SVG Masks and Clip Paths and Text on a path

One of the fun things to explore is how we can mask and clip SVG elements and how we can do “text on a path” effects. As we explore these new elements we’ll visit additional elements that make part of the mask and path functionality and how CSS interacts with SVG (TL,DR: It doesn’t use the same elements).

Masks

`mask` Defines an alpha mask for compositing the current object into the background. It will ‘bleed’ the background into the foreground element

The `pattern` element defines a graphics object which you can tile (repeat at x and y coordinate intervals) to cover an area. This is similar to what we can do with blend modes in CSS

The `patternUnits` attribute defines the coordinate system in use for x, y, `width` and `height` properties of the element.

The `image` SVG element includes images inside SVG documents. **It can display raster image files (JPG or PNG) or other SVG files. The behaviort for animated GIFs is undefined.**

The SVG `text` element defines a graphics element consisting of text. It’s possible to apply a gradient, pattern, clipping path, mask, or filter to `text`, just like any other SVG graphics element.

If text is included within SVG outside a `text` element, it is not rendered. **This is different from being hidden by default, as setting the display property will not show the text.**

The SVG portion of the example includes the image pattern and the text that we want to use as the mask.

```
<svg>
  <defs<defs>pattern  id="wood"
```

```
                patternUnits="userSpaceOnUse"
                width="400" height="400">
        <image
              <image
              xlink:href="http://subtlepatterns.com/patterns/purty_wood.png"
              width="400" height="400" />/defs>
  <!--
    The text below will have the background image of the pattern
  -->
  <text y="1.2em">SVG rocks!</text>
</svg>
```

The CSS sets the dimensions of the SVG element and the text fill, the color inside the element, to use the wood pattern we've defined in the SVG.

When we use CSS to style SVG elements we need to be aware that SVG doesn't use the same selectors, attributes, and values than regular CSS.

```
svg {
    width: 8em;
  height: 2em;
    font-weight: 900;
  font-size: 5em;
  line-height: 1.2;
  font-family: 'Ar'Arial Black'ans-serif;
}

text {
  fill: url(#wood);
}
```

Clip Path

`clipPath` defines a clipping path.

The idea is that whatever changes we make to the image will not go larger than the clipping path. In this example, we create to elements inside the SVG a

circle and a path with the shape of a heart.

We use the clipping path with the `clip-path` property.

```
<svg viewBox="0 0 100 100">
  <clipPath id="myClip">
    <circle cx="40" cy="35" r="35" />
  </clipPath>

  <path id="heart" d="M10,30 A20,20,0,0,1,50,30 A20,20,0,0,1,90,30 Q90,60

  <use clip-path="url(#myClip)" xlink:href="#heart" fill="red" />
</svg>
```

The CSS handles the animation using key frames. We're animating the `r` attribute from CSS. We want the circle to grow into the full size of the cliping element and ignore anything falling outside.

Unfortunately this example doesn't work in Firefox (tested with 67 nightly in a Mac) because it doesn't support geometry properties in CSS. Bug 1383650 is tracking the issue but it doesn't appear to be high priority.

The workaround is to remove the `clip-path="url(#myClip)"` from the use element . We loose the animation but at least we get the color we want to display the clipped element in.

I'm also researching if it's possible to do this with JavaScript or with CSS `@support` techniques.

```
html,body,svg {
  height:100%;
}

@keyframes animateHeart {
  from {
    r: 0
  }
```

```
  to {
    r: 60px
  }
}

#myClip circle {
  animation: animateHeart 15s infinite;
}
```

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 the 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');

body {
  margin: 0;
  text-align: end;
}
svg { width: 80%; }

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

Conclusion

SVG can be as simple or as complicated as you need it to be. There are other areas that I want to further explore as I move deeper into the things you can and cannot do with SVG as a vector graphic format.