Graphics in SVG

Relevant Links

- Comparison of Canvas and SVG[1]
- MDN start page for graphics on the web[2]
- MDN start page for SVG[3]
- ACM Learning center books on SVG[4] and Canvas[5].
- SVG.js library[6].
- SVG Paths details[7].
- MDN guide to paths[8]
- SVG path tutorial at CSS-tricks[9].

Notes

Graphics in Javascript

Javascript offers three main systems for doing more elaborate graphics.

- **Canvas** is essentially more of image drawing. You create a <canvas> element and then use Javascript instructions to draw into that canvas as you would do on a painting.
- **SVG** stands for "Scalable Vector Graphics". These are actually more like a set of DOM Elements that you can manipulate, but whose intent is to represent "vector graphics" elements.
- **WebGL** is infrastructure for doing 3D graphics (while the other two focus on 2D graphics).

We will focus on SVG for now, which is based on the SVG XML specification.

SVG Graphics

Here is an example of how an SVG graphic might look in the code:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
  viewBox="0 0 100 100" preserveAspectRatio="xMidYMid slice"
  style="width:100%; height:100%; position:absolute; top:0; left:0; z-index:-1;">
```

[1] https://blogs.msdn.microsoft.com/ie/2011/04/22/thoughts-on-when-to-use-canvas-and-svg/
[2] https://developer.mozilla.org/en-US/docs/Web/Guide/Graphics
[3] https://developer.mozilla.org/en-US/docs/Web/SVG
[4] https://learning.acm.org/books/book_detail.cfm?id=940325&type=24
[5] https://learning.acm.org/books/book_detail.cfm?id=1973125&type=24
[6] http://svgjs.com/
[7] https://www.w3.org/TR/SVG/paths.html#PathData
[8] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths
[9] https://css-tricks.com/svg-path-syntax-illustrated-guide/

```
  <linearGradient id="gradient">
    <stop class="begin" offset="0%"/>
    <stop class="end" offset="100%"/>
  </linearGradient>
  <rect x="0" y="0" width="100" height="100" style="fill:url(#gradient)" />
  <circle cx="50" cy="50" r="30" style="fill:url(#gradient)" />
</svg>
```

So in many ways it is just HTML elements, except that they are a separate set of such elements than the normal HTML elements. But the syntax is very similar, and they can be targetted and manipulated like HTML elements.

Here are the key concepts to be familiar with:

- There are some standard elements for circles, rectangles, lines etc
- A path element can be used for arbitrary curves
- transform elements can be used to create shape transformations (e.g. rotation)
- We can set up handlers to respond to events on the shapes
- A defs element can be used to define elements that are referenced elsewhere
- You can define a clip-path attribute to an element, to specify that only a specific region of that element is to be drawn and interacted on (as if everything else was cut away). You can see some examples here[10].
- You can have text that goes along a path.

Elements Here are the main SVG elements we can create:

svg This is the container element. Physical dimensions are set here.
circle For drawing a circle with a given center and radius.
ellipse For drawing ellipses.
g Used for grouping elements together (and setting custom transforms to them, for example).
line For drawing a line connecting two points.
path Used for arbitrary curves.
polyline For a series of connected lines. There is also a **polygon** element that closes the endpoints.
rect For drawing rectangles.
text For adding text to graphs.

Of course creating this code on your own can be fairly painful. There are libraries to help you along, and SVG.js[11] is the one we will use. Setting it up could be as simple as including the following script on your page:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/svg.js/2.7.1/svg.min.js"></script>
```

or dynamically on a page using the following:

[10] https://developer.mozilla.org/en-US/docs/Web/SVG/Element/clipPath
[11] http://svgjs.com/

```
var s = document.createElement('script');
s.setAttribute('src', 'https://cdnjs.cloudflare.com/ajax/libs/svg.js/2.7.1/svg.min.js');
document.body.appendChild(s);
```

This gives us an SVG global object to use. We can now use that to create elements. Here's a typical interaction:

```
var aDiv = document.createElement('div');
aDiv.setAttribute('id', 'myDrawing');
document.body.prepend(aDiv);
var draw = SVG('myDrawing').size(100, 300);
```

This creates a new div element, and creates a new SVG empty element within it. Let's add a rectangle in it:

```
var rect = draw.rect(60, 100).attr({ fill: '#f06' });
```

Let's move it, then change its color:

```
rect.animate().move(0, 50);
rect.animate().fill('#f55');
```

And let's read its coordinate values. And then change the x:

```
rect.x(); rect.y();
rect.x(50);
```

We can also round its corners:

```
rect.radius(10);
rect.animate()
  .radius(40);
```

You can find more things you can do with each element here[12].

Let's work with a polyline, and then do some animation on it:

```
var polyline = draw.polyline('0,0 100,50 50,100')
  .fill('none').
  stroke({ width: 1 });
polyline.animate(2000)
  .plot([[0,0], [100,50], [50,100], [150,50], [200,50], [250,100], [300,50], [350,50]]);
```

SVG paths can do a lot of cool stuff, though they need some more work.

Paths Paths are a challenging topic, so they deserve some more discussion. A path element contains a d attribute, which is a series of instructions to an imaginary "marker".

- Each instruction starts with a letter indicating what kind of instruction it is.
- This is typically followed by pairs of coordinates.
- Uppercase letters indicate absolute coordinates, while lowercase letters indicate relative coordinates.

[12] http://svgjs.com/elements/

- Roughly there are *move commands*, *line commands*, *curve commands* and *arc commands*.
- For the curve commands you'll need to understand a little bit Bezier curves. In essence there are two kinds of Bezier curves:
  - Cubic Bezier curves have a start and end point, but also two *control points* that suggest how the curve should behave. The curve starts from the first point *towards the first control point* and ends in the direction *from the second control point towards the second point*. The curve never actually passes through the control points.
  - Quandratic Bezier curves only use one control point, and it is used to guide both the start and the end.
  - If you chain many Bezier curves together you can tell it to use as a first control point the reflection of the previous control point. This makes for a smoother path.

Let us look at some examples:

```
M 15, 20      <————————   Move to the coordinates (15, 20)
m 20, 30      <————————   Move 20 pixels to the right and 30 pixels down

L 30, 40      <————————   Draw a straight line to location (30, 40)
l −30, 40     <————————   Draw a straight line going 30 pixels to the left and 40 pixels down
H 30          <————————   Draw a straight horizontal line to the point with x coordinate 30
v 10          <————————   Draw a straight vertical line going 10 pixels down
Z             <————————   Close the path, joining the current point to the start

C 10,20 20,20 30,10  <———— Draw a "cubic bezier curve" to the point (30, 10) using the two
S 20,20 10,10   <———— Continue the previous cubic bezier step, and use as a first control p
c, s    <———— all coordinates are relative
Q 20,20 30,30   <———— A quadratic curve from the current point to the point (30, 30), usin
T, t     <————— Continue a quadratic curve with the reflected control point.

A        <————— Used for arcs. We will not discuss these.
```

All this is pretty complicated in the abstract, perhaps some examples would demonstrate. The following attempts to draw the letter G:

```
var letterG = draw.path('M 50,50 v −2 h 10 v 2 c −5,0 −5,5 −5,30 c −40,15 −40,0 −40,−32 s 0,−
```

Practice: Try to create some other letters.

Gradients and Patterns You can create interesting fills with gradients and patterns. Here's an example of a gradient from the SVG.js documentation[13]:

```
var gradient = draw.gradient('linear', function(stop) {
  stop.at(0, '#333');
  stop.at(0.5, '#A44');
  stop.at(1, '#fff');
}).from(0, 0).to(1, 1);

var c = draw.circle().x(50).y(50).radius(30).fill(gradient);
```

[13] http://svgjs.com/elements/gradient/

We can also create patterns from any existing elements, then use them to fill other elements. For example here's a checkered pattern:

```
var pattern = draw.pattern(20, 20, function(add) {
  add.rect(20,20).fill('#f06');
  add.rect(10,10);
  add.rect(10,10).move(10,10);
});

var c = draw.circle().x(50).y(50).radius(30).fill(pattern);
```

Transforms There are a number of transformations. We'll make a simple clock using them to rotate the indices.

```
var circle = draw.circle(80).cx(50).cy(50).fill('white').stroke('black');
var secIndex = draw.path('M 50,50 v -38').stroke('black').fill('white');
var minIndex = draw.group();
minIndex.path('M 50,50 m 0,-35 m -5,5 l 5,-5 l 5,5').stroke('black').fill('white').attr('stro
minIndex.path('M 50,50 v -35').stroke('black').fill('white').attr('stroke-width', 2);
var hourIndex = draw.group();
hourIndex.path('M 50,50 m 0,-15 m -5,5 l 5,-5 l 5,5 m -5,-5').stroke('black').fill('white').a
hourIndex.path('M 50,50 v -15').stroke('black').fill('white').attr('stroke-width', 3);

function showTime() {
  let now = new Date();
  secIndex.transform({ rotation: (6 * now.getSeconds()), cx: 50, cy: 50 });
  minIndex.transform({ rotation: (6 * now.getMinutes()), cx: 50, cy: 50 });
  hourIndex.transform({ rotation: (6 * now.getHours()), cx: 50, cy: 50 });
}
showTime();
var t = setInterval(showTime, 1000);
```

Practice 1: Add ticks at every hour. Start by putting one at 12 o'clock, then using use and rotation for the other 11.

Practice 2: Add the hours numbers.

Events on SVG SVG elements are normal DOM elements, and we can therefore place handlers on them to react when they are clicked etc. In this section we will decide a "door" which when clicked will open.

Let's start with the door basics. We want a rectangular frame containing another rectangle.

```
var aDiv = document.createElement('div');
aDiv.setAttribute('id', 'door');
aDiv.setAttribute('style', 'height: 200px;width:80px;');
document.body.prepend(aDiv);
var draw = SVG('door').size(100, 200);

var frame = draw.rect(80, 120).x(10).y(10).fill('white').stroke('purple').attr('stroke-width'
var door = draw.rect(76, 116).x(12).y(12).fill('magenta').stroke('magenta');
```

Then we want the door to "slide" when the user clicks it:

```
door.click(function() {
    this.animate().transform({ relative: true, scaleX: 0, cx: 90, cy: 130 });
});
```

You can read more about the available SVG transformations over here[14].

[14] https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/transform