4b: SVG Basics

Objective

We'll start by looking at the fundamentals of how graphics are drawn in the browser.

This is to build a foundation such that we can manipulate them using D3.js later in the module.

We will start by looking at SVG tags, see how to manipulate them using text attributes, look at CSS stylings, and then put it all together by opening up and manipulating a SVG (icon, chart, etc.)

If we have time we will also learn to do some light animation using CSS keyframe animations.

Drawing for web

- **SVG**: Has a DOM-like structure, easy to register events for interaction, scale invariant, can be themed with CSS, and easy to manipulate.
- **Canvas (2D)**: Is low level and event firing is far trickier. However, canvas is useful for code art work, when you need to draw lots of shapes on the screen and animate quickly.
- **HTML elements**: Elements like divs and imgs can be used, but more painful to manipulate.
- **WebGL (3D)**: Use libraries like three.js or babylon.js rather than native as they have excellent design patterns (camera, lighting, etc.)

When in doubt, go SVG for all the reasons listed above. It has great support by modern browsers (post IE9)

SVG specifications

SVG specifications from W3C are here.

Like HTML and CSS, there are many, many tags. I will only go through some of the more common basic tags. Google when you need it!

SVG

Let's start with a simple HTML template with an empty SVG tag embedded in it.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <svg></svg>
</body>
</html>
```

SVG 2

Inspect the SVG tag. You can see it under the Chrome inspector. Change the width and height so that it is easier to see.

```
<svg width="1000" height="1000"></svg>
```

Technically you should add in the XML namespace attribute, e.g. xmlns="http://www.w3.org/2000/svg". Required for standalone images, but for inline HTML, browsers will ignore.

Circle

Draw a circle

```
<circle cx="500" cy="500" r="300"></circle>
```

Add in fill and stroke attributes, e.g: stroke="red" and fill="blue"

Make stroke-width more visible, e.g: stroke-width="5"

If not defined, the units are in px. You can also use em.

Rectangles

Rectangle syntax

```
<rect x="700" y="700" width="100" height="100" fill="#aa7755"></rect>
```

Can you create an empty (not filled) rectangle, and the stroke is by dashed lines? Google the syntax for this.

Lines

Line syntax

```
<line x1="50" y1="50 x2="900" y2="900" stroke="#0f0"></line>
```

CSS styling

SVG can be styled using using CSS (it is DOM-like). For example, add the CSS class tag:

```
<style>
rect {
  fill: #777;
}
</style>
```

What happens when you have an attribute and style tag for that element?

Because elements can be styled by CSS, that means they follows CSS rules. CSS style selectors like **id** and **class** can be used. Try it.

You can move many attributes to the CSS, but it only makes sense as a design pattern to only move out styling attributes.

Grouping

You can group elements together using the <g> tag. All attributes then also apply to the child tags. Example:

```
<g fill="#0f0" stroke="#f00">
<circle cx="500" cy="500" r="300"></circle>
<rect class="rect" x="700" y="100" width="100" height="100"></rect>
</g>
```

Recap: In CSS child tags inherit styles from the parent tag.

Cascade and specificity rules of course also apply.

Exercise 1

1. Create 2 circles, 2 rectangles and 2 lines anywhere on the SVG canvas.
2. Group the circles together. Also group the rectangles together, and then group all the groups.
3. Then use CSS to style the group elements so that it is inherited by child elements (fill, stroke, stroke-width, etc.).
4. Show that group styles do not overide elements inside the groups when these elements have their own specific styles (specificity).

SVG has no z-index property. Whatever is drawn the latest will be drawn on top of the last tag.

Path

Path syntax

```
<path d="M100,200 L300,600 L500,200Z"></path>
```

This is a simple example to draw shapes using the path tag. There are lots of other commands in the path tag

The Z tag closes the path. Try it with and without and see the difference.

The commas can be just spaces. But this accepted syntax makes it easier for humans to understand the code. Default is spaces.

Path 2

Many of the lines you will see on a visualization are going to be paths. You can draw quadratic/bezier curves, or portions of arcs. E.g:

```
<path d="M10 500 C 900 100, 100 900, 990 500" stroke="#000" fill="none"></path>
```

Can you draw in the control path lines for the above so we can see more clearly the control points?

There are many commands to draw shapes in SVGs, and you probably only need to Google for the command if you have a specific use case.

Still, it is useful to have some idea of how paths are built.

Exercise 2

Draw a 5 pointed star or hexagon.

Text

Text is part and parcel of the SVG specification. Syntax:

```
<text x="100" y="100">Hello World</text>
```

What happens when you try to style with both fill and stroke? Try it. Also try setting a thicker stroke width.

Can you style the font-family, font-style? Use CSS.

Text 2

SVG text has a lot of properties you can play with, e.g. text-anchor, tspan (text breaks), textpath, etc. Example:

```
<path id="helloworld" d="M10 500 C 900 100, 100 900, 990 500" stroke="#000" fill="none"></path>

  <text font-size="60px">
    <textPath xlink:href="#helloworld">
        Hello World. Ipsum Lorem dolor sit amet.
    </textPath>
  </text>
```

Transforms

Transform is very, very useful to know, as it help you position elements (especially when grouped) correctly.

```
<g transform="translate(200, 200)">
<circle cx="500" cy="500" r="100"></circle>
<rect x="700" y="100" width="100" height="100"></rect>
</g>
```

Translate, scale, rotate are probably some of the more common transforms you will probably use. You can also try out others like skew and matrix.

Try setting the transform-origin attribute to "50% 50%". Otherwise by default it will transform from the top-left corner (i.e. "0% 0%").

ViewBox

Another extremely useful SVG property. Auto resizes chart to a specific viewBox view port. Example:

```
<svg viewBox="0 0 1000 1000"></svg>
```

How is this different from setting SVG height and width?

Exercise 3

1. Download a SVG icon from anywhere (e.g. flaticon)
2. Open up the file in a text-editor, strip out erroneous tags, and add the code to an inline SVG.
3. Rotate the entire icon 45 degrees in your SVG.
4. Feel free to tranfrom different elements any way you want to.

Extra: CSS keyframe animation

Add the below animate class to a path or SVG element and add the code snippet to the CSS.

```
.animate {
  stroke-dasharray: 1000;
  stroke-dashoffset: 1000;
  animation: dash 5s linear forwards;
}

@keyframes dash {
  to {
    stroke-dashoffset: 0;
  }
}
```

Questions?

Chi-Loong | V/R