SVG Animation

Perhaps the most interesting and complex topic in SVG, for me, is animations. I have a basic understanding on how to create keyframe animations for CSS content and a very basic understanding of how to use GSAP to animate HTML elements. I want to explore how to create smaller targeted animations using both CSS and SVG specific animations techniques.

SMIL and the animate element

SVG comes with its own animation tools in the form of the animate element and SMIL.

The most basic animation element is `animate`. It takes the following attributes:

- **attributeName**: The name of the attribute we want to animate
- **attributeType**: One of CSS, XML or auto
  - CSS specifies that the value of attributeName is the name of a CSS property defined as animatable
  - **XML** represents an XML attribute defined as animatable in the SVG namespace
  - **auto** tells the browser (in this case) to first search through the list of CSS properties for a matching property name, and if it doesn't find one, then search the default XML namespace for the element
- **from** is the initial value of the property we want to animate
- **to** is the final value of the property we want to animate
- **dur** tells the browser how long we want the animation to last
- **repeatCount** tells the user agent how many times we want to run the animation. It can be a positive integer or the keyword `indefinite`

The example below will animate the movement of the rectangle on the x axis from left to right, lasting 10 seconds and repeating the animation from the beginning 10 times.

```
<svg width="100" height="100" viewBox="0 0 100 100" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
```

```
  <rect x="10" y="10"
        width="90" height="90"
        fill="purple" >
    <animate attributeType="XML"
             attributeName="x"
             from="100" to="-100"
             dur="10s"
             repeatCount="10"/>
  </rect>
</svg>
```

The second example builds on the first one. It adds additional `animate` elements to animate different properties of the same target element. The second `animate` element will move the rectangle on the y axis, and the third `animate` element will animate the color from purple to blue.

If we wanted to we could give these properties their own duration or repeatCount values but, for this example,

```
<svg width="100" height="100" viewBox="0 0 100 100" version="1.1"
     xmlns="http://www.w3.org/2000/svg">

  <rect x="10" y="10"
        width="90" height="90"
        fill="purple" >
    <animate attributeType="XML"
             attributeName="x" from="100" to="-100"
             dur="10s"
             repeatCount="10"/>
    <animate attributeType="XML"
             attributeName="y" from="100" to="-100"
             dur="10s"
             repeatCount="10"/>
    <animate attributeType="XML"
             attributeName="fill" from="purple" to="blue"
             dur="10s"
             repeatCount="10" />
```

```
  </rect>
</svg>
```

animateMotion

The second element is `animateMotion` and it animates the target element along a pre-defined path.

- **calcMode** indicates what interpolation mode to use for animation. It takes one of the following values:
    - **discrete**: The animation function will jump from one value to the next without any interpolation
    - **linear**: Simple linear interpolation between values is used to calculate the animation function
    - **paced**: Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). For `animateMotion`, this is the default value
    - **spline**: Interpolates from one value in the values list to the next according to a time function defined by a cubic Bézier spline. The points of the spline are defined in the `keyTimes` attribute, and the control points for each interval are defined in the `keySplines` attribute
- `path` represent the path the animation will follow
- `keyPoints` takes a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the object shall move at the moment in time specified by corresponding 'keyTimes' value. Each value in the list corresponds to a value in the `keyTimes` attribute list. The number of `keyPoint` values must exactly match the values in the 'keyPoints' list
- **rotate** controls the positioning of the animated object. It can take one of the following values:
    - **auto**: The object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path
    - **auto-reverse** The object is rotated over time by the angle of the direction (i.e., directional tangent vector) of the motion path plus 180 degrees.
    - **number** The target element has a constant rotation transformation

applied to it, where the rotation angle is the specified number of degrees.
    - The default value is 0.
- **origin** is defined in the SMIL Animation Specification and has no effect on SVG

The `mpath` sub-element provides the ability to reference an external `path` element as the definition of a motion path.

1. The example creates an ellipses to be used as the path our object will navigate.
2. Then we create three circles and place them on the path, one at the beginning, one on the middle and one on the end.
3. Finally we create a triangle and animate it along the path we defined in step 1

```
<svg  width="5cm" height="3cm"
      viewBox="0 0 500 300"
      xmlns="http://www.w3.org/2000/svg">

  <path id="path1"
        d="M100,250 C 100,50 400,50 400,250"
        fill="none"
        stroke="bl<path id="path1"
        d="M100,250 C 100,50 400,50 400,250"
        fill="none"
        stroke="blue"
        stroke-width="7.06"  />lue"  />
  <circle cx="400" cy="250" r="12.64" fill="blue"  />

  <path id="triangle"
        d="M-25,-12.5 L25,-12.5 L 0,-87.5 z"
        fill="yellow"
        stroke="red"
        stroke-width="7.06"  >
    <circle cx="400" cy="250" r="12.64" fill="blue"  /><!-- Define the mot
    <animateMotion  dur="6s"
                    repeatCount="10"
                    rotate="auto" >
```

```
      <mpath href="#path1"/>
    </animateMotion>
  </path>
</svg>
```

animateTransform

The `animateTransform` element control transform-based animations and does a similar job to the CSS `transform`. Because it handles different types of transformations there is no single set of paramepters to discuss here. I'll refer you to the element's description in the SVG Animations specification.

```
<svg width="400" height="600" viewbox="0 0 100 100">
  <rect id="Rectangle1" fill="#3C81C1" x="0" y="0" width="100" height="125
    <animateTransform   attributeName="transform"
                        type="translate"
                        from="0 0"
                        to="150 20"
                        begin="0s"
                        dur="2s"
                        repeatCount="indefinite"
    />
  </rect>
</svg>
```

Multiple animations

One important detail is how to run multiple `animateTransform` elements for the same target.

We need to wrap all the SVG children in a g element and then, like in the example, define one inside the target element and one as the last child of the g grouping element.

```
<svg width="600" height="400" viewbox="0 0 600 400">
  <g>
```

```
    <rect id="Rectangle1"
          fill="#3C81C1"
          x="0" y="0"
          width="100" height="125">
        <animateTransform   attributeName="transform"
                            type="scale"
                            from="1 1"
                            to="3 1.25"
                            begin="0s"
                            dur="2s"
                            repeatCount="5"
        />
    </rect>
    <animateTransform   attributeName="transform"
                        type="translate"
                        from="0 0"
                        to="150 20"
                        begin="0s"
                        dur="2s"
                        repeatCount="5" />
  </g>
</svg>
```

Set

The ‘set’ element provides a simple means of just setting the value of an attribute for a specified duration. It supports all attribute types, including those that cannot reasonably be interpolated, such as string and boolean values.

In the example we set the `stroke-width` attribute, which takes a string to be 5px starting 5 seconds into the animation.

```
<svg width="100" height="100" viewBox="0 0 100 100" version="1.1"
     xmlns="http://www.w3.org/2000/svg">

  <rect x="10" y="10"
        width="90" height="90"
```

```
        fill="blue"
        stroke="red">
    <animate attributeType="XML"
             attributeName="x"
             from="100" to="-100"
             dur="10s"
             repeatCount="10"/>
    <set  attributeName="stroke-width" to="10px"
          begin="5s"
          dur="10s"/>
  </rect>
</svg>
```

Discard

The ‘discard’ element allows authors to specify the time at which particular elements are to be discarded, thereby reducing the resources required by an SVG user agent. This is particularly useful to help SVG viewers conserve memory while displaying long-running documents. This element will not be processed by static SVG viewers.

The ‘discard’ element may occur wherever the ‘animate’ element may.

```
<svg xmlns="http://www.w3.org/2000/svg" width="352" height="240" playback

  <ellipse cx="98.5" cy="17.5"
           rx="20.5" ry="17.5"
           fill="blue" stroke="black"
           transform="translate(9 252) translate(3 -296)">
    <animateTransform attributeName="transform"
                      begin="0s" dur="2s"
                      fill="remove"
                      calcMode="linear"
                      type="translate"
                      additive="sum"
                      from="0 0" to="-18 305"/>
    <discard begin="2s"/>
```

```
  </ellipse>

  <rect x="182" y="-39" width="39" height="30"
        fill="red" stroke="black"
        transform="translate(30 301)">
    <animateTransform   attributeName="transform"
                        begin="1s" dur="2s"
                        fill="remove"
                        calcMode="linear"
                        type="translate"
                        additive="sum"
                        from="0 0" to="-26 -304"/>
    <discard begin="3s"/>
  </rect>

  <polygon points="-66,83.5814 -43,123.419 -89,123.419"
           fill="green" stroke="black"
           transform="matrix(1 0 0 1.1798 0 -18.6096)">
    <animateTransform attributeName="transform"
                      begin="2s" dur="2s"
                      fill="remove" calcMode="linear" type="translate" ad
                      from="0 0" to="460 63.5699"/>
    <discard begin="4s"/>
  </polygon>
</svg>
```

The downside is that SMIL, the technology behind SVG native animations is not in IE/Edge and never will be; it has been deprecated by Blink (but the deprecation was withdrawn and will not happen… for now). Check caniuse.com for SMIL support and how it evolves over time.

> There are other options to animate SVG. We'll explore them

CSS Animations

Interestingly, most of what you can do natively in SVG you can do with CSS transitions and transformations. We'll work with CSS inside the SVG element but it could easily be done with an external style sheet that integrates all the styles for

the document.

The example shows how using the `style` element inside an inline SVG we can control SVG animations the same way we control them in CSS.

In this example we define the styles for the `.rotator` element and the keyframes we need for the animation to work.

The rectangle in the SVG matches the class in the CSS so it gets animated and rotates.

```
<svg  xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="https://www.w3.org/1999/xlink"
      width="600" height="450"
      viewbox="0 0 100 100">

  <style><style>
    .rotator {
      fill: deeppink;
      animation: hideshow 5s ease-in 10;
      transform-origin: center;
    }

    @keyframes hideshow {
       50% { fill: purple;}
        to {
          transform: rotate(360deg);
          fill: blue;
      }
    }
  </style>


  <rect class="rotator"
        x="50" y="50"
        width="25" height="25"></rect>
</svg>
```

Web Animations API: One API to rule them all

The last animation technique I want to discuss is the Web Animations API; an attempt to unify all the animations APIs available in CSS and SVG. Yes, it means adding a script to the page and, potentially blocking render if we're not careful but I think it's the best way to use animations short of going the GSAP route.

I wrote about Web Animations API in 2016 but it wasn't until I started looking at SVG that I came back to the API as a one-stop way to animate content.

This example is broken into two sections. The first one is the SVG rectangle where we assign an ID to the element we want to rotate so we can grab it later from the script.

```
<svg  xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="https://www.w3.org/1999/xlink"
      width="600" height="450"
      viewbox="0 0 100 100">
  <rect id="rotator"
        x="50" y="50"
        fill="deeppink"
        width="50"
```

```
        height="50"></rect>
</svg>
```

This is the first, and only, example that will use Javascript. Web Animation APIs main method is `animate`; it takes two parameters, an array of objects containing the different animations to run and another object containing the settings for the animation.

```
const player = document.getElementById('toAnimate').animate([
    { transform: 'scale(1)', opacity: 1, offset: 0 },
    { transfo'scale(1)'(.5)', opacity: .5, offset: .3 },
    { transform: 'scale(.667)', opacity: .667 },
    { transform: 'scale(.6)', opacity: .6 }
  ], {
    'scale(.667)'// duration in milliseconds: 10000,
    //'linear', a'linear'//'linear', a bezier curve, etc.
    easing: 'ease-in-out',
    // delay in milliseconds
    delay: 10,
    // infinity or a positive integer numbernity,
    //'normal', 'reverse''normal'//'normal', 'reverse', etc.    //'backwa
    fill: 'forwards'
  });
```

Yeah, much more verbose, but it’ll work everywhere and it doesn’t require a third party library.