SVG and CSS for cool effects with text

In recent years there has been a lot of new features that make magazine-style layouts possible and enticing for designers to use and for developers to implement.

Some of these features are quite old in terms of when they were introduced but it wasn't until I saw the work of Jenn Simmons and Andy Clarke that I started seeing the real possibilities of dooing design on the web.

Sure we're still missing fragmentation primitives for the web and the purists seem to have won, so far. Regions are dead and no alternatives have surfaced yet.

Clip-path

Clipping an image means that we hide parts of the images from view while showing others.

The following example will create a circle clipping area 100px in diameter around whatever image we add the class to.

```
.clipped {
  clip-path: circle(90px);
}
```

This is equivalent to the following code that you may see in some examples.

```
.clipped {
  clip-path: circle(90px at center center);
}
```

The second version of the code allows for more flexibility as we can move the center from which we generate the clipping path to create interesting variations on a theme.

For example, using portraits of presenters at a meetup, we can play with the direction of the clip to create more interesting clip effects.

Furthermore, we can define clip paths with SVG (and the full power it gives us), like the example below that will place circles in different places of the image.

```
<svg width="0" height="0">
  <defs>
    <clipPath id="bubbles">
      <circle cx="200" cy="100" r="40" />
      <circle cx="300" cy="60" r="40" />
      <circle cx="490" cy="50" r="40" />
      <circle cx="380" cy="140" r="40" />
      <circle cx="220" cy="220" r="40" />
    </clipPath>
  </defs>
</svg>
```

And reference the `clipPath` SVG element from CSS like so:

```
.clipped {
  clip-path: url('#bubbles');
}
```

An easy way to create clip path values for clipping is to use Bennet Feely's Clippy.

Mask

Masking is, from my perspective, far more complex than clipping. According to Sarah Drasner's Masking vs. Clipping: When to Use Each:

> Think about masking as a way to apply complex, detailed, and shapes with varying opacity over another element. This can lead to really beautiful visual effects and performant alternatives to other techniques.

And according to Chris Collier's Practical SVG:

> Clipping and masking are related concepts because they are both capable of hiding parts of an image. But the distinction between them can be confusing, so let's clear that up right now:
>
> Clipping is created from vector paths. Anything outside the path is hidden; anything inside is shown.
>
> Masking is created from images. Black parts of the image mask hide; white parts show through. Shades of gray force partial transparency—imagine a black-to-white gradient over an image that "fades out" the image.

Using SVG makes it easier to create masks that we can use in an HTML document.

The first SVG document creates the definitions of the elements we want to use inside a defs to hold templates for the things we want to use inside symbols

```
<div class="wrapper">
  <svg width="200" height="300">
    <defs>
      <symbol id="s-mask-circles">
        <g stroke="lightgrey" stroke-width="10" fill="white">
          <circle cx="125px" cy="30%" r="20%" />
          <circle cx="52%" cy="62%" r="32%" />
        </g>
      </symbol>

      <symbol id="illiad">
        <image xlink:href="images/Illiad1-mod.jpg" width="398" height="300
      </symbol>

      <symbol id="coffee">
        <image xlink:href="images/coffee2.png" width="199" height="375" />
      </symbol>

      <mask id="mask-circles">
        <use xlink:href="#s-mask-circles" />
      </mask>
```

```
    </defs>

  </svg>
```

We can then use additional SVG elements to create the masks. The examples below uses the elements we defined earlier in the `mask` and `use` elements.

```
<svg width="200" height="300">
  <g mask="url(#mask-circles)">
    <use xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#illiad">
  </g>
</svg>

<svg width="200" height="300">
  <g mask="url(#mask-circles)">
    <use xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#coffee">
  </g>
</svg>
```

This works with SVG elements and the flexibility they give you is amazing. But it doesn't work with CSS directly.

> Using masks in CSS is a two step process.

1. Create the mask image using a tool like Photoshop or GIMP
2. Use the mask image as the attribute for the `mask-image` attribute
3. Use the `mask-type` attribute to indicate what type of mask you want to use

In the HTML document we provide the background image for our experiment.

```
<img src="images/chile-01.jpg">
```

The CSS does a few things:

1. It says the image to `display-block` just to make sure in case we set it to something else in the stylesheet
2. Set `mask-image` to the image that we want to set as the mask
3. Position the mask image using `mask-position`

4. Choose if the we want to repeat the mask image using `mask-repeat`

```
img {
  margin: 20vh auto;
  display: block; /* 1 */
  width: 50%;
  height: 400px;
  -webkit-mask-image:
    url(images/mask-image.png); /* 2 */
  mask-image:
    url(images/mask-image.png); /* 2 */
  -webkit-mask-position: center bottom; /* 3 */
  mask-position: center bottom; /* 3 */
  -webkit-mask-repeat: no-repeat; /* 4 */
  mask-repeat: no-repeat; /* 4 */
}
```

Right now we have to duplicate code foor the mask attributes because Blink supports only prefixed versions of the attributes but we still want to future proof our code. Tools like Autoprefixer or Prefix Free will keep our code DRY.

In the video CJ Gammon, from Adobe, presents a deep dive on masking, which may be easier to process.

Motion on a path

Motion on a path is interesting. It allows you to create animations that follow a set path (defined using SVG path elements).

> The three aspects of this code that are important:

1. offset-path sets the path we want the animation too navigate
2. The animation
3. We create the steps of the animatioon using the @keyframes at-rule. Make sure the name matches the one we used in the animation rule

```
#motion-demo {
  offset-path: path('M6,150C49.6'M6,150C49.63,93,105.79,36.65,156.2,47.55
  animation: move 4000ms infinite alternate ease-in-out; /* 2 */
  width: 40px;
  height: 40px;
  background: rebeccapurple;
}

@keyframes move { /* 3 */
  0% {
    offset-distance: 0%;
  }
  100% {
    offset-distance: 100%;
  }
}
```

Text on a path

We can also do text on a path using SVG paths, text, and textPath elements.

```
<svg viewBox="0 0 425 300">
  <path id="curve"
      d="M6,150C49.63,93,105.79,36.65,156.2,47.55,207.89,58.74,213,131.91
```

```
  <text x="25">
    <textPath xlink:href="#curve">
      Dangerous Curves Ahead
    </textPath>
  </text>
</svg>
```

There are ways to animate the text as it moves on the path we've set up. Sadly it's an all-SVG solution right now.

```
<svg id="canvas" xmlns="http://www.w3.org/2000/svg" version="1.1" width="8
  <defs>
    <path id="curve" d="
          M 400, 200
          a100,100 0 1,1 0,-.00001 z"
          fill="none"
          stroke="tr<defs>ent"
          stroke-width="1"></path> </path><!-- 2 -->fs>
  <text dx="50"
        font-family="Verdana"
        font-size="18"
        fill="rebeccapurple"> <!-- 3 <text dx="50"
        font-family="Verdana"
        font-size="18"
        fill="rebeccapurple"><!-- 3 -->tPath> <!-- 4 -->
      <animate
        attributeName="dx"
        type="xml"
        dur="12s"
        from=<!-- 4 --><!-- 4 -->"
        repeatCount="indefinite" /> <!-- 5 -->
  </text>
</svg>
<!-- 5 --><!-- 5 -->
  </text>
</svg>
```

The example does the following:

1. Set up the SVG documents with the correct namespaces and
2. Under `defs` we set up a `path` that the text will navigate on
3. Set up a text element containing the text path and the associated animation
4. Set up the content for the text we want to animate. The link to the path defined in step 2 associates the text with the path that
5. We use a SMIL animate tag to generate the animation

This last bit is important. I'm not 100% sure if the `animate` element uses SMIL or not but either waay we should move it to CSS or Web Animation API to make sure that it will continue too work after SMIL is removed from browsers.

Chris Coyier's CSS Tricks Screencast "Three Ways to Animate SVG" covers different ways to do animated SVGs

If nothing else works, you can always pull out the big guns and use animation libraries like GSAP to animate your SVG. See Sarah Drasner's SVG Animations for more information.

Blend Modes

One of the first things that attracted me to typography and the expressive potential of the web was seeing text with a background image bleeding through inside the shape of the letters

I later learned that technique is called blend modes and the one in particular I was interested in is called mix-blend-mode and there are 16 different posibilities:

- color
- color-burn
- color-dodge
- darken
- difference
- exclusion
- hard-light
- hue
- lighten
- luminosity
- multiply
- normal
- overlay
- saturation
- screen
- soft-light

See MDN's Blend Mode for a description of what the different blend mode values do.

The basic mix blend looks like this.

```
h1 {
  margin: 0;
  font-size: 20vw;
  text-transform: uppercase;
  font-family: Montserrat, sans-serif;
  line-height: 1.2;
}

.blended {
  background-image:
    url(../images/tea-ceremony.jpg);
  background-position: 50% 50%;
}

.blended h1 {
  color: #000;
  background: #fff;
```

```
  font-size: 19vw;
  mix-blend-mode: lighten;
}
```

The markup for the blending is as simple as possible. We use a div for the background image and the text we want to blend it with

```
<div class="blended">
  <h1>Ichi-Go Ichi-E</h1>
</div>
```

The second type of blending we can do is background blending. In this we take one or more background images, gradients, or solid colors and blend them together.

The second example, taken from Codrops background-blend-mode page mixes two backround images and a color to create a very interesting effect.

The important part is that we can have any number of background images, colors or gradients and any number of blend modes to create whatever effects we can imagine.

```
.backgrounder2 {
  width: 600px;
  height: 400px;

  background-image:
    url(images/blend-mode-texture.jpg),
    url(images/blend-mode-image.jpg);
  background-color: olive;
  background-blend-mode: hard-light;
}
```

One final thing when talking about blend modes is that we can use gradients as background images to blend.

In the next example we swap one of the images with a linear gradient to blend

the image with, taking advantage of everything we can do with gradients.

```
.backgrounder3 {
  width: 600px;
  height: 400px;

  background-image:
    linear-gradient(to right, #69B62F, #DE3375),
    url(images/blend-mode-example-texture.jpg);
  background-blend-mode: hard-light;
}
```

Blend modes give us one more tool in the battle for creative layouts on the web.

z-index

z-index controls the stacking order of elements that share the same parent. For people looking at the screen elements with a higher z-index value will appear on top, or over, other elements without z-index.

`z-index` will only work on positioned elements. Even if you give an element a z-index value, it will have no effect on the element unless it is positioned; i.e. unless it has a position value other than the default `static`.

This figure illustrates the stacking order of different elements.

Figure 1: Stacking order when using z-index and other elements

This example uses the following HTML:

```
<div class="container">
  <div class="image">
    <img
      src='image/chile-02.jpg'
      alt='Easter Island, Chile'>
  </div>
  <h1>Easter Island</h1>
```

```
</div>
```

And the acompanying CSS

```
.container {
  position: absolute;
}

.image {
  position: relative;
  z-index: 0;
  width: 100%;
}

.text {
  position: absolute;
  z-index: 10;
  width: 100vw;
  color: white;
  font-family: cursive;
  font-size: 8vw;
  left: 500px;
  top: 800px
}
```

To stack the text on top of the image to give the impression that the text is laid on top. Because we're using absolutely positioned content we can play with left, top and z-index to create combinations of positioned text.

Final Notes: This is a nice to have

One final thing to remember is that, for all the cool things you can do with the techniques in this post, in the end they are icing on the cake.

Where possible, and assuming that your target browsers support it, use feature queries, the @support at-rule.

This way you can provide two branches of code based on the support for a feature or the lack thereof.

As an example, this code will only use the code for `#motion-demo` if the browser supports either the prefixed or unprefixed versions of clip-path

```
@supports (clip-path: polygon(0 0)) or (-webkit-clip-path: polygon(0 0)) {
  #motion-demo {
    offset-path: path('M6,150C49.63,93,105.79,36.65,156.2,47.55,207.89,58
    background: rebeccapurple;
  }
}
```

Lkewise, if you use an inline SVG image to take advantage of what you can do with it, remember to provide an alternative if your target browsers don't support the feature.

References and tools

- clip-path
  - clip-path @ MDN
  - Clippy
  - Firefox Shape Path Editor
  - The SVG path Syntax: An Illustrated Guide
- Masking
  - Mask @ MDN
  - Masking vs. Clipping: When to Use Each
  - mask-image reference at Codrops
  - mask-composite reference at Codrops
- Motion on a path
  - CSS Motion Path @ MDN
  - Get Moving (or not) with CSS Motion Path
  - SVG Path Generator using Mavo
  - Moving Text on a Curved Path
  - #135: Three Ways to Animate SVG CSS Tricks Screencast
  - How to draw a circle using SVG paths
- Blend Modes
  - Compositing and Blending Level 1 W3C Editor's Draft
  - Basics of CSS Blend Modes

        - mix-blend-mode @ MDN
        - background-blend-mode @ MDN
        - mix-blend-mode reference at Codrops
        - background-blend-mode reference at Codrops
- z-index
    - Z’s Still Not Dead Baby, Z’s Still Not Dead
    - CSS Reference: z-index
- Accessibility considerations
    - Designing Safer Web Animation For Motion Sensitivity
    - An Introduction to the Reduced Motion Media Query