Interactive Data Visualisation

Information Visualisation and SVG

Dr Ruairi O’Reilly

Scalar Vector Graphics

SVG

- An XML-based vector image format for 2D graphics with support for interactivity and animation.
- The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999.
- SVG images and their behaviors are defined in XML text files. This means that they can be searched, indexed, scripted, and compressed.
- As XML files, SVG images can be created and edited with any text editor, as well as with drawing software.
- All major modern web browsers have SVG rendering support.

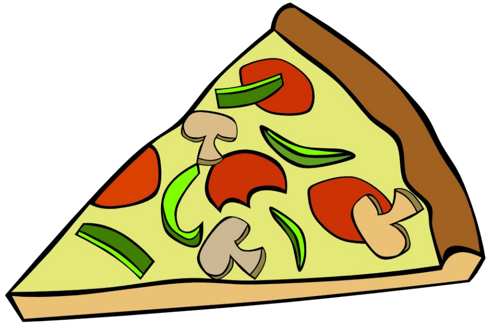
Reference - wiki article

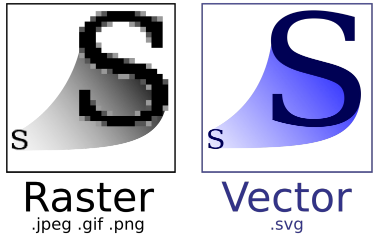
SVG allows three types of graphic objects:

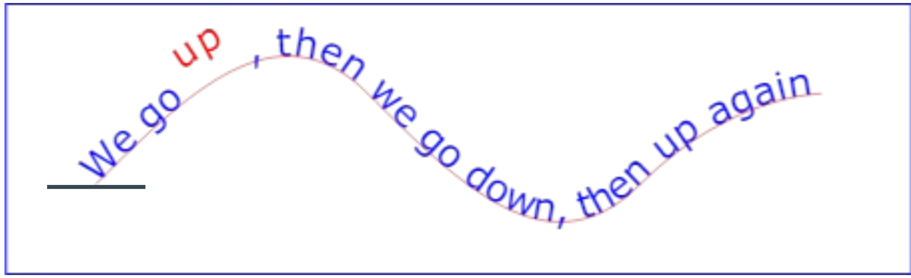
Vector graphic (shapes such as paths and outlines consisting of straight lines and curves)

Bitmap images

Text







Graphical objects can be:

Grouped, styled, transformed and composited into previously rendered objects.

The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and can include animation, defined in the SVG XML elements or via scripting that accesses the SVG Document Object Model (DOM). SVG uses CSS for styling and JavaScript for scripting.

Internationalization and localization - text appearing in plain text within the SVG DOM enhances the accessibility of SVG graphics.

The SVG specification was updated to version 1.1 in 2011. There are two 'Mobile SVG Profiles,' SVG Tiny and SVG Basic, meant for mobile devices with reduced

Printing: Though the SVG Specification primarily focuses on vector graphics markup language, its design includes the basic capabilities of a page description language like Adobe's PDF. It contains provisions for rich graphics, and is compatible with CSS for styling purposes. SVG has the information needed to place each glyph and image in a can element of a field which is used to define a vector space.hosen location on a printed page.

Scripting and animation: SVG drawings can be dynamic and interactive. Time-based modifications to the elements can be described in Synchronised Mutltimedia Integration Language (SMIL), or can be programmed in a scripting language (JavaScript). The W3C explicitly recommends SMIL as the standard for animation in SVG.

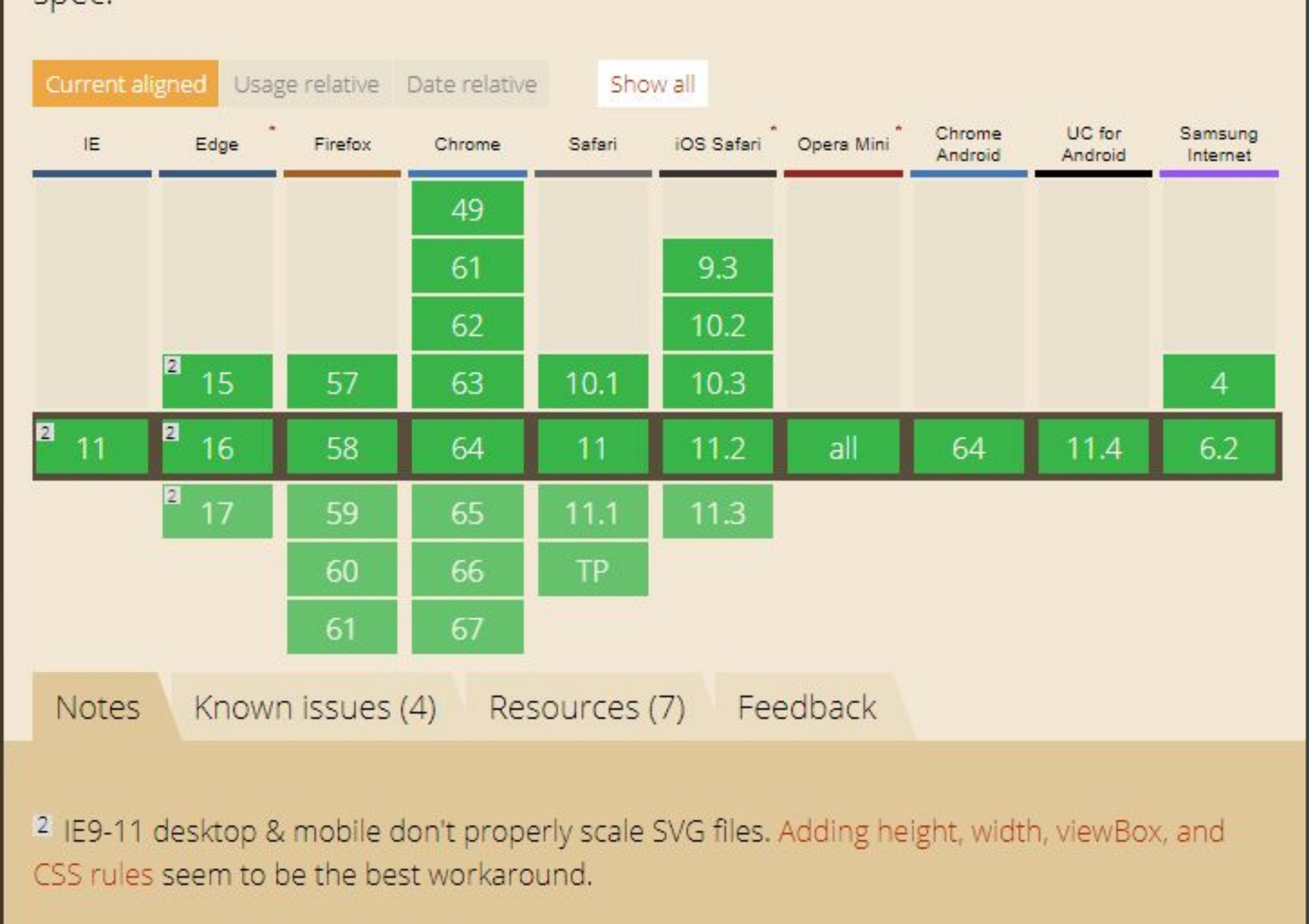
A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object.

Compression: SVG images, being XML, contain many repeated fragments of text, so they are well suited for lossless data compression algorithms.

SVG images that have been compressed with the industry standard gzip algorithm are referred to as an "SVGZ" image and uses the corresponding .svgz filename extension.

Conforming SVG 1.1 viewers will display compressed images. An SVGZ file is typically 20 to 50 percent of the original size.

Why not read a bit more about the dev history of SVG on the Wikipedia page from which this content was taken. WIKI spec.



IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome Android	UC for Android	Samsung Internet
			49						
			61		9.3				
			62		10.2				
	[2] 15	57	63	10.1	10.3				4
[2] 11	[2] 16	58	64	11	11.2	all	64	11.4	6.2
	[2] 17	59	65	11.1	11.3				
		60	66	TP					
		61	67						

[2] IE9-11 desktop & mobile don't properly scale SVG files. Adding height, width, viewBox, and CSS rules seem to be the best workaround.

SVG - A Simple Example - JSFiddle

The rendering process involves the following:

1) We start with the svg root element:

- a doctype declaration as known from (X)HTML should be left off because DTD based SVG validation leads to more problems than it solves
- to identify the version of the SVG for other types of validation the version and baseprofile attributes should always be used instead
- as an XML dialect, SVG must always bind the namespaces correctly (in the xmlns attribute).

2) The background is set to red by drawing a rectangle <rect/> that covers the complete image area

3) A green circle <circle/> with a radius of 80px is drawn atop the center of the red rectangle (offset 30+120px inward, and 50+50px upward).

4) The text "SVG" is drawn. The interior of each letter is filled in with white. The text is positioned by setting an anchor at where we want the midpoint to be: in this case, the midpoint should correspond to the center of the green circle. Fine adjustments can be made to the font size and vertical position to ensure the final result is aesthetically pleasing.

Basic properties of SVG files

The order of rendering of elements for SVG files - later elements are rendered atop previous elements (The further down an element is the more will be visible). SVG files on the web can be displayed directly in the browser or embedded in HTML files via several methods:

- If the HTML is XHTML and is delivered as type application/xhtml+xml, the SVG can be directly embedded in the XML source.
- If the HTML is HTML5, and the browser is a conforming HTML5 browser, the SVG can be directly embedded, too. However, there may be syntax changes necessary to conform to the HTML5 specification
- The SVG file can be referenced with an object element.
- Likewise an iframe element can be used.
- An img element can be used theoretically, too. However this technique doesn't work in Firefox before 4.0.
- Finally SVG can be created dynamically with JavaScript and injected into the HTML DOM. With this method replacement technologies can be implemented for browsers which normally can't process SVG.

SVG File Types - .svg & .svgz

SVG files come in two flavors. Normal SVG files are simple text files containing SVG markup. The recommended filename extension for these files is ".svg" (all lowercase).

The SVG specification also allows for gzip-compressed SVG files (Geographic data = big files). The recommended filename extension for these files is ".svgz" (all lowercase). Unfortunately it is very problematic to get gzip-compressed SVG files to work reliably across all SVG capable user agents when served from Microsofts IIS server, and Firefox can not load gzip-compressed SVG from the local computer. Avoid gzip-compressed SVG except when you are publishing to a webserver that you know will serve it correctly.

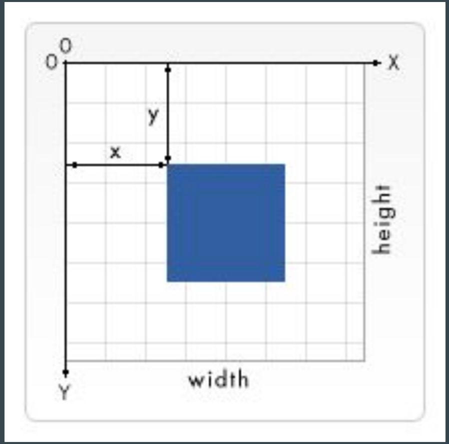
SVG Positions and size of objects - The grid

Lets look at how SVG represents the positions and sizes of objects within a drawing context, including coordinate system and what a "pixel" measurement means in a scalable context.

SVG uses a coordinate system or grid system similar to the one used by canvas. The top left corner of the document is considered to be the point (0,0), or point of origin.

Positions are then measured in pixels from the top left corner, with the positive x direction being to the right, and the positive y direction being to the bottom.

Note that this is different to the way you're taught to graph in Maths. However, this is the same way elements in HTML are positioned (By default, LTR documents are considered not the RTL documents which position X from right-to-left).



What are "pixels"?

In the most basic case one pixel in an SVG document maps to one pixel on the output device (a.k.a. the screen). But SVG wouldn't have the "Scalable" in its name, if there weren't several possibilities to change this behaviour. Much like absolute and relative font sizes in CSS, SVG defines absolute units (ones with a dimensional identifier like "pt" or "cm") and so-called user units, that lack that identifier and are plain numbers. Without further specification, one user unit equals one screen unit. To explicitly change this behaviour, there are several possibilities in SVG. We start with the svg root element:

```
<svg width="100" height="100">
```

The above element defines a simple SVG canvas with 100x100px. One user unit equals one screen unit.

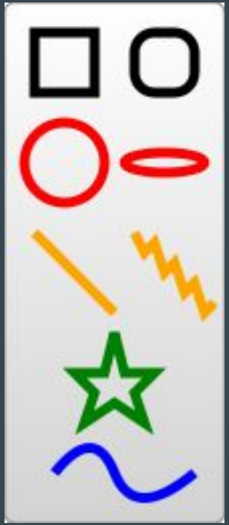
```
<svg width="200" height="200" viewBox="0 0 100 100">
```

The whole SVG canvas here is 200px by 200px in size. However, the viewBox attribute defines the portion of that canvas to display. These 200x200 pixels display an area that starts at user unit (0,0) and spans 100x100 user units to the right and to the bottom. This effectively zooms in on the 100x100 unit area and enlarges the image to double size.

The current mapping of user units to screen units is called user coordinate system. Apart from scaling the coordinate system can also be rotated, skewed and flipped. The default user coordinate system maps one user pixel to one device pixel *. Lengths in the SVG file with specific dimensions, like "in" or "cm", are then calculated in a way that makes them appear 1:1 in the resulting image.

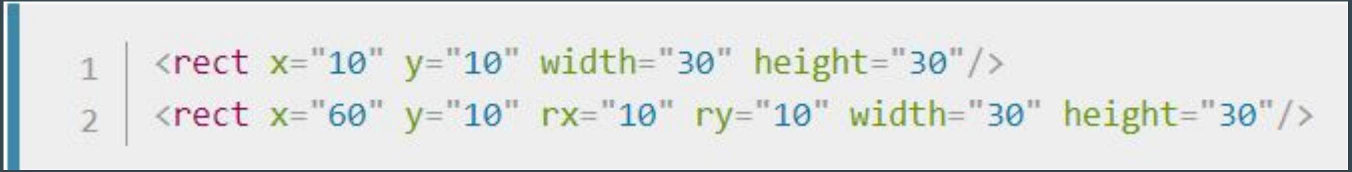
Basic shapes

To insert a shape, you create an element in the document. Different elements correspond to different shapes and take different attributes to describe the size and position of those shapes. Some are slightly redundant in that they can be created by other shapes, but they're all there for your convenience and to keep your SVG documents as short and as readable as possible. All the basic shapes are shown in the image to the right. The code to generate this is available at: JSFiddle



Rectangles

The rect element does exactly what you would expect and draws a rectangle on the screen. There are really only 6 basic attributes that control the position and shape of the rectangle on screen here. The image shown earlier shows two rect elements, which I admit is a bit redundant. The one on the right has its rx and ry attributes set, giving it rounded corners. If they're not set, they default to 0.

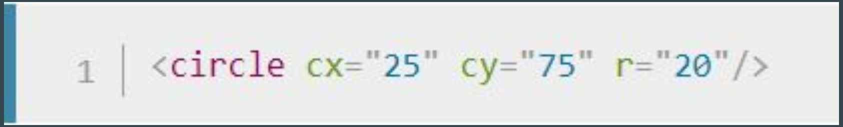


```
<rect x="10" y="10" width="30" height="30"/>
<rect x="60" y="10" rx="10" ry="10" width="30" height="30"/>
```

- x - The x position of the top left corner of the rectangle.
- y - The y position of the top left corner of the rectangle.
- width - The width of the rectangle
- height - The height of the rectangle
- rx - The x radius of the corners of the rectangle
- ry - The y radius of the corners of the rectangle

Circle

As you would have guessed, the circle element draws a circle on the screen. There are really only 3 attributes that are applicable here.

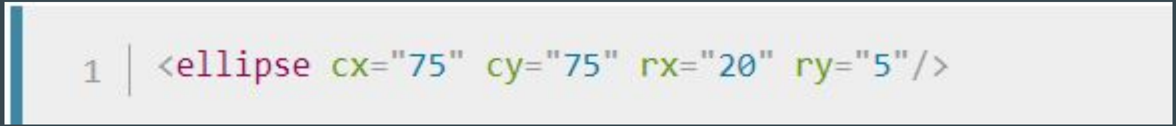


```
<circle cx="25" cy="75" r="20"/>
```

- r - The radius of the circle.
- cx - The x position of the center of the circle.
- cy - The y position of the center of the circle.

Ellipse

Ellipses are actually just a more general form of the circle element, where you can scale the x and y radius (commonly called the semimajor and semiminor axis by math people) of the circle separately.

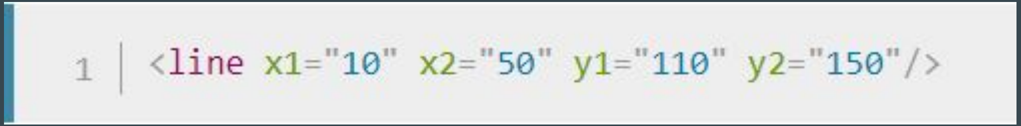


```
<ellipse cx="75" cy="75" rx="20" ry="5"/>
```

- rx - The x radius of the ellipse.
- ry - The y radius of the ellipse.
- cx - The x position of the center of the ellipse.
- cy - The y position of the center of the ellipse.

Line

Lines are again, just straight lines. They take as attributes two points which specify the start and end point of the line.

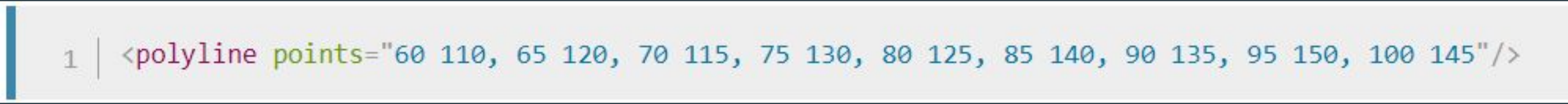


```
<line x1="10" x2="50" y1="110" y2="150"/>
```

- x1 - The x position of point 1.
- y1 - The y position of point 1.
- x2 - The x position of point 2.
- y2- The y position of point 2.

Polyline

Polylines are groups of connected straight lines. Since that list can get quite long, all the points are included in one attribute:

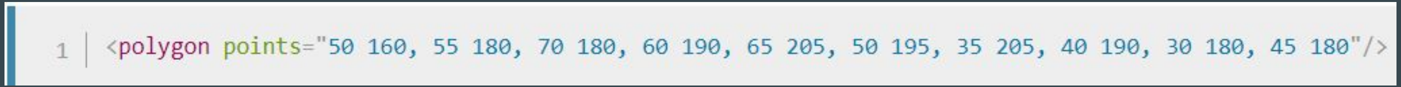


```
<polyline points="60 110, 65 120, 70 115, 75 130, 80 125, 85 140, 90 135, 95 150, 100 145"/>
```

- points - A list of points, each number separated by a space, comma, EOL, or a line feed character. Each point must contain two numbers, an x coordinate and a y coordinate. So the list (0,0), (1,1) and (2,2) could be written: "0 0, 1 1, 2 2".

Polygon

Polygons are a lot like polylines in that they're composed of straight line segments connecting a list a points. For polygons though, the path automatically returns to the first point for you at the end, creating a closed shape. Note that a rectangle is a type of polygon, so a polygon can be used to create a <rect/> element in cases where you need a little more flexibility.

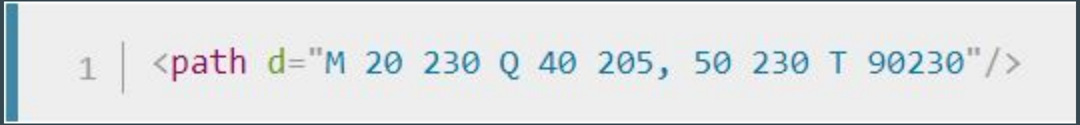


```
<polygon points="50 160, 55 180, 70 180, 60 190, 65 205, 50 195, 35 205, 40 190, 30 180, 45 180"/>
```

- points - A list of points, each number separated by a space, comma, EOL, or a line feed character. Each point must contain two numbers, an x coordinate and a y coordinate. So the list (0,0), (1,1) and (2,2) could be written: "0 0, 1 1, 2 2". The drawing then closes the path, so a final straight line would be drawn from (2,2) to (0,0).

Path

Path is probably the most general shape that can be used in SVG. Using a path element you can draw rectangles (with or without rounded corners), circles, ellipses, polylines, and polygons. Basically any of the other types of shapes, bezier curves, quadratic curves, and many more. For that reason, paths alone will be the next section in this tutorial, but for now I will just point out that there is a single attribute used to control its shape.



```
<path d="M 20 230 Q 40 205, 50 230 T 90230"/>
```

- d - A list of points and other information about how to draw the path. See the Paths section for more information.

Paths

The <path> element is the most powerful element in the SVG library of basic shapes. You can use it to create lines, curves, arcs and more. Paths create complex shapes by combining multiple straight lines or curved lines. Complex shapes composed only of straight lines can be created as polylines. While polylines and paths can create similar-looking shapes, polylines require a lot of small straight lines to simulate curves and don't scale well to larger sizes. A good understanding of paths is important when drawing SVGs. While creating complex paths using an XML editor or text editor is not recommended, understanding how they work will allow you to identify and repair display issues in SVGs.

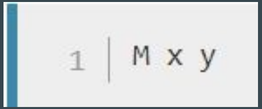
The shape of a path element is defined by one attribute: *d* . The "*d*" attribute contains a series of commands and parameters used by those commands.

Each of the commands is instantiated (for example, creating a class, naming and locating it) by a specific letter. For instance, let's move to the x and y coordinates (10, 10). The "Move to" command is called with the letter M. When the parser runs into this letter, it knows you want to move to a point. So, to move to (10,10) you would use the command "M 10 10". After that, the parser begins reading for the next command.

All of the commands also come in two variants. An uppercase letter specifies absolute coordinates on the page, and a lowercase letter specifies relative coordinates (e.g. move from the last point 10px up and 7px to the left). Coordinates in the "d" attribute are always unitless and hence in the user coordinate system. Later, we will learn how paths can be transformed to suit other needs.

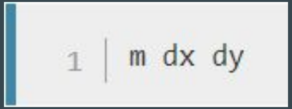
Line commands

here are five line commands for <path> nodes. As the name suggests, each one just draws a straight line between two points. The first command is the "Move To" or M, which was described above. It takes two parameters, a coordinate ' x ' and coordinate ' y ' to move to. If your cursor already was somewhere on the page, no line is drawn to connect the two places. The "Move To" command appears at the beginning of paths to specify where the drawing should start. e.g. :



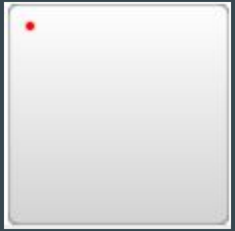
```
M x y
```

or



```
m dx dy
```

In the following example we only have a point at (10,10). Note, though, that it wouldn't show up if you were just drawing the path normally. For example: JSFIDDLE



SVG File Types

There are three commands that draw lines. The most generic is the "Line To" command, called with L. L takes two parameters—x and y coordinates—and draws a line from the current position to a new position.

```
L x y (or l dx dy)
```

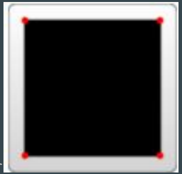
There are two abbreviated forms for drawing horizontal and vertical lines. H draws a horizontal line, and V draws a vertical line. Both commands only take one argument since they only move in one direction.

```
H x (or h dx)
 V y (or v dy)
```

An easy place to start is by drawing a shape. We will start with a rectangle (the same type that could be more easily be made with a <rect> element). It's composed of horizontal and vertical lines only:

An easy place to start is by drawing a shape.

We will start with a rectangle (the same type that could be more easily be made with a <rect> element). It's composed of horizontal and vertical lines only: JSFIDDLE



We can shorten the above path declaration a little bit by using the "Close Path" command, called with Z. This command draws a straight line from the current position back to the first point of the path. It is often placed at the end of a path node, although not always. There is no difference between the uppercase and lowercase comm

```
Z (or z)
```

So our path above could be shortened to:

```
<path d="M10 10 H 90 V 90 H 10 Z" fill="transparent" stroke="black"/>
```

You can also use the relative form of these commands to draw the same picture. Relative commands are called by using lowercase letters, and rather than moving the cursor to an exact coordinate, they move it relative to its last position. For instance, since our box is 80 x 80, the path element could have been written:

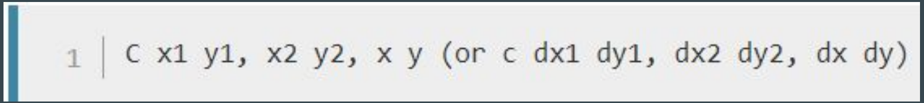
```
<path d="M10 10 h 80 v 80 h -80 Z" fill="transparent" stroke="black"/>
```

In these examples, it would probably be simpler to use the <polygon> or <polyline> elements. However, paths are used so often in drawing SVG that developers may be more comfortable using them instead. There is no real performance penalty or bonus for using one or the other.

Curve commands

There are three different commands that you can use to create smooth curves. Two of those curves are Bezier curves, and the third is an "arc" or part of a circle. For a complete description of the math behind Bezier curves, go to a reference like the one on Wikipedia. There are an infinite number of Bezier curves, but only two simple ones are available in path elements: a cubic one, called with C, and a quadratic one, called with Q.

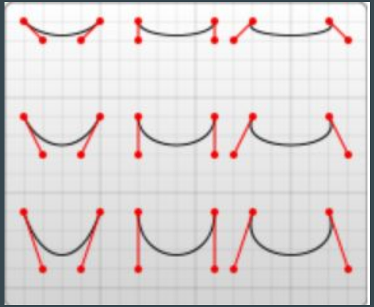
The cubic curve, C, is the slightly more complex curve. Cubic Beziers take in two control points for each point. Therefore, to create a cubic Bezier, you need to specify three sets of coordinates.



```
C x1 y1, x2 y2, x y (or c dx1 dy1, dx2 dy2, dx dy)
```

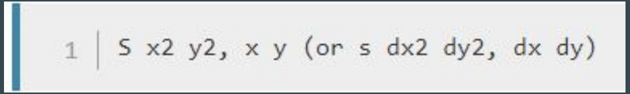
The last set of coordinates here (x,y) are where you want the line to end. The other two are control points. The control point for the start of your curve is (x1,y1), and (x2,y2) is the end point of your curve. The control points essentially describe the slope of your line starting at each point. The Bezier function then creates a smooth curve that transfers you from the slope you established at the beginning of your line, to the slope at the other end.

Bezier curve - JSFIDDLE (cubic) JSFIDDLE (cubic S)

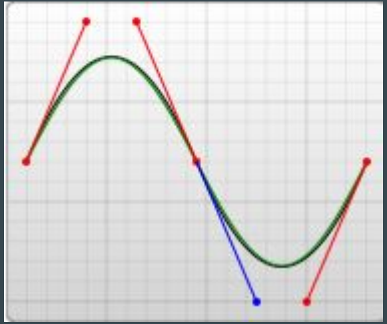


The example depicted creates nine Cubic Bezier curves. As the curves move toward the left, the control points become spread out horizontally. As it moves towards the right, they become further separated from the end points. The thing to note here is that the curve starts in the direction of the first control point, and then bends so that it arrives along the direction of the second control point.

You can string together several Bezier curves to create extended, smooth shapes. Often, the control point on one side of a point will be a reflection of the control point used on the other side to keep the slope constant. In this case, you can use a shortcut version of the cubic Bezier, designated by the command S (or s).



```
S x2 y2, x y (or s dx2 dy2, dx dy)
```



S produces the same type of curve as earlier, but if it follows another S command or a C command, the first control point is assumed to be a reflection of the one used previously. If the S command doesn't follow another S or C command, then the current position of the cursor is used as the first control point. In this case the result is the same as what the Q command would have produced with the same parameters. An example of this syntax is shown below, and in the figure to the left the specified control points are shown in red, and the inferred control point in blue.

Bezier curve - JSFIDDLE (Quadratic) JSFIDDLE (Q T)

The other type of Bezier curve, the quadratic curve called with Q, is actually a simpler curve than the cubic one. It requires one control point which determines the slope of the curve at both the start point and the end point. It takes two arguments: the control point and the end point of the curve.

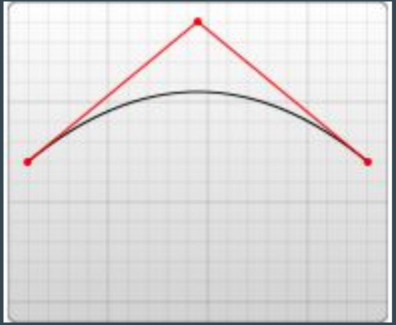
```
Q x1 y1, x y (or q dx1 dy1, dx dy)
```

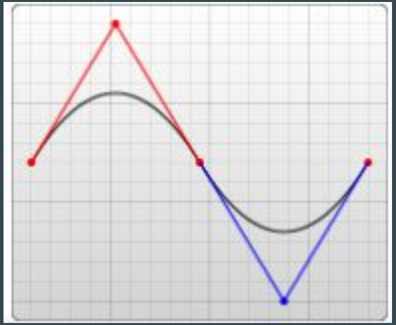
As with the cubic Bezier curve, there is a shortcut for stringing together multiple quadratic Beziers, called with T.

```
T x y (or t dx dy)
```

This shortcut looks at the previous control point you used and infers a new one from it. This means that after your first control point, you can make fairly complex shapes by specifying only end points. Note: This only works if the previous command was a Q or a T command. If it is not, then the control point is assumed to be the same as the previous point, and you'll only draw lines.

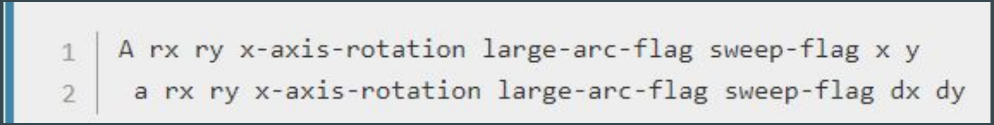
Both curves produce similar results, although the cubic allows you greater freedom in exactly what your curve looks like. Deciding which curve to use is situational and depends on the amount of symmetry your line has.





Arcs

The other type of curved line you can create using SVG is the arc, called with A. Arcs are sections of circles or ellipses. For a given x-radius and y-radius, there are two ellipses that can connect any two points (as long as they're within the radius of the circle). Along either of those circles there are two possible paths that you can take to connect the points, so in any situation there are four possible arcs available. Because of that, arcs have to take in quite a few arguments:

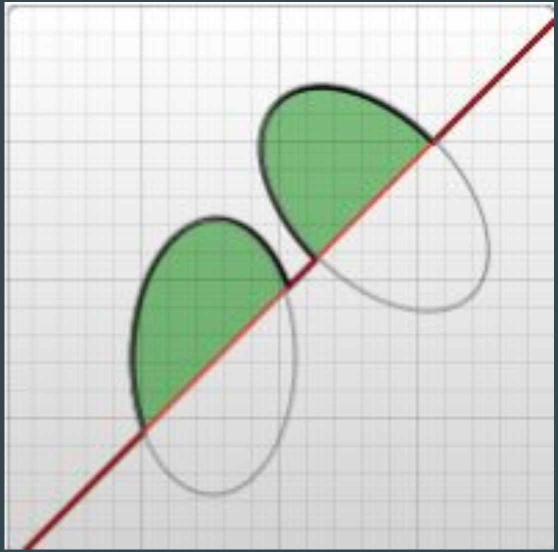


```
A rx ry x-axis-rotation large-arc-flag sweep-flag x y
 a rx ry x-axis-rotation large-arc-flag sweep-flag dx dy
```

- At its start, the arc element takes in two arguments for the x-radius and y-radius.
- The final two arguments designate the x and y coordinates to end the stroke. Together, these four values define the basic structure of the arc.
- The third parameter describes the rotation of the arc. This is best explained with an example:

Arcs - JSFIDDLE (Two elliptical arcs) JSFIDDLE (4 paths)

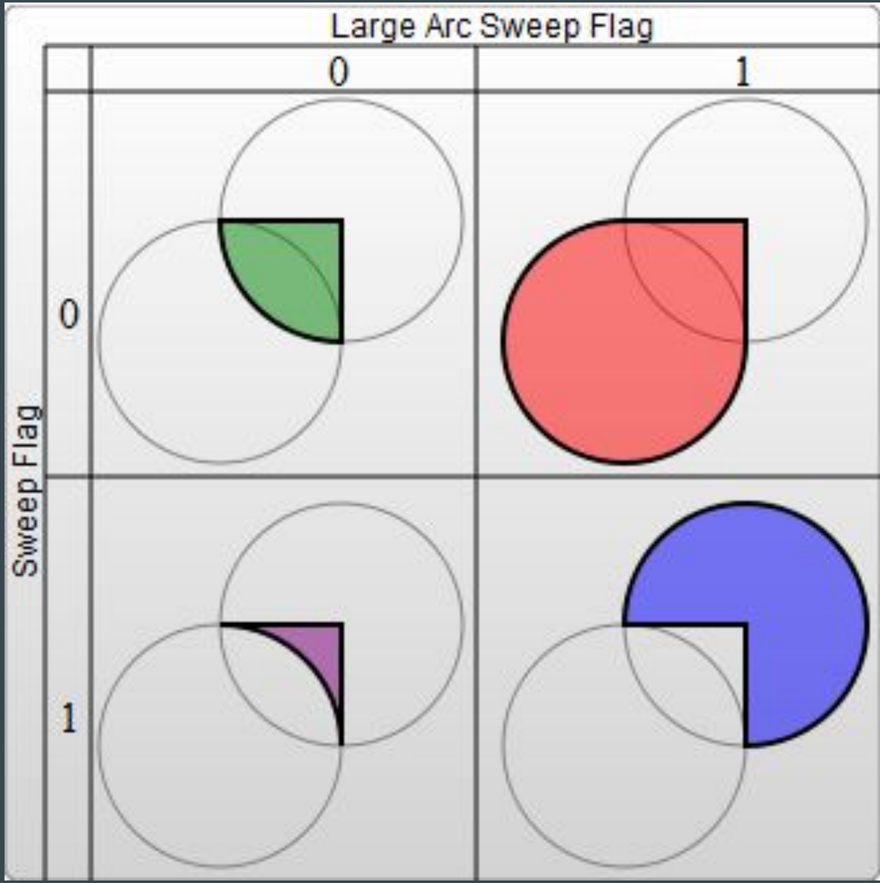
The example shows a path element that goes diagonally across the page. At its center, two elliptical arcs have been cut out (x radius = 30, y radius = 50). In the first one, the x-axis-rotation has been left at 0, so the ellipse that the arc travels around (shown in gray) is oriented straight up and down. For the second arc, though, the x-axis-rotation is set to -45 degrees. This rotates the ellipse so that it is aligned with its minor axis along the path direction, as shown by the second ellipse in the example image.



The four different paths mentioned are determined by the next two argument flags. As mentioned earlier, there are still two possible ellipses for the path to travel around and two different possible paths on both ellipses, giving four possible paths.

The first argument is the large-arc-flag. It simply determines if the arc should be greater than or less than 180 degrees; in the end, this flag determines which direction the arc will travel around a given circle.

The second argument is the sweep-flag. It determines if the arc should begin moving at positive angles or negative ones, which essentially picks which of the two circles you will travel around. The example below shows all four possible combinations, along with the two circles for each case.



SVG Continued

There is loads more to mastering SVG but we're out of time, I highly recommend you work through the following on your own:

- Fills and Strokes
- Gradients in SVG
- Patterns
- Texts
- Basic Transformations
- Clipping and masking
- Other content in SVG
- Filter effects
- SVG fonts
- SVG image element
- Tools for SVG

References & Tools

Inkscape is a free tool which allows you to create SVG graphics

SVG is a huge specification. We attempted to cover the basics. Once you are familiar you should be able to use the Element Reference and the Interface Reference to find out anything else you need to know.

Lecture slides 1 - 32 based entirely on content taken from Mozilla Developer Network SVG Tutorial

That will do for now...

Remember - No lectures next week
- Next Wednesdays lab is taking place!
- Next Thursdays lab will not take place!