SVG

Mohamed Emary

December 5, 2023

What is SVG

SVGs or "Scalable Vector Graphics" are a **scalable** image format. You can change their properties through CSS and JavaScript.

SVGs are often used for:

- Icons
- Graphs/Charts
- Large, simple images
- Patterned backgrounds
- Applying effects to other elements via SVG filters

SVGs are Vector graphics which means they are images defined by math. however raster images are defined by a grid of pixels.

SVGs are defined using XML. XML (aka, "Extensible Markup Language") is an HTML-like syntax which is used for lots of things, from **APIs**, to **RSS**, to spreadsheet and word editor software.

Since SVG uses XML it is human-readable. If you were to open up a JPEG in a text editor, it would just look meaningless. But SVGs will look like this:

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100">
    <rect x=0 y=0 width=100 height=50 />
    <circle class="svg-circle" cx="50" cy="50" r="10"/>
</svg>
```

They are understandable compared to **binary file formats** like JPEG.

> *A binary file is a computer file that is not a text file.*

XML is designed to be interoperable with HTML, So you can put the above code directly in an HTML file, without any changes, and it should display the image. And because these can become elements in the DOM just like HTML elements, you can target them with CSS and create them using the Element WebAPI you've already been using!

SVGs are great for relatively simple images, but because every single detail of the image needs to be written out as XML, they are extremely inefficient at storing complex images. If your image is supposed to be photo-realistic, or it has fine detail or texture ("grunge textures" are a great example), then SVGs are the wrong tool for the job.

Anatomy of an SVG

`xmlns` is the XML namespace. It tells the browser that this is an SVG file.

`viewBox` defines the bounds of your SVG. When you have to define the positions of different points of the elements in your SVG, this is what that's referencing. It also defines the aspect ratio and the origin of your SVG. Be sure to play around with different values in the example above to get a feel for how it affects the shapes in this example.

`class` & `id` are just like in HTML. They allow you to target elements with CSS and JavaScript.

Elements such as `<circle>`, `<rect>`, `<path>`, and `<text>` are defined by the SVG namespace. They are the basic building blocks of SVG images. You can find a full list of SVG elements here.

Many SVG attributes, such as `fill` and `stroke`, can be changed in your CSS.

Linking SVG Image

Linking SVGs works basically the same way as linking any other image. You can use an HTML image element such as `<img>`, or link it in your CSS using `background-image: url(./my-image.svg)`. They will still scale properly, but the contents of the SVG will not be accessible from the webpage.

The alternative is to **inline** your SVGs by pasting their contents directly into your webpage's code, rather than linking to it as an image. It will still render correctly, but the SVG's properties will be visible to your code, which will allow you to alter the image dynamically via CSS or JavaScript.

Inlining SVGs allows you to unlock their full potential, but it also comes with some serious drawbacks: it makes your code harder to read, makes your page less cacheable, and if it's a large SVG it might delay the rest of your HTML from loading.

Some of the drawbacks of inlining SVG code can be avoided once you've learned a front-end JavaScript library like React, or a build-tool like webpack.

Use linking unless you need to alter the SVG's properties via CSS or JavaScript.

This lesson have some useful additional resources