Splash vector graphics on your responsive site

- Scaling fundamentals
- Making Vector Backgrounds
  - Adjusting the background to look nicer
  - A better solution
- Using Gradients Efficiently
- Animate With Ease
- Where to from here?
- Other good resources

Creating mobile content that dazzles means balancing the amount of data downloaded against maximal visual impact. Vector graphics are a great way to deliver stunning visual results using minimal bandwidth.

Many people think of canvas as the only way to draw a mixture of vectors and rasters on the web, but there are alternatives that have some advantages. A great way to achieve vector drawing is through the use of Scalable Vector Graphics (SVG) which is a key part of HTML5.

We all know responsive design is a big part of handling varying screen sizes, and SVG is ideal for handling different size screens with ease.

SVG is a great way to present vector based line drawings and is a great complement to bitmaps, the latter being better suited for continuous tone images.

One of the most useful things about SVG is that it’s resolution independent, meaning that you don’t need to think about how many pixels you have on your device, the result will always scale and be optimized by the browser to look great.

Popular authoring tools like the Drawing application in Google Drive, Inkscape, Illustrator, Corel Draw and lots of others generate SVG so there are lots of ways to generate content. We'll dive into some ways to use SVG assets, plus some optimization tips to get you going.

Scaling fundamentals

Let's start with a simple scenario - you want a full page graphic to be the background of your web page. It'd be really useful to have your company logo, or anything like that full-screen in the background at all times, but of course that's super hard to do nicely with all the different screen sizes out there. So to illustrate, we'll start with the humble HTML5 logo.

The HTML5 logo is shown below - and you guessed it, it originates as an SVG file.

HTML5 logo

Click on the logo and take a look at it in any modern browser and you'll see that it scales beautifully to any size window. Try opening it up in your favorite browser, resize the window and observe that the image is crisp at any magnification. If we were to try that with a bitmap image, we'd either have to serve many different sizes for each screen we could encounter, or be forced to put up with horribly pixelated scaled images.

So what's the big deal? Well if you hadn't noticed, this is the only format that scales independently of the device we're using to look at it with. So we only need to serve one asset to our users, without ever needing to know what their screen or window size is - neat!

But wait, there's more - the HTML5 logo is just 1427 bytes! Yikes, that's so small that it will hardly dent any mobile data plan when loading it, which makes it fast to load and that makes it cheap and fast for your users!

Another nice thing about SVG files is that they can be GZIP compressed to further shrink them. When you compress SVG that way, the file extension has to be changed to ‘.svgz’. In the case of the HTML5 logo, it shrinks to just 663 bytes when compressed - and most modern browsers handle it with ease!

Note, some browsers can load the GZIP files OK but fail on loading via the `<img>` tag. Luckily HTTP can use GZIP compression so setting up the web server properly may be a better way to take advantage of GZIP compression.

With our example file on some of the latest devices we see something like a 60x advantage using compressed vector data! Also, note that these comparisons are being made between the JPEG and the SVG, rather than PNG. However, the JPEG is a lossy format which results in lower quality than either the SVG or PNG. If we were to use PNG, the advantage would be over 80x which is astounding!

But of course PNG and JPEG are not created equal. A number of optimization tips tell you to use JPEG instead of PNG, but that's not always such a great idea. Take a look at the images below. The image on the left is a PNG image of the top right part of the HTML5 logo enlarged 6x. The image on the right is the same thing but encoded with JPEG.

PNG image

```
<figcaption>PNG image</figcaption>
```

JPEG image<

```
<figcaption>JPEG image</figcaption>
```

It's easy to see that the file size saving in JPEG comes at a cost, with color artifacts at the sharp edges - probably making your retina think it needs glasses:-) To be fair, JPEG is optimized for photos, and that's why it's not as good for vector art. In any case, the SVG version is the same as the PNG in quality so wins on all accounts - both file size and clarity.

Making Vector Backgrounds

Lets take a look at how you could use a vector file as the background of a page. One easy way is to declare your background file using CSS fixed positioning:

```
<style>
#bg {
  position:fixed;
  top:0;
  left:0;
  width:100%;
  z-index: -1;
}
</style>
<img src="HTML5-logo.svgz" id="bg" alt="HTML5 logo"></pre>
```

You'll notice that no matter what the size of the display, the image is sized nicely with crisp clean edges.

Then of course, we'd like to put some content over the background.

Logo with background

But as you can see, the result is less than ideal since we can't read the text. So what do we do?

Adjusting the background to look nicer

The obvious thing we need to do is make all the color in the background image be lighter. This is easily achieved by use of the CSS opacity property - or using opacity in the SVG file itself. You could make this work by simply adding this code to your CSS content:

```
#bg {
  opacity: 0.2;
}
```

That would give you a result like so:

Adjusting the background to look nicer

This solution whilst easy, is likely to be a performance pain point on a mobile device. For most existing mobile browsers, use of the opacity property can be a lot slower to draw with compared with opaque objects.

A better solution

Modifying the color in the original SVG content is far better than setting opacity with CSS. Here is our HTML5 logo modified to look faded out by changing the colors used, and in the process avoiding the opacity property altogether. So the background image below looks identical to the result from changing the opacity, but will actually paint a lot faster and save us CPU time and save precious battery life in the process.

Logo faded

So now we have a decent grasp of some fundamentals, let's move on to some other features.

Using Gradients Efficiently

Let's say we want to build a button. We could start by creating a rectangle with nice rounded corners. Then we could add a nice linear gradient to give the button some nice texture. The code to do so could look something like:

```
<svg xmlns="http://www.w3.org/2000/svg">
  <defs>
    <linearGradient id="blueshiny">
      <stop stop-color="#a0caf6" offset="0"/>
      <stop stop-color="#1579df" offset="0.5" />
      <stop stop-color="#1675d6" offset="0.5"/>
      <stop stop-color="#115ca9" offset="1"/>
```

```
    </linearGradient>
  </defs>
  <g id="button" onclick="alert('ouch!');">
    <rect fill="url(#blueshiny)" width="198" height="83" x="3" y="4" rx="15" />
    <text x="100" y="55" fill="white" font-size="18pt" text-anchor="middle">Press me</text>
  </g>
</svg>
```

The resultant button would end up with something like this:

Glossy button

Notice how the gradient we've added goes from left to right. This is the default gradient direction in SVG. But we can do better, for a couple of different reasons: aesthetics and performance. Let's try to change the gradient direction to make it look a bit nicer. Setting the 'x1', 'y1', 'x2', and 'y2' attributes on the linear gradient controls the direction of the fill color.

Setting just the 'y2' attribute lets us change the gradient to be diagonal. So this small code change:

```
<linearGradient id="blueshiny" y2="1">
```

gives us a different look for our button, it ends up looking like the image below.

Glossy button slanted

We can also easily change the gradient to go from top to bottom with this small code change:

```
<linearGradient id="blueshiny" x2="0" y2="1">
```

and that ends up looking like the image below.

Glossy button vertical

So what's with all the discussion about different angles of the gradient I hear you ask?

Tip: use top to bottom gradients for better performance.

Well it turns out that the last example - the one with the gradient running from top to bottom is the fastest to draw on most devices. It's a very little known secret amongst the graphic geeks who write browser code that vertical (top to bottom) gradients paint almost as fast as a solid color. (The reason is that painting an object is done in horizontal lines down the page - and the guts of the drawing code understand that the color doesn't change across each line, and so they optimize it).

So when you choose to use gradients in your page design, vertical gradients will be faster and use less battery as a side effect. This speedup applies to CSS gradients as well, so it's not just an SVG

thing.

Note, that radial gradients should be avoided unless you know they'll be hardware accelerated - in software they are painfully slow.

If we feel really adventurous with this new gradient knowledge, then we can perhaps add a cool gradient behind our HTML5 logo by adding the code below:

```
<defs>
<linearGradient id="grad1" x2="0" y2="1">
    <stop stop-color="#FBE6FB" offset="0" />
    <stop stop-color="#CCCCFF" offset="0.2" />
    <stop stop-color="#CCFFCC" offset="0.4" />
    <stop stop-color="#FFFFCC" offset="0.6" />
    <stop stop-color="#FFEDCC" offset="0.8" />
    <stop stop-color="#FFCCCC" offset="1" />
</linearGradient>
</defs>
<rect x="-200" y="-160" width="910" height="830" fill="url(#grad1)"/></pre>
```

The code above adds a faded vertical linear gradient to the backgound of our HTML5 logo to give a subtle multi-color tinge that runs fast - as fast as a solid color background.

Note, also that the choice of the size of the background rectangle guarantees the color fills the screens on all of our mobile devices listed at the start of this article (their screen aspect ratios range from 10/16 through 18/10).

If you load the content in a desktop browser and resize to extreme aspect ratios, you'll see white bars appear at the top/bottom or left/right sides. Anyway, after the code changes made above the resultant background will look like this:

Logo faded with gradient

Animate With Ease

By now you might be wondering what's the point of using an SVG gradient as the backdrop for your page. Indeed it might make sense to do that with CSS gradients, but one advantage of SVG is that the gradient itself lives in the DOM. This means that you can modify it with script, but more importantly you can take advantage of SVG's built in animation capability to add subtle changes to your content.

As an example, we'll modify our colorful HTML5 logo by changing the linear gradient definition to the code below:

```
<linearGradient id="grad1" x2="0" y2="1">
    <stop stop-color="#FBE6FB" offset="0">
    <animate attributeName="stop-color"
        values="#FBE6FB;#CCCCFF;#CCFFCC;#FFFFCC;#FFEDCC;#FFCCCC;#FBE6FB"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
    <stop stop-color="#CCCCFF" offset="0.2">
    <animate attributeName="stop-color"
        values="#CCCCFF;#CCFFCC;#FFFFCC;#FFEDCC;#FFCCCC;#FBE6FB;#CCCCFF"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
    <stop stop-color="#CCFFCC" offset="0.4">
    <animate attributeName="stop-color"
        values="#CCFFCC;#FFFFCC;#FFEDCC;#FFCCCC;#FBE6FB;#CCCCFF;#CCFFCC"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
    <stop stop-color="#FFFFCC" offset="0.6">
    <animate attributeName="stop-color"
        values="#FFFFCC;#FFEDCC;#FFCCCC;#FBE6FB;#CCCCFF;#CCFFCC;#FFFFCC"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
    <stop stop-color="#FFEDCC" offset="0.8">
    <animate attributeName="stop-color"
        values="#FFEDCC;#FFCCCC;#FBE6FB;#CCCCFF;#CCFFCC;#FFFFCC;#FFEDCC"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
    <stop stop-color="#FFCCCC" offset="1">
    <animate attributeName="stop-color"
        values="#FFCCCC;#FBE6FB;#CCCCFF;#CCFFCC;#FFFFCC;#FFEDCC;#FFCCCC"
        begin="0s" dur="20s" repeatCount="indefinite"/>
    </stop>
</linearGradient>
```

Take a look at the image below to see the result of these changes above.

Linear gradient

The code is changing the colors of our linear gradient through all the different color stops we've defined in a continuous cycle that takes 20 seconds to run. The effect of that is that the gradient looks like it's moving up the page in a continuous motion that never stops.

The beauty of this is that there's no script required! That's why it runs as a referenced image from this page, but also it reduces workload on a mobile CPU by removing the need for script.

Also, the browser itself can take advantage of its knowledge of the painting to ensure minimal CPU overhead is used to do the fancy animation.

There is one caveat: some browsers don't handle this style of animation at all, but in that case you'll still get a nice colored background but it just won't change - this could be worked around by using script (and requestAnimationFrame) but that's a bit beyond this article.

One more thing to note, is that this uncompressed SVG file is only 2922 bytes - incredibly small to supply such a rich graphics effect, keeping your users and those data plans happy in the process.

Where to from here?

There are many cases where SVG is less than ideal, photos and videos being better represented in other formats. Text is another one, where native HTML and CSS work much better in general. However, as a tool in the arsenal for line drawn artwork it can be the ideal choice.

We've touched on a few basic fundamental uses for SVG graphics, showing how easy it is to generate tiny content that provides full screen vivid graphics with a minimal amount of download. Small enhancements to the content can create amazing graphical results easily with trivial amounts of markup. In the next article we'll explore some more details about how the animation built into SVG can be used for more simple and powerful effects and compare the use of canvas with SVG to choose the right tool for authoring your mobile graphics site.

Other good resources

- Inkscape an open source drawing application that uses SVG as its file format.
- Open Clip Art a huge open source clip art library containing thousands of SVG images.
- W3C SVG Page containing links to specifications, resources, etc.
- Raphaël a Javascript library that provides a convenient API to draw and animate SVG content with great fallback for older browsers.
- SVG Resources from Slippery Rock University - includes a link to a great SVG Primer.

□□□ #3
□ □□□ □□□ 2026-10-31 22:19:55 UTC

 cocobook-sync □□□ 2026-11-04 17:38:10 UTC