How to Scale SVG

Amelia Bellamy-Royds on Jun 27, 2017

The following is a guest post by Amelia Bellamy-Royds. Amelia has lots of experience with SVG, as the co-author of SVG Essentials (http://shop.oreilly.com/product/0636920032335.do) and author of the upcoming Using SVG with CSS3 and HTML5 (http://shop.oreilly.com/product/0636920037972.do) . Amelia and I both will be speaking on SVG at the upcoming RWD Summit (http://environmentsforhumans.com/2015/responsive-web-design-summit/) as well! Here, she shares an epic guide to scaling SVG, covering all the ways you might want to do that. It's not nearly as straightforward as scaling raster graphics, but that can be good, because it opens up interesting possibilities.

You've made the decision. You're finally going to do it. This year, you are going to start using SVG (https://css-tricks.com/using-svg/) in your web designs. You create a fabulous header logo in Inkscape and you copy and paste the SVG code it spits out into your WordPress header file. Of course, you're not giving up on last year's resolution to always use responsive design, so you set `svg.banner { width: 100%; height: auto; }` in your CSS and you think you're set.

Until you open up your web page in test browsers and discover that some leave huge blocks of whitespace above and below the image, while others crop it off too short.

SVG stands for Scalable Vector Graphics. So, scaling SVG should be easy, right? Isn't that what the SVG advocates have been saying all along, that SVG looks good at any size? It is, but yet it isn't. SVG looks great at any scale, but it can scale in so many different ways that getting it to behave just the way you want can be confusing for SVG beginners. It doesn't help that browsers have only recently started to adopt a standard approach to sizing inline SVG content.

(#svg-is-not-just-an-image) SVG is not (just) an image

Part of the reason that scaling SVG is so difficult is because we have a certain idea about how images *should* scale, and SVG doesn't behave in the same way.

Raster images like JPG, PNG, and GIF, have a clearly defined size. The image file describes how the browser should color in a grid that is a certain number of pixels wide and a certain number of pixels tall. An important side effect is that raster images have a clearly defined *aspect ratio*: the ratio of width to height.

You can force the browser to draw a raster image at a different size than its intrinsic height and width, but if you force it to a different aspect ratio, things will get distorted. For this reason, since the early days of the web there has been support for `auto` scaling on images: you set the height *or* the width, and the browser adjusts the other dimension so that the aspect ratio stays constant.

Embedded Pen Here

SVG images, in contrast, can be drawn at any pixel size, so they don't need a clearly defined height or width. And they won't always have a clearly defined aspect ratio. You're going to need to explicitly provide this information (and more) if you want the SVG to scale to fit the dimensions you give it.

If you don't, SVG won't scale at all. The following example uses inline SVG, adjusting the dimensions of the element (dotted line), without ever altering the size of the drawn graphic:

Embedded Pen Here

Why does it behave this way? Because SVG isn't (just) an image. SVG is a document. Although the above example uses inline SVG, it could just as easily have used `<object>` or `<iframe>`. It would look the exact same even if you used `<img>` tags to embed the same SVG code.

When you include an HTML file with an `<iframe>`, you don't expect the text inside to scale when you change the size of the frame. Same with SVG. By default, it will be drawn at the size specified in the code, regardless of the size of the canvas.What happens if you set the height or width (or both) to `auto` for these SVGs? The default size for HTML replaced elements will be used: 300px wide, 150px tall. This applies for `<img>`, `<object>` or `<iframe>`. The default 300×150 size also applies to inline `<svg>` elements within HTML documents, but that's a relatively recent consensus from the HTML5 specifications: other browsers will by default expand inline SVG to the full size of the viewport—equivalent to `width: 100vw; height: 100vh;` — which is the default size for SVG files that are opened directly in their own browser tab. Internet Explorer cuts the difference, using width of 100% and height of 150px for images and inline SVG.

In other words, even if you think 300×150 is a perfect image size (though why would you?), don't rely on having a default size for `<svg>` in HTML.

In addition to deciding what size you want your SVG to be, you're also going to have to decide how you *want* your graphic to scale to fit that size. Below, I describe the code you need to get the scale you want for the most common situations:

- Scaling to fit a certain size, without distorting the image
- Scaling to fit a certain size, stretching or compressing the graphic as necessary
- Scaling to fit the available width, while maintaining the width-to-height aspect ratio
- Scaling in non-uniform ways, so that some parts of the graphic scale differently from others

But first: If you want to take control of the scale of your SVG, you're going to need to familiarize yourself with the SVG scaling attributes and other tools available.

(#the-svg-scaling-toolbox) The SVG Scaling Toolbox

Other images scale because the browser knows the height, width, and aspect ratio of the image, and it adjusts everything together. Giving SVG these properties is the first step to getting it to scale. However, scaling SVG goes beyond what is possible with other images.

(#the-height-and-width-attributes) The `height` and `width` attributes

A first glance at the SVG specifications would suggest that the `height` and `width` attributes on the top-level `svg` element will implicitly set an aspect ratio and therefore make SVG scale like other images. It's true that setting height and width will override the default dimensions when you use SVG as an image. But of course it's not that easy:

- If you use an `<img>` to embed your SVG, setting `height` and `width` will make the SVG scale predictably in *most* browsers, but not in Internet Explorer. With CSS like `img { width: 100%; height: auto; }`, IE will auto-scale the image area to keep the width:height aspect ratio constant, but it won't scale the actual drawing to match the scale of the image dimensions.
- If you use an `<object>`, `<embed>`, or `<iframe>` to embed your SVG, setting height and width on the `<svg>` won't change the size of the frame; you'll just get scrollbars inside your iframe if the SVG is too big.
- If you use inline SVG (i.e., `<svg>` directly in your HTML5 code), then the `<svg>` element does double duty, defining the image area within the web page as well as within the SVG. Any height or width you set for the SVG with CSS will override the height and width attributes on the `<svg>`. So a rule like `svg {width: 100%; height: auto;}` will cancel out the dimensions and aspect ratio you set in the code, and give you the default height for inline SVG. Which, as mentioned above, will be either 150px or 100vh, depending on the browser.

So forget `height` and `width`. You don't actually want to set the *exact* height and width anyway, you want the SVG to scale to match the width and/or height you set in the CSS. What you want is to set an *aspect ratio* for the image, and have the drawing scale to fit. You want a `viewBox`.

(#the-viewbox-attribute) The `viewbox` attribute

The SVG `viewBox` is a whole lot of magic rolled up in one little attribute. It's the final piece that makes vector graphics *Scalable* Vector Graphics. The viewBox does many things:

- It defines the aspect ratio of the image.
- It defines how all the lengths and coordinates used inside the SVG should be scaled to fit the total space available.
- It defines the origin of the SVG coordinate system, the point where x=0 and y=0.

The `viewBox` is an attribute of the `<svg>` element. Its value is a list of four numbers, separated by whitespace or commas: x, y, width, height. The width is the width in user coordinates/px units, *within the SVG code*, that should be scaled to fill the width of the area into which you're drawing your SVG (the view*port* in SVG lingo). Likewise, the height is the number of px/coordinates that should be scaled to fill the available height. Even if your SVG code uses other units, such as inches or centimeters, these will also be scaled to match the overall scale created by the `viewBox`.

The x and y numbers specify the coordinate, in the scaled `viewBox` coordinate system, to use for the top left corner of the SVG viewport. (Coordinates increase left-to-right and top-to-bottom, the same as for identifying page locations in JavaScript). For simple scaling, you can set both values to 0. However, the x and y values are useful for two purposes: to create a coordinate system with an origin centered in the drawing (this can make defining and transforming shapes easier), or to crop an image tighter than it was originally defined.

Some example `viewBox` values:

- `viewBox="0 0 100 100"`: Defines a coordinate system 100 units wide and 100 units high. In other words, if your SVG contains a circle centered in the graphic with radius of 50px, it would fill up the height or width of the SVG image, even if the image was displayed full screen. If your SVG contained a rectangle with `height="1in"`, it would also nearly fill up the screen, because 1 inch = 96px in CSS, and all lengths will get scaled equally.
- `viewBox="5 0 90 100"`: Almost the same view, but cropped in by 5% on the left and right, so that the total width=90 units and the x-coordinate on the left=5.
- `viewBox="-50 -50 100 100"`: A view with the same scale, but now with the top-left corner given the coordinates (-50, -50). Which means that the *bottom-right* corner has the coordinates (+50, +50). Any shapes drawn at (100, 100) will be far offscreen. If you wanted to draw a circle that completely filled the image area, it would be centered at (0, 0).

Once you add a `viewBox` to your `<svg>` (and editors like Inkscape and Illustrator will add it by default), you can use that SVG file as an image, or as inline SVG code, and it will scale perfectly to fit within whatever size you give it. However, it still won't scale quite like any other image. By default, **it will *not* be stretched or distorted** if you give it dimensions that don't match the aspect ratio. Instead, the scale will be adjusted in order to preserve the aspect ratio defined in the code.

(#the-preserveaspectratio-attribute) The `preserveAspectRatio` attribute

The `viewBox` attribute has a sidekick, `preserveAspectRatio`. It has no effect unless a `viewBox` exists to define the aspect ratio of the image. When there is a viewBox, `preserveAspectRatio` describes how the image should scale if the aspect ratio of the viewBox doesn't match the aspect ratio of the viewport. Most of the time, the default behavior works pretty well: the image is scaled until it just fits both the height and width, and it is centered within any extra space.

Just like `viewBox`, `preserveAspectRatio` has a lot of information in a single attribute. The default behavior can be explicitly set with `preserveAspectRatio="xMidYMid meet"`. The first part, `xMidYMid` tells the browser to center the scaled viewBox region within the available viewport region, in both the x and y directions. You can replace `Mid` with `Min` or `Max` to align the graphic flush against one side or the other. Watch the camelCase capitalization, though: SVG is XML, and is therefore case sensitive. The `x` is lowercase but the `Y` is capital.

The second half of the default `preserveAspectRatio`, `meet`, is the part that tells the browser to scale the graphic until it just fits both height and width. It's equivalent for CSS background images is `background-size: contain;`. The alternative value for SVG is `slice` (equivalent to `background-size: cover;`). A `slice` value will scale the image to fit the more generous dimension, and slice off the extra. Except, it doesn't necessarily *slice* off the extra; that depends on the value of the `overflow` property.

(Side note: If you wish every image could be centered in the dimensions you give it, instead of getting stretched or distorted, the new `object-fit` CSS property allows you to do the same with other image types (http://demosthenes.info/blog/967/The-Widescreen-Web-Using-CSS-object-fit).)

There's also `preserveAspectRatio="none"` option to allow your SVG to scale exactly like a raster image (but with much better resolution), stretching or squishing to fit the height and width you give it.

(#how-to-scale-svg-to-fit-within-a-certain-size-without-distorting-the-image) How to Scale SVG to Fit

within a Certain Size (without distorting the image)

Probably the most common requirement is to have an SVG icon scale to fit a specific size, without distortion. The `viewBox` attribute is really all you need here, although you can use `preserveAspectRatio` to adjust the alignment.

As mentioned, if you're creating your SVG in a graphical editor, it's probably already including a viewBox. However, you may want to adjust the viewBox to get the positioning just right. The pot-of-gold graphic has been given a `viewBox="0 0 60 55"` for the rest of the examples. That leaves some extra space around it; to create a tight-cropped icon, you could use `viewBox="4.5 1.5 51 49"`. The following example also shows the effect of the default `preserveAspectRatio`, centering the graphic in the space provided:

Embedded Pen Here

(#how-to-scale-svg-to-fit-the-available-width-and-adjust-the-height-to-match) How to Scale SVG to Fit the Available Width (and adjust the height to match)

SVG with a `viewBox` will scale to fit the height and width you give it. But what about auto-sizing? With raster images, you can set `width` *or* `height`, and have the other scale to match. Can SVG do that?

It can, but it gets complicated. There are a couple different approaches to chose from, depending on how you are including your SVG.

(#option-1-use-image-auto-sizing) Option 1: Use image auto-sizing

When an SVG file has a `viewBox`, and it is embedded within an `<img>`, browsers will (nearly always) scale the image to match the aspect ratio defined in the `viewBox`.

Embedded Pen Here

Internet Explorer, however, remains the bane of SVG. Although it normally works just fine, I used `display: table-cell` to lay out the figures in an earlier version of this example, and IE distorted the images in weird ways.

If you completely auto-size the image, Internet Explorer applies the standard default 300×150 size. However, other browsers will apply `{ width: 100%; height: auto; }` by default if the image has a `viewBox`; this behaviour is not defined in any specification.

So to recap: To auto-scale SVG used as `<img>`,

1. Set a `viewBox` attribute.
2. Set at least *one* of height or width.
3. Don't put it inside a table layout if you care about supporting Internet Explorer.

(#option-2-use-css-background-images-and-the-padding-bottom-hack) Option 2: Use CSS Background Images and the `padding-bottom` Hack

For the most part, using SVG as a CSS background image works much the same way as using it in an `<img>` (but with the added benefit that you can define raster fallbacks for old browsers). There are a few bugs with older browsers scaling the image after converting it to raster instead of before (i.e. pixelating it), but for the most part the `viewBox` is all you need.

However, auto-sizing isn't an option for CSS background images; after all, the image is *supposed* to be secondary to the HTML content of the element. If you want the element to exactly match the aspect ratio of the image you're going to use, you're going to have to hack it a little bit.

There are a select number of CSS properties that allow you to adjust height-based attributes based on the available width. If you set the `border`, `padding`, or `margin` of a block-layout element to percentage values, the percentages will be calculated relative to the available *width* of the container, even for the *top* and *bottom* borders, padding, and margin.

The intended purpose is to create evenly-sized borders and padding even when height is automatic. But that's beside the point. For our purposes, the key point is that you can adjust the total height of an element in proportion to the width. In other words, you can control the aspect ratio. To create a `<div>` with 100% width that exactly matches the 4:3 aspect ratio of an image you're using as its background, you can use:

```css
.ratio4-3 {
 width: 100%;
 background-image: url(image-with-4-3-aspect-ratio.svg);
 background-size: cover;
 height: 0;
 padding: 0; /* reset */
 padding-bottom: calc(100% * 3 / 4);
}
```

Things to note:

- To get the desired height as a percentage of the available width, you multiply the percentage width by the desired height factor, divided by the desired width factor.
- If you want to support browsers that don't support `calc()`, you'll need to do the math yourself (or with a CSS pre-processor).
- If you by default set every element to `box-sizing: border-box` (https://css-tricks.com/inheriting-box-sizing-probably-slightly-better-best-practice/), you'll have to re-set it to use `box-sizing: content-box`. We want it to be `height: 0` *plus* padding, after all.
- The `padding-bottom` property is used instead of `padding-top` because of problems in IE5 (http://alistapart.com/article/creating-intrinsic-ratios-for-video#figure1). Although you're probably not worried about supporting IE5, you might as well be consistent. It is called the `padding-bottom` hack, after all.

For the pot-of-gold image, the aspect ratio was 60:55, which works out as bottom padding of 92%. In action, it looks like this:

Embedded Pen Here

(#option-3-use-inline-svg-and-the-latest-blink-firefox-browsers) Option 3: Use Inline SVG and the latest Blink/Firefox Browsers

SVG images are nice, but in many cases you'll prefer to use inline SVG. Inline SVG reduces the number of HTTP requests, allows user interactions, and can be modified by the CSS in your main web page. But will it scale?

It will if you're using the latest Firefox or Blink browsers. Just set the `viewBox` on your `<svg>`, and set one of height or width to `auto`. The browser will adjust it so that the overall aspect ratio matches the `viewBox`. Beautiful.

Embedded Pen Here

But chances are, these aren't the only browsers you need to support.

Many browsers—IE, Safari, and versions of Opera and Chrome released prior to summer 2014—will not auto-size inline SVG. If you don't specify both height and width, these browsers will apply their usual default sizes, which as mentioned previously will be different for different browsers. The image will scale to fit inside that height or width, again leaving extra whitespace around it. Again, there are also inconsistencies in what happens if you leave both height *and* width `auto`.

The solution is to again use the `padding-bottom` hack to control the aspect ratio yourself. The easiest approach, which works for inline SVG as well as `<object>`, `<iframe>` and other replaced elements like `<video>`, is to use a container element.

(#option-4-use-the-padding-bottom-hack-on-a-container) Option 4: Use the `padding-bottom` Hack on a Container

To use a container `<div>`, add classes or inline styles to the div to give it the correct aspect ratio, as was done above when using a background image. But also set `position: relative` on the container, so that it will become the reference frame for absolutely positioned content. Then set the SVG (or other object) to `position: absolute`, with height and width of 100%. The absolute positioning is required so that the percentages will be calculated relative to the height of the `<div>` *including* the padding, and not relative to the zero-height content area.

Unless you have a lot of graphics with the same aspect ratio, it usually makes sense to declare the `padding-bottom` inline, so that it is right next to the `viewBox` it needs to match:

```html
<div class="scaling-svg-container"
   style="padding-bottom: 92% /* 100% * 55/60 */">
  <svg class="scaling-svg" viewBox="0 0 60 55" >
    <!-- SVG content -->
  </svg>
</div>
```

```css
.scaling-svg-container {
 position: relative;
 height: 0;
 width: 100%;
 padding: 0;
 padding-bottom: 100%;
 /* override this inline for aspect ratio other than square */
}
.scaling-svg {
 position: absolute;
 height: 100%;
 width: 100%;
 left: 0;
 top: 0;
}
```

Embedded Pen Here

The container approach works, but at the cost of an extra wrapper element in your markup. And it isn't a general-purpose container, either: it's a container that has to be customized to the exact aspect ratio your SVG needs. Things get even trickier if you don't want it to scale to a full 100%; you'll need to use another wrapper `<div>` to set the desired width and other positioning attributes. I personally would rather keep all information about the SVG aspect ratio in the SVG code itself. To do that for inline SVG, you're going to need to tell the browser to draw outside the lines, and into the padding.

(#option-5-use-the-padding-bottom-hack-on-an-inline-svg-element) Option 5: Use the `padding-bottom` Hack on an Inline <svg> Element

To use the padding-bottom hack to control the aspect ratio of the total `<svg>` area, the official height is going to be (essentially) zero. With the default `preserveAspectRatio` value, the graphic would be scaled down to nothing. Instead, you want your graphic to stretch to cover the entire width you give it, and to spill out onto the padding area you have carefully set to the correct aspect ratio.

Again, I like to use inline styles for the `padding-bottom` aspect ratio, since it needs to be customized to the `viewBox` attribute. In the example that follows, I also use it for the other style properties, although you could use a class if you have many graphics that need the same effects:

```HTML
<svg viewBox="0 0 60 55" preserveAspectRatio="xMidYMin slice"
   style="width: 100%; padding-bottom: 92%; height: 1px; overflow: visible">
  <!-- SVG content -->
</svg>
```

There are a few other details in there:

- The height is 1px, not 0, otherwise the SVG may not be drawn at all (Firefox) or may not scale at all (Chrome).
- The `preserveAspectRatio` uses `YMin` for the vertical alignment, so that the graphic is aligned neatly against the top of the `<svg>` content area, spilling out into the bottom padding.
- Although `overflow: visible` may be the default for HTML, it needs to be set explicitly for SVG.

If you want the SVG to scale to some percentage less than 100% width, remember to adjust `padding-bottom` accordingly. Or use a wrapper `<div>` to set the size.

Embedded Pen Here

(#how-to-scale-stretch-and-squish-svg-to-exactly-fit-a-certain-size) How to Scale, Stretch, and Squish SVG to Exactly Fit a Certain Size

Although preserving the aspect ratio is usually desirable, sometimes the image is an abstract or flexible image that you want to stretch to fit.

(#option-1-use-percentages) Option 1: Use percentages

One option to stretch to fit is to use percentage values for all size and position attributes in the SVG.

Embedded Pen Here

Things to note about percentages and SVG:

- If you're using percentages to stretch and squish, don't include a `viewBox` (although you can specify default height and width).
- Some lengths in SVG aren't clearly associated with either height or width; for example, the radius of a circle. If you use percentage values in these cases, the length will be calculated as a geometric average (square root of the sum of the squares, divided by square root of 2) of the equivalent percentage of height and width. This preserves the Pythagorean theorem relationship of diagonal lines to rectangular lines, but is otherwise somewhat confusing. (https://codepen.io/AmeliaBR/pen/bNBeNz?editors=110)
- Many lengths in SVG *cannot* be specified with percentages, most importantly the coordinates of `<path>` and `<polygon>` elements.

(#option-2-use-preserveaspectrationone) Option 2: Use `preserveAspectRatio="none"`

If you want a flexibly scaling SVG that also includes SVG paths, you need to use a `viewBox` plus `preserveAspectRatio="none"`. Here's a slightly fancier version of that rainbow, with puffy cloud s:

Embedded Pen Here

Be aware that with `preserveAspectRatio="none"`, everything gets stretched or squished equally, just as if you were unevenly scaling other image types. That means that circles get stretched into ellipses, and text will be distorted as well. To avoid that, you'll need to use a mixture of scaling approaches.

(#how-to-scale-parts-of-an-svg-separately) How to Scale Parts of an SVG Separately

The `viewBox` and `preserveAspectRatio` attributes are incredibly flexible. Once you stop thinking of SVG as just another image format, you can start asking yourself *how* you want your graphic to scale as the window changes size.

An important thing to realize is that you don't need to define a single `viewBox` and `preserveAspectRatio` option for the entire SVG. Instead, you can use nested `<svg>` elements, each with their own scaling attributes, to have different parts of your graphic scale independently. (You can also use these attributes on `<symbol>` and `<pattern>` elements, and you can use `preserveAspectRatio` on other images embedded in your SVG.) With this approach, you can create a header graphic that stretches to fill a widescreen display without taking excessive height as well:

Embedded Pen Here