SVG Text

In a previous post we saw how to make text follow a path. Now we'll look at basic text and elements that will make interacting with the text easier. We'll also look at how to use Web Fonts with SVG. This is not a replacement for HTML, but an alternative for doing things that are not possible, or not easily possible with HTML and CSS alone.

Text

The `text` element allows you to write text as part of your SVG content. It interact well with CSS, either written as a style inside the SVG element or in an external stylesheet.

The example below uses an internal stylesheet to define three clases to control diffeerent aspect of the text.

Next, we define a group of text elements and give it an ID of `catMessage`. Each text block has a class matching the ones we defined in CSS.

Lastly we use the `catMessage` group. Because we associated each text block to a CSS class we don't need to do any further styling.

```
<svg viewBox="0 0 240 80"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink"https://www.w3.org/1999/xlink">
  <style>
    .small {
      font: italic 0.75em sans-serif; fill: blue
    }
    .heavy {
      font: bold 2em sans-serif; fill: rebeccapurple
    }
    .Rrrrr {
      font: italic 3em serif; fill: red;
    }
  </style>
```

```
  <defs>
    <g id="catMessage">
      <text x="20" y="35" class="small">My</text>
      <text x="40" y="35" class="heavy">cat</text>
      <text x="55" y="55" class="small">is</text>
      <text x="65" y="60" class="Rrrrr">Grumpy!</text>
    </g>
  </defs>

  <use xlink:href="#catMessage"/>
</svg>
```

The example adds `class` to tell the parser what CSS class we want to use for that particular element.

text

The `text` element renders text in the SVG document. This is the only way to render text in SVG. If there is any text in the SVG document that is outside a `text` element, it won't render.

x and y are the coordinates where, in the viewport, we want to place the text block.

tspan

`tspan` allows for changes in only some parts of the text element without changing the complete line of text. It is similar to the HTML span element

```
<svg viewBox="0 0 240 80"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink"https://www.w3.org/1999/xlink">
  <style>
    .small {font: italic 0.75em sans-serif; fill: blue}
    .heavy {font: bold 2em sans-serif; fill: rebeccapurple}
    .Rrrrr {font: italic 3em serif; fill: red;}
```

```
  </style>

  <defs>
    <g id="catMessage">
      <text x="20" y="35" class="small">My</text>
      <text x="40" y="35" class="heavy">cat</text>
      <text x="55" y="55" class="small">is</text>
      <text x="65" y="60" class="Rrrrr">Gru<tspan style="font-size: 150%;
    </g>
  </defs>

  <use xlink:href="#catMessage"/>
</svg>
```

Loading fonts via CSS

Rather than depending on the system fonts, we can load fonts via CSS' `font-face` attribute just like we do for our regular web pages.

The first version of this example uses Marvin Visions as the font, loaded with `@font-face`.

This example uses a single `text` element for the content of the text.

The CSS loads the font and sets formatting attributes for the text. The attributes set are:

- Font family
- Font size
- The color of the inside of the font (the fill)
- Text shadow

```
<svg viewBox="0 0 240 360"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink"https://www.w3.org/1999/xlink">
  <style>
    @font-face {
```

```
      font-family: "Marvin";
     "Marvin"url("MarvinVisionsBig-Bold.woff")ormat("woff");
    }

    te"woff"text {
      font-family: Marvin;
      font-size: 3em;
      fill: #e65f1d;
      text-shadow: 12px 12px 6px rgba(0, 0, 0, 1);
    }
  </style>

  <defs>
    <g id="title">
      <text x="10" y="40">Nightfall</text>
    </g>
  </defs>

  <use xlink:href="#title"/>
</svg>
```

A logical, and cumbersome, next step would be to add `tspan` elements surrounding each letter so we can style them individually, similar to an exercise in the CSS: Advanced Typographic Techniques course from LinkedIn Learning.

The idea is that, by assigning a `tspan` with an ID attribute to each letter we can style them differently from each other and it works, sort of.

Because we use a single `text` element as the container we must make changes to all the `tspan` children that need to move. In the example below, I moved the word fall into its own line by changing the coordinates of the first letter; all the following elements moved along.

If we want to play with x and y coordinates we need to keep in mind that if we make changes to one we must make changes to all the subsequent ones. In the example below we use this trick to move the second word to its own line by changing the coordinates for the first letter in that word.

Because we've assigned `id` attributes to each letter we can style them

individually outside the coordinates that we must set in the individual svg spans.

```
<svg viewBox="0 0 240 360"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink"https://www.w3.org/1999/xlink">
  <style>
    /* Load the font */
    @font-face {
      font-family: "Marvin";
      src: "Marvin"url("path/to/MarvinVisionsBig-Bold.woff")"woff");
    }

    text {
 "woff"text {
      font-family: Marvin;
      font-size: 3em;
      fill: #e65f1d;
      text-shadow: 12px 12px 6px rgba(0, 0, 0, 1);
    }

    /* Individual letters styles */
    #l6 {fill: rebeccapurple;}
    #l7 {fill: navy;}
    #l8 {fill: red;}
    #l9 {fill: limegreen;}
  </style>

  <defs>
  </style><!--
    tspan elements broken for readability.

    They should all be in a single line or browsers will
    insert weird spaces between the characters
  -->
    <g id="title">
      <text x="10" y="40"><tspan id="l1">N</tspan>
      <tspan id="l2">i</tspan>
      <tspan id="l3">g</tspan>
```

```
      <tspan id="l4">h<tspan>
      <tspan id="l5">t</tspan>
      <tspan id="l6" x="10" y="80">f</tspan>
      <tspan id="l7">a</tspan>
      <tspan id="l8">l</tspan>
      <tspan id="l9">l</tspan></text>
    </g>
  </defs>

  <use xlink:href="#title"/>
</svg>
```

Rotating text

CSS allows you to rotate text in different writing modes but it's not supported in all browsers. SVG provides an alternative way to rotate and display the text that will work in browsers that support SVG.

We apply the same attribute/value combination that we use in CSS: `writing-mode: vertical-lr`. There is a SVG specific attribute `writing-mode="tb"` that we can use, but support is not uniform across browser.

```
<svg viewBox="0 0 240 720"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink"https://www.w3.org/1999/xlink">
  <style>
    @font-face {
      font-family: "Marvin";
     "Marvin"url("MarvinVisionsBig-Bold.woff")ormat("woff");
    }

    te"woff"text {
      font-family: Marvin;
      font-size: 3em;
      fill: #e65f1d;
      text-shadow: 12px 12px 6px rgba(0, 0, 0, 1);
      writing-mode: vertical-lr;
```

```
    }
  </style>

  <defs>
    <g id="title">
      <text x="40" y="10">Nightfall</text>
    </g>
  </defs>

  <use xlink:href="#title"/>
</svg>
```

Text Path

`textPath` places text the shape of a path element defined alongside it.

For `textPath` to work, the element we use to wrap our content around must be a path. In earlier versions of this example I tried to use a `circle` but it doesn't work. Manually creating a circle with SVG is not easy, so I cheated. I created the circle in Adobe Illustrator and then exported it as SVG and copied it to Codepen.

`textLength` (essentially the circumference of the circle) is used as an alternative to letter-spacing for Firefox, which currently doesn't support letter-spacing for SVG

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/199
      viewBox="0 0 600 600">

  <defs>
    <path d="M276.14,178C395,178,491.29,259.54,491.29,360.12S395,542.24,27
  </defs>

  <text textLength="1200">
    <textPath xlink:href="#textcircle">
      The fellowship of the ring
    </textPath>
  </text>
```

```
</svg>
```

The CSS portion of this example is simple. It defines the dimensions of the svg element and font attributes.

In most production cases I would never import the font directly into the CSS file but since this is a demo I'm making an exception. For production, I would likely host the font on the same server and cache it using a service worker.

```
@import url('https://fonts.googleapis.com/css?family=Fredoka+One');

text {
  font-size: 38px;
    font-family: 'Fredoka One', cursive;
    font-weight: 900;
    text-transform: uppercase;
    l'Fredoka One': 24px;
    fill: rebeccapurple;
  opacity: 0.8;
}
```