Intro to SVGs

(for academic folk)

*Scalable **Vector** Graphics*

This is a training on SVGs (scalable vector graphics). It is aimed at people in academia and and how they might most commonly use the format.

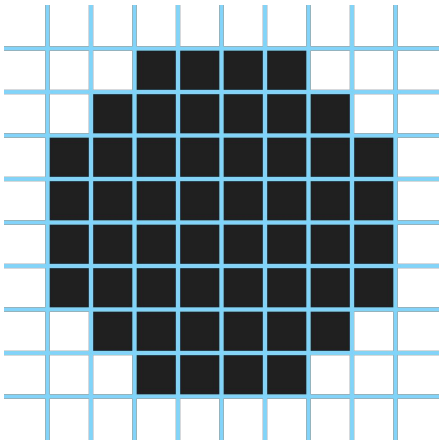
This training should teach you:

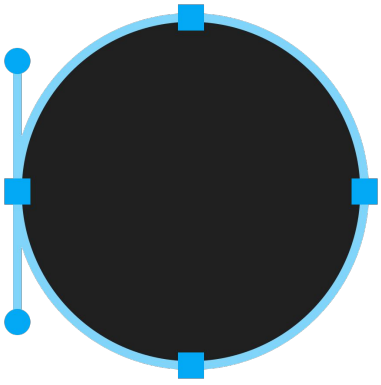
- The basic building blocks of the SVG format
- How to make basic SVGs by hand
- How to more confidently edit SVGs made by software or third parties
- When it is appropriate to make an SVG by hand and when it is not
- General familiarity with the format such that you can more effectively Google a particular problem

These slides should also serve as a good basic reference for when you forget a particular method or syntax.

Background

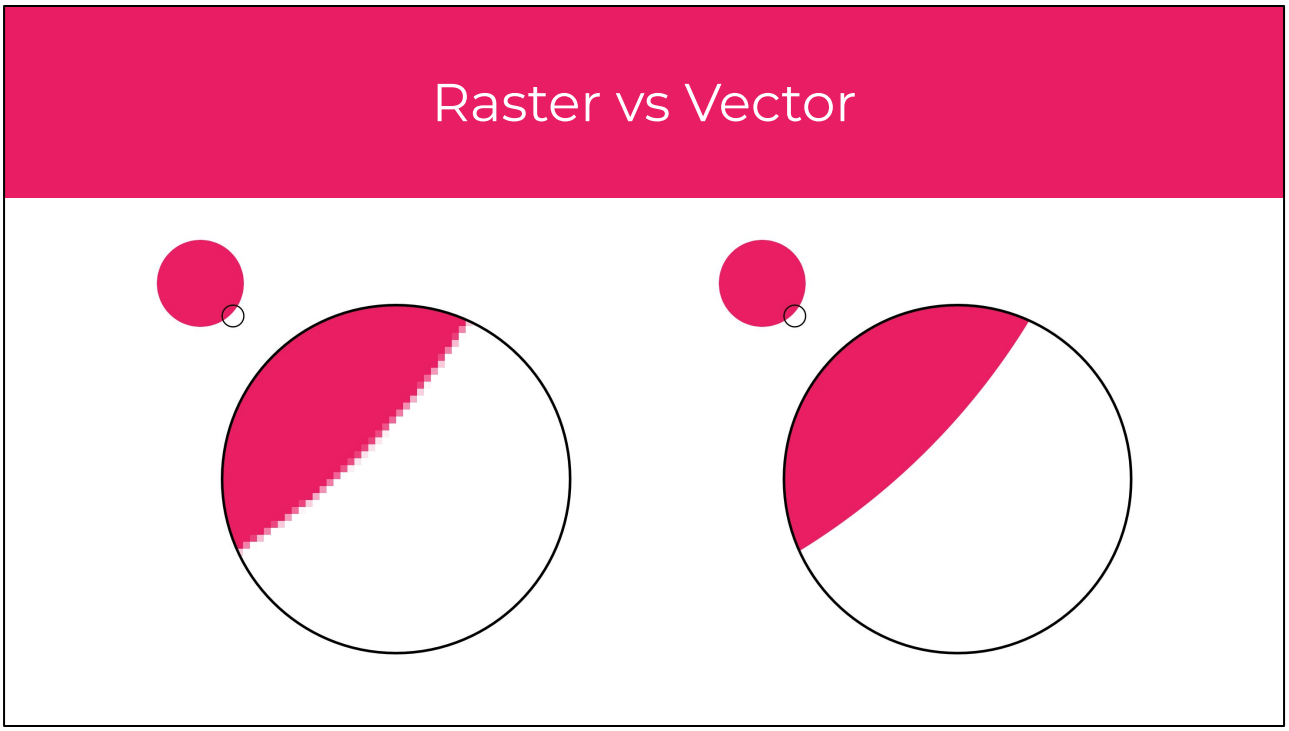
Raster vs Vector





What are vector and raster graphics?

A raster graphic is a grid of pixels.
A vector graphic is a collection primitive shapes.



Why do we prefer vector graphics over raster graphics?

A vector graphic can be scaled to any size with perfect clarity and definition. Internally, when a piece of software displays a vector graphic, it "renders" the shapes to a grid of pixels with the same resolution as your monitor, producing a smooth, crisp result. A raster graphic can be scaled to any size too, but requires some sort of upsampling algorithm to interpolate what should go in between the original pixels, which usually produces poor, blurry results.

In addition, vector graphics usually have a smaller file size than raster graphics, because they are defined by a few lines of text that describe shapes, rather than many rows and columns of individual pixels.

What's an example of vector graphics that almost everyone has used? [redacted]

Limitations of Vector Graphics

What are the limitations of vector graphics?

Because vector graphics are drawn with primitive shapes, they are better suited to simpler, less detailed, more "geometric" images that can be drawn using basic shapes. More "photographic" images, such as realistic depictions of people, animals, etc, are usually better captured by raster images.

What is SVG

What is SVG?

It's the most popular vector graphic format. It was developed by the W3C, the organization in charge of defining web standards like HTML, CSS, and JavaScript.

Cautionary:

Although it was originally aimed at the web, SVG became so popular that you can now see it in a lot of other contexts too, like Word documents, PDFs, graphs, illustrations, graphic design, printed media, etc. Keep this in mind when using SVG outside of a browser: the context you're using it in might not support all of the advanced features that a browser does, because it has essentially co-opted the technology from another platform.

Basics

How are SVGs written

```
<element attribute="value">

    <childElement attribute="value">
        ... more content ...
    </childElement>

    <childElement attribute="value" />

</element>

<!-- comment -->
```

How are SVGs written?

SVGs are just plain text files that contain text descriptions of what shapes to draw. You can create and edit them in any text editor. You can also use software like Inkscape or Adobe Illustrator to make more complex SVGs, but they are still saved and represented as plain text.

What is the language of these text descriptions?

SVGs are written in a simple markup language called XML that consists of three main concepts:

1) Elements - the individual components or building blocks of your image
2) Element attributes - the properties attached to an element that describes its appearance, behavior, etc.
3) Element hierarchy - the organizational structure of your image, formed by arranging elements in an order or nesting them within one another

Element with children elements inside:
`<element><child>...</child></element>`
Element with attribute:
`<element attribute="value">...</element>`
Self-closing/empty element:
`<element attribute="value" />`

The <svg> tag

```
<svg
    xmlns="http://www.w3.org/2000/svg"
    viewBox="..."
    width="..."
    height="..."
>
    ...
</svg>
```

How do we start writing an SVG?

In every SVG, there is a top level `<svg>` element that contains all the contents of the image and key attributes about the image. There are only 4 attributes you will likely ever use here: `xmlns`, `viewBox`, `width`, and `height`.

In SVGs generated by software, you will often see many other attributes and tags at or near the top-level of the document. Many of these are unnecessary, or only necessary in very specific contexts. Most likely, they are there either to support legacy browsers or versions of the SVG spec, or to provide a specific SVG editing software with supplemental metadata to aid editing. When in doubt, just remove a line and see if it still works.

What is xmlns?

The `xmlns` attribute simply tells the viewing software that the XML document is meant to be parsed as an SVG. It is always required, except in the rare case that you are including an SVG directly (inline) in an HTML document.

Units

Absolute:

1px = 1

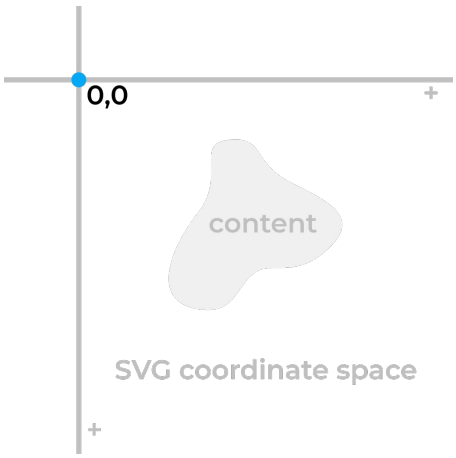
1in = 96

1cm = 37.795

1pt = 1.333

Relative:

1em = current font size



How do units work in SVG?

An SVG has an abstract coordinate system, measured in arbitrary units called "SVG units" or "user units". It is a consistent, Cartesian coordinate space (positive down and to the right) that eventually gets mapped to some real world space. The dimensions/positions/etc of all elements are given in this coordinate space.

It is possible to specify dimensions/positions/etc in terms of "real world units" like inches, but it is typically not advisable. Real world units will be converted to SVG units based on constants defined in the SVG standard, and the resulting actual size of the element will be affected by the `viewBox`, `width`, and `height` attributes, and may not actually end up as the size you intended.

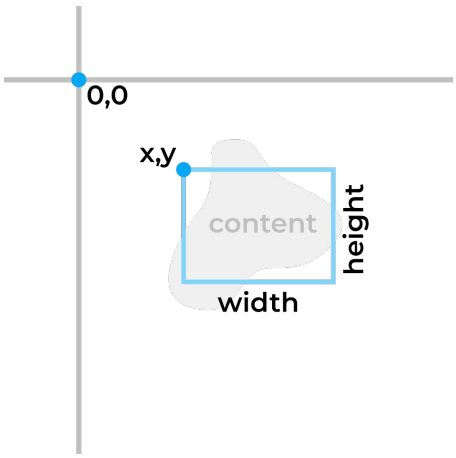
Bottom line: write SVG without units. It is standard practice, and most SVG editing software seems to generate SVGs in this manner by default. It is also in line with the main purpose of SVGs, which is to create images that are independent of actual size.

Reference:
https://oreillymedia.github.io/Using_SVG/guide/units.html

viewBox

```
<!-- format -->
viewBox="x y width height"

<!-- example -->
viewBox="70 60 100 75"
```



How do we define what is visible in the SVG?

The viewbox is the window into the SVG's coordinate space, and defines the boundaries of the image. You can think of it like a camera or a frame. You specify the x/y coordinates of the upper left corner and the width/height of the viewbox, in SVG units.

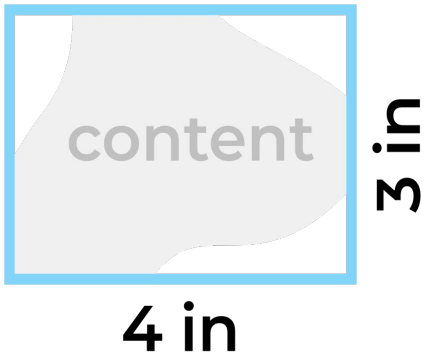
Contents of the SVG can extend beyond the viewbox, which you may or may not be able to see depending on the software and the overflow attribute (more on this later).

`viewBox` should always be specified; weird things can happen if it isn't.

Width and height

```
<!-- format -->
width="..." height="..."

<!-- example -->
width="4in" height="3in"
```



How do we size our image to real world units when we have to?

The `width` and `height` attributes specify how large the image should appear in its final context. Along with `viewBox`, it essentially defines a mapping from SVG units to real world units.

This is the only place in your SVG where you can specify real world units. If no unit is specified, they are interpreted as pixels.

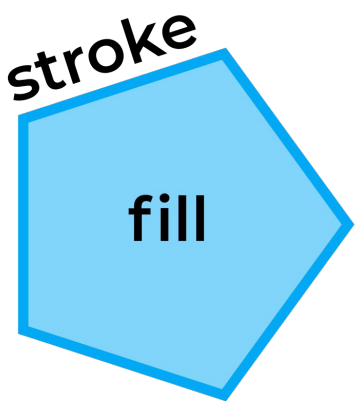
If these attributes are omitted entirely, default behavior will vary from software to software. Browsers will usually scale the image to fit the dimensions of its container. Some software may make them the same as the viewbox width/height in pixels.

In practice, it is usually more useful to not hard-code these attributes into the SVG, and to simply scale the image in situ to the needed size (eg, in CSS for a webpage, or in Inkscape before rendering as a PNG). As such, the minimum/boilerplate code to form a valid SVG is an `<svg>` element with the `xmlns` and `viewBox` attributes.

Stroke vs fill

```
<!-- format -->
fill="..." stroke="..."

<!-- example -->
fill="skyblue" stroke="blue"
```

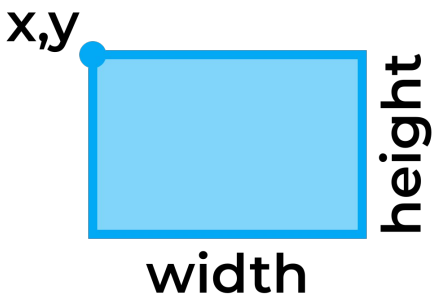


Before getting into drawing basic shapes, it is necessary to understand the `stroke` and `fill` attributes. The stroke is the outline of a shape, and the fill is the area within a shape. Both attributes can be set to a color, or to `none` to be disabled. For now, specify colors as just regular English color words, like "red" or "violet". Even less common color words like "forestgreen" or "chartreuse" can usually be used.

By default, SVG shapes have `fill="black"` and `stroke="none"`; even shapes that are intended to be just strokes, like lines. You will likely have to override this frequently.

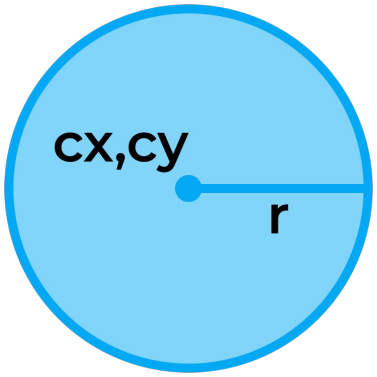
Rectangle

```
<rect
    x="..."
    y="..."
    width="..."
    height="..."
/>
```



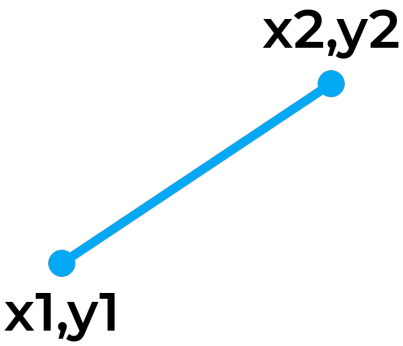
Circle

```
<circle
    cx="..."
    cy="..."
    r="..."
/>
```



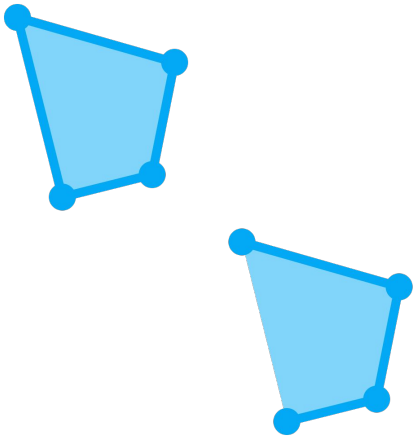
Line

```
<line
    x1="..."
    y1="..."
    x2="..."
    y2="..."
/>
```



Polygon / polyline

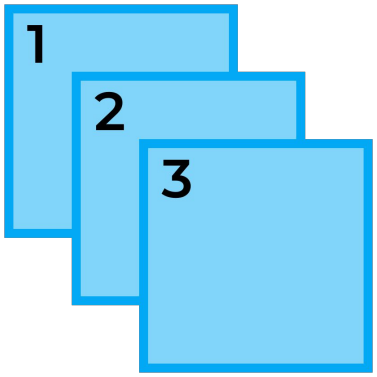
```
<polygon/polyline
   points="... x y x y ..."
/>
```



A `<polygon>` element is intended for closed shapes, where the last point is automatically connected to the first. A `<polyline>` element is intended for multi-segment lines (open shapes), and is not automatically closed. For the `points` attribute, specify a sequence of x/y coordinates, separated by single space or comma.

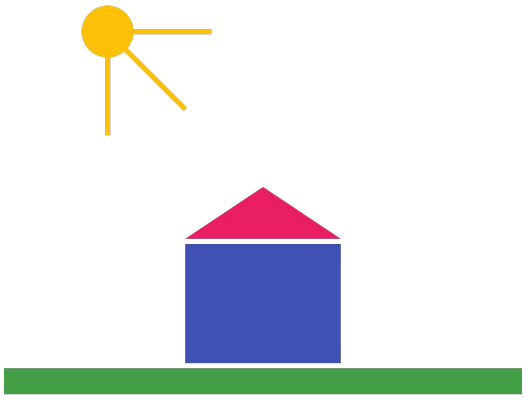
Z-order

```
<!-- 1 -->
<rect ... />
<!-- 2 -->
<rect ... />
<!-- 3 -->
<rect ... />
```



Elements are stacked in the order they appear in your SVG document. Later defined elements are stacked on-top/in-front of earlier defined elements.

Exercise 1



Recreate this SVG on your own using the techniques covered so far. The exact colors, lengths, and dimensions are not important; just try to capture the basic picture.