Drawing with SVG: Lines, Shapes, paths, groups, def and use

Since SVG is a drawing format, it makes sense to cover drawing commands and ways to reuse content. Also note that the elements are not in the order I will use them.

The container

<svg> Wraps and defines the entire graphic. <svg> is to a scalable vector graphic what the <html> element is to a web page.

The example below is the minimum SVG example:

```
<svg  viewbox="0 0 100 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <!-- content goes here -->
</svg>
```

I've chosen to use CSS for the height and width of the SVG images and to explicitly list the default name space for SVG along with the name space for XLink, which we'll use to create links inside the SVG element.

Drawing Primitives

Now that we've refreshed our minds about the SVG element and ViewBox attribute let's draw.

Line

The simplest drawing primitive is `line`. It will draw a single straight line between too coordinates, defined by the pairs `x1 y1` and `x2 y2` and a `stroke` that represents the line's color. These are all required attributes

```
<svg  viewbox="0 0 100 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">

  <line   x1="10" y1="10
          x2="90" y2="50"
          stroke="rebeccapurple">
</svg>
```

Polyline

<polyline creates straight lines connecting several points; usually creating open shapes as the last point doesn't have to be connected to the first point.

The points attribute is mandatory and represents the coordinates for the different coordinates for the line.

`fill` represents the color inside the lines and `stroke` represents the color of the lines themselves.

```
<svg  viewbox="0 0 200 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- Example of the same polyline shape with stroke and no fill -->
  <polyline points="0,
                    25 150,
                    25 150,
                    75 200,
                    0"
            fill="blue" stroke="black" />
</svg>
```

`polyline` is different than a poligon, discussed later.

Rect

`rect` allows us to create rectanles and squares in SVG. We can control whether the

element has rounded corners and both the line (stroke) color and the color of the inside of the element.

```
<svg  viewBox="0 0 220 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <!-- Simple rect element -->
  <rect x="10" y="10" width="50" height="50" />

  <rect x="10" y="10" width="50" height="50" /><!-- Rounded corner rect e
  <rect x="120" y="10" width="50" height="50" rx="15" ry="15" />
</svg>
```

It takes four required attributes and, at least, four optional ones. The required attributes are:

- x and y are the coordinates for the rectangle's origin
- `width` and `height` are the rectangle's dimensions

The optional attributes I use the most are:

- `rx` and `ry` control the size of the rounded corners. Combined they are similar to CSS' border radius property
- `stroke` controls the color of the line drawing the element
- `fill` controls the color inside of the element drawn with the stroke color

If you use `stroke` and `fill` you must use both. At least in Chrome, using one of these will render the element invisible.

Circle

Circles are self explainatory. They draw circles of the given radius at the specified coordinates.

```
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
  <circle cx="25" cy="22" r="20"
    fill="navy" stroke="transparent" style="opacity: 0.25"/>
</svg>
```

- `cx` and `cy` indicate the position of the ellipse
- `r` indicates the radius of the circle

Ellipse

The ellipse element is what SVG uses to draw ellipses based on a center coordinate, and both their x and y radius.

```
<svg  viewBox="0 0 200 100"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <ellipse cx="100" cy="50" rx="100" ry="50" />
</svg>
```

The required attributes are:

- `cx` and `cy` indicate the position of the ellipse
- `rx` and `ry` represents the individual radius (X and Y) of the ellipse

Optional attributes

- `stroke` controls the color of the line drawing the element
- `fill` controls the color inside of the element drawn with the stroke color

If you use `stroke` and `fill` you must use both. At least in Chrome, using one of these will render the element invisible.

Polygon

Here's where things get interesting. Polygons allow us to create multiline closed shape consisting of a set of connected straight line segments... think of all the faces of each shape of your RPG dice. The last point is connected to the first point unlike the `polyline` element where the shapes are open.

```
<svg  viewBox="0 0 100 100"
      xmlns="http://www.w3.org/2000/svg">
  <polygon points="0 20, 40 20, 20 0"
           stroke="black" fill="purple">
```

```
  </polygon>
</svg>
```

The polygon has only one require attribute. `points` defines at least three set of coordinates for the lines we want to draw.

Path

This is the heavy duty drawing element in SVG. I will cover the basics here and will refer you to MDN articles on the path element and the d attribute.

Where all the elements we've discussed so far allow you to draw discrete elements, the path allows you to create arbitrary elements using movement and drawing primitives exclusive to the path element.

SVG defines 6 types of path commands, for a total of 20 commands:

- MoveTo: `M`, `m`
- LineTo: `L`, `l`, `H`, h, `V`, v
- Cubic Bézier Curve: `C`, `c`, `S`, `s`
- Quadratic Bézier Curve: `Q`, `q`, `T`, `t`
- Elliptical Arc Curve: `A`, a
- ClosePath: `Z`, `z`

Note that the commands are case sensitive and that the upper case `M` command means something diffrent than lower case `m`. See the reference for the d command on MDN for a full reference.

In the example below we perform the following tasks:

```
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
  <path d=" M 10,30
            A 20,20 0,0,1 50,30
            A 20,20 0,0,1 90,30
            Q 90,60 50,90
            Q 10,60 10,30 z"
      fill="red" stroke="transparent"
    />
```

```
</svg>
```

In this example we move the drawing element to a set of coordinates, we draw two arcs and we draw two elements using quadratic bezier curves before we close the path to make a full shape.

Conclusion

I know that doing this by hand can be too much. Fortunately, tools like Adobe Illustrator from Adobe and Inkscape allow you to export illustrations as SVG to use on the web.