SVG as images and their fallbacks

Most of the work I've done recently has been as inline SVG meaning that the SVG is inserted directly into the document; this has advantages and disadvantages. In this post we'll discuss why we would use svg as images, what are the disadvantages and disadvantages and a possible fallback using the picturefill polyfill.

SVG is a very powerful vector graphics format that can be used either as an inline element or as a format for images on web pages. Which one you use will depend on a few things:

- What browsers do we need to support?
- What are we using the graphics for?
- What SVG features do we need for the individual graphic

For the following discussion we'll assume we need to support IE9 and newer plus all modern evergreen browsers; we won't need animation baked into individual icons, if we need to animate we'll do so from CSS or using GSAP. We'll use SVG to create a small set of social media icons to use on the page.

Advantages And Disadvantages Of SVG As An Image

Here are some advantages of working with SVG in images:

Smaller file size: SVG images are made of text describing the shape of the objects in the image so they will be consistently smaller than equivalent raster images.

Scale easier: Because they are vector graphics they scale up or down regardless of resolution. That means that you only have to load one image for all the resolutions and pixel densities you want to use on the page

Compresses better: SVG is text and, most of the time, text will compess better than binary data

Not everything is rainbow and roses, there are a few disadvantages of working with SVG inside an image

Cannot be formatted with CSS: Most of the time you can style SVG images with CSS either inside the element itself or through an external CSS. I can't seem to do so with svg images.

Will not work on older browsers: Not all browsers support SVG images, particularly IE9 and older. IE9 will supprt it but with a workaround.

Next we'll explore how to provide fallbacks for non-supported browsers and a polyfill for making the job easier.

Providing fallbacks

The simplest way for this to work is to use the `picture` element, part of the Responsive Images additions to the HTML specification

The example below shows one ideal way of providing fallback for SVG images and providing a default image to render when neither source is supported. This is a first item matched is used algorithm, similar to what browsers do for the `video` and `audio` elements.

In this example the browser tests support for SVG images and loads and renders it if supporter; if not the browser checks if it can render WebP images and if it doesn't then it falls back to the `img` element that should be rendered by all browsers. I've used a single `src` attribute for the image, we could also add `srcset` and `sizes` to the image to further enhance the responsiveness.

For larger line drawings or diagrams below the fold we could also incorporate lazy loading (native and through polyfill).

```
<picture>
  <source   srcset="examples/images/large.svg"
            type="image/svg+xml">
  <source   srcset="examples/images/large.webp"
            type="image/webp">
  <img      src="examples/images/large.png"
            alt="Large Image Of Cats">
```

```
</picture>
```

Working with Picturefill

The problem is that older browsers are not likely to follow the ideal case. For browsers that don't support the `picture` element we'll have to use a polyfill to make sure that the image will load regardless of the browser we're using.

I've chosen to work with `Picturefill` polyfill for responsive images. It's stable and works in the cases and for the browsers we wanted to tackle when defining the project.

To run the polyfill we first need to trick older versions of IE to accept the picture element before the polyfill has loaded and add the polyfill to the page using a `script` tag with the `async` attribute.

```
<script>
  // Picture element HTML5 shiv
  document.createElement( "picture" );
"picture"</script>
<script async src="picturefill.min.js"></script></script>
```

This makes all responsive images elements (`picture`) and attributes (`srcset`, and `sizes`) available to the page.

Now we move to the fallback solution for SVG images (finally!).

The final code looks pretty close to our ideal example, except for the IE-specific conditional commetns that will only load the video element wrapper for IE9 (this addresses an issue with IE9 handling of source attributes inside a picture).

```
<picture>
  <!--[if IE 9]><video style="display: none;"><![endif]-->
  <source srcset="examples/images/large.svg" type="image/svg+xml">
  <<source srcset="examples/images/large.svg" type="image/svg+xml"><!--[i
  <img src="examples/images/large.png" alt="…">
</picture>
```

And that’s it. We have a way to display SVG images and provide multiple fallbacks for browsers that do not support them and a default image that will be supported everywhere.