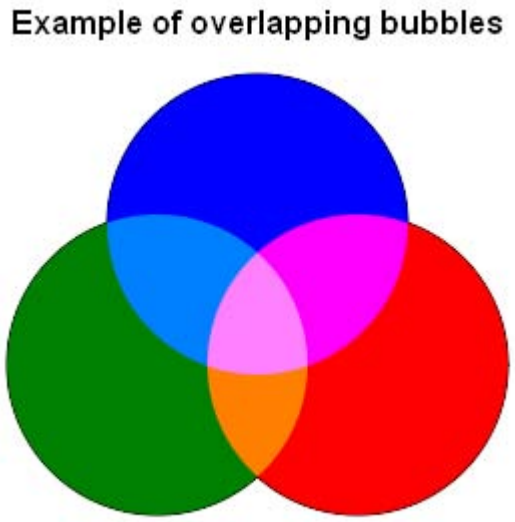

Figure 6 Overlapping bubbles with globalCompositeOperation set to 'lighter' (Mozilla Firefox 3)

DRAWING PIE SLICES

Drawing pie slices uses the arc() method similar to drawing circles, with the difference that start and end angle need to be specified. First tell the canvas that a new path is going to be drawn with the beginPath() method and move to the center of the pie with the moveTo() method. Next draw the outer edge of the slice using the arc() method whilst ensuring that the start angle is the same as the end angle of the previous slice. Using closePath() will close the path and thus connect the beginning and the end of the arc to the center point. After that you can choose to either fill the slice using the fill() method and/or outline the slice using the stroke() method. This once again uses the fillStyle, strokeStyle and, lineWidth attributes.

```
put "   context.beginPath();";
put "   context.moveTo(" __xc +(-1) "," __yc +(-1) ");";
put "   context.arc(" __xc +(-1) "," __yc +(-1) "," __radius +(-1) "," @;
put __piestart +(-1) "*Math.PI*2/100," __pieend +(-1) @;
put "*Math.PI*2/100,false);";
put "   context.closePath();";
put "   context.lineWidth=" __piewid +(-1) ";";
put "   context.strokeStyle=""" __piescol +(-1) """;";
put "   context.stroke();";
put "   context.fillStyle=""" __fcolor +(-1) """;";
put "   context.fill();";
```

When adding some more detail like exploding a slice, adding callouts and/or adding text within the slice sine and cosine functions become important. This can of course be done in SAS using the SIN and COS functions, but JS also allows this with the Math.sin() and Math.cos() functions.

Using the arc() method combined with the methods described for the line plot and bubble plot a pie chart can be created. The input data once again comes from the State Gaming Control Board in Nevada and contains the taxable gaming revenue for Clark County, Nevada by area for 2010.

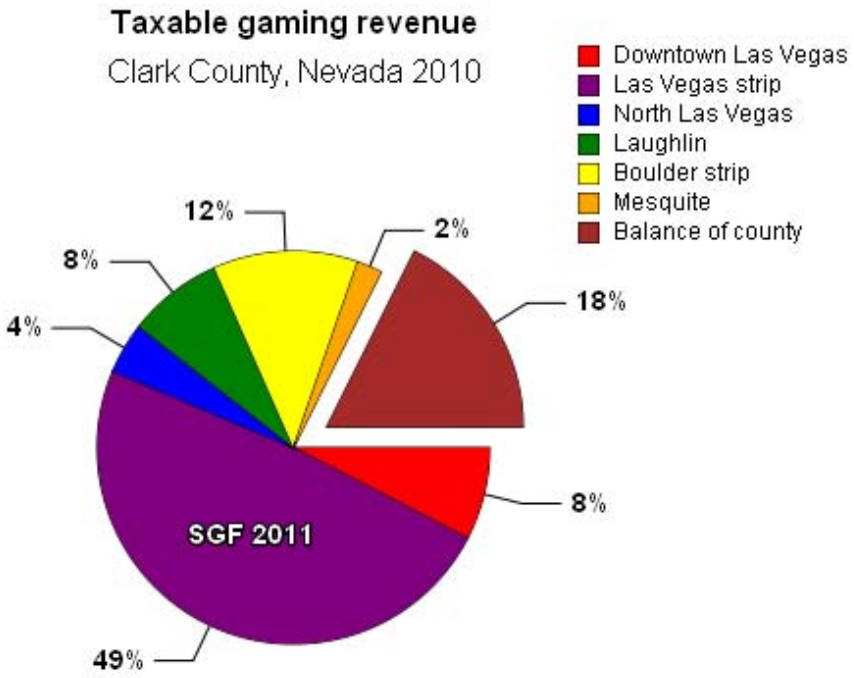

Figure 7 Pie chart (Mozilla Firefox 3)

ADDITIONAL CANVAS METHODS AND ATTRIBUTES

There are more methods and attributes that can be used when drawing on canvas that are not covered in this paper. Additional methods that can be useful include:

- quadraticCurveTo(), bezierCurveTo() and arcTo(): used to created curved lines and arcs from a coordinate to another coordinate;
- createLinearGradient(), createRadialGradient() and addColorStop(): used to create gradients that can be used as fillStyle (note that ExplorerCanvas only supports linear gradients);
- createPattern(); used to create patterns that can be used as fillStyle;
- drawImage(): used to draw an image on canvas.

Additional attributes include:

- globalAlpha: used to set transparency of shapes;
- shadowOffsetX, shadowOffsetY, shadowBlur and shadowColor:: used to add shadows to shapes.

MOUSE OVER EVENTS

Since a canvas graph does not consist of objects no events like mouse over or mouse clicks can be captured for specific areas. However, it is possible to place invisible div elements behind the canvas and assign a title and events that those div elements. The downside of this is that div elements are always rectangular. Let's assume a canvas, with "graph" as id and a width and height of 401, has the following blue rectangle:

```
graph_context.beginPath();
graph_context.moveTo(105.5,105.5);
graph_context.lineTo(105.5,5.5);
graph_context.lineTo(5.5,5.5);
graph_context.lineTo(5.5,105.5);
```

```
graph_context.closePath();
graph_context.strokeStyle="#000000";
graph_context.stroke();
graph_context.fillStyle="#0000ff";
graph_context.fill();
```

Since the canvas needs to be in front of any div elements the z-index should be set using a style attribute.

```
<style type="text/css">
   #graph {
      z-index:5;
   }
</style>
```

The div element should now get a z-index less than the z-index of the graph to ensure it's behind the graph. Width and height should be specified. Title is the text that will be displayed as tool tip on mouse over and cursor is set so that it's clear when the mouse is over the div. The div element should be placed relative to the top left corner of the graph, but this can not be done until the HTML has finished loading. The easiest thing to do now is set the left and top attributes the same as absolute position of the rectangle within the canvas.

```
<div id="blue"
     width="100px"
     height="100px"
     title="Blue"
     style="z-index:1;
            opacity: 0.01;
            position:absolute;
            width:100px;
            height:100px;
            left:5.5px;
            top:5.5px;
            cursor:pointer;"
     onClick="location.href='http://en.wikipedia.org/wiki/Blue';">
```

When the HTML output has finished loading the div element can be moved to the correct spot, by calling a JS function when the body of the HTML has been loaded. The JS function below will take the offset coordinates of the graph and the div element and use that as the new offset coordinates for the div element (doTopLeft('graph','blue');).

```
function doTopLeft(graphId,divId) {
   var graphleft = graphtop = 0;
   var graphobj = document.getElementById(graphId);
   var divobj = document.getElementById(divId);

   if (graphobj.offsetParent) {
      do {
         graphleft += graphobj.offsetLeft;
         graphtop += graphobj.offsetTop;
      } while (graphobj = graphobj.offsetParent);
      divobj.style.left=divobj.offsetLeft + graphleft + "px";
      divobj.style.top=divobj.offsetTop + graphtop + "px";
   }
}
```

The div elements should now be placed in the correct spot as seen by hovering over or clicking on a square in the graph.

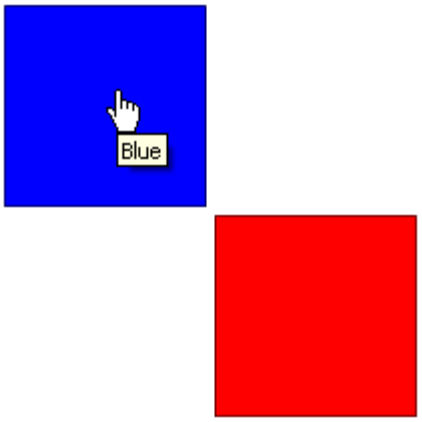

Figure 8 Div element behind graph with mouse events

MULTIPLE CANVAS GRAPHS ON A PAGE

In the examples so far it was assumed that the HTML output would contain one graph only and thus the DATA step could also write the body tags and add onLoad events to the opening body tag. When the output contains multiple graphs this is no longer possible. Luckily JS functions and style sheets can also be included in the body of HTML output, but then these functions need to be added to the onLoad event afterwards. Assuming a page contains two canvas elements and the JS functions that draw on these elements are called drawit1 and drawit2. At any place in the HTML output these functions can be added to the onLoad event with the following JS:

```
function addLoadEvent(func) {
   var oldonload = window.onload;
   if (typeof window.onload != 'function') {
      window.onload = func;
   } else {
      window.onload = function() {
         if (oldonload) {
            oldonload();
         }
         func();
      }
   }
}

addLoadEvent(drawit1);
addLoadEvent(drawit2);
```

This will ensure that both canvas elements will be filled upon completion. When adding multiple canvas elements to a single page keep in mind that all canvas elements need to have a unique id.

CONCLUSION

The canvas element is an interesting tool to create graphs using a DATA step. Just a few JS methods are needed to accomplish this. With some div elements mouse over events and drill down can be added making it an alternative to other graphics formats.

REFERENCES

- http://www.sascommunity.org/wiki/Can_SAS%C2%AE_Do_Canvas%3F_Creating_Graphs_Using_HTML_5_Canvas_Elements sasCommunity.org page that contains code for the examples in this paper
- http://code.google.com/p/explorercanvas/ ExplorerCanvas: JS library to add canvas functionality to Internet Explorer (use the most recent version from the SVN trunk)
- http://diveintohtml5.org Dive Into HTML5 by Mark Pilgrim

- http://blog.nihilogic.dk/2009/05/canvas-cheat-sheet-update.html Canvas Cheat Sheet

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Edwin J. van Stein
Global Data Science – Global Clinical Programming
Astellas Pharma Global Development Europe
Elisabethhof 19 P.O. Box 108
2350 AC Leiderdorp
The Netherlands
edwin.stein@eu.astellas.com / ejvanstein@gmail.com
http://nl.linkedin.com/in/ejvanstein