

25

SVG (SCALABLE VECTOR GRAPHICS)

OVERVIEW

- **Shape elements in SVG**
- **Clipping and masking**
- **Filter effects**
- **Styling SVGs**
- **Interactivity and animation**
- **SVG tools**
- **Production tips**
- **Responsive SVGs**

Intro to SVG

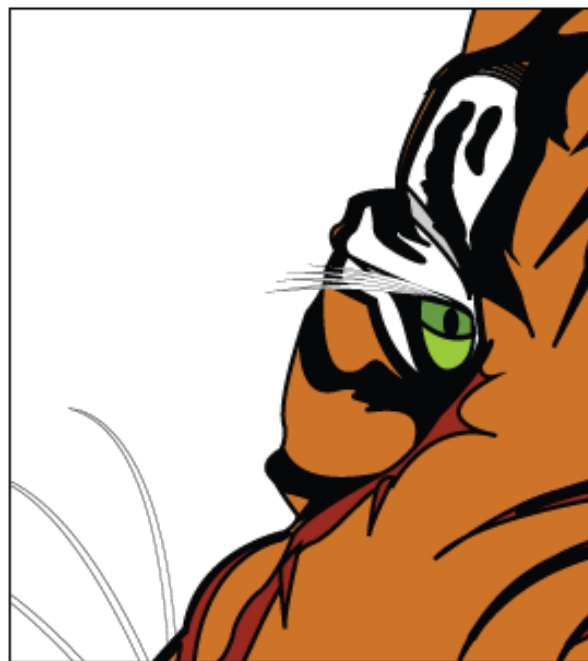
SVG (Scalable Vector Graphics) is a **vector** image format.

Vector images, which are defined by coordinates and lines, can be resized without loss of quality.

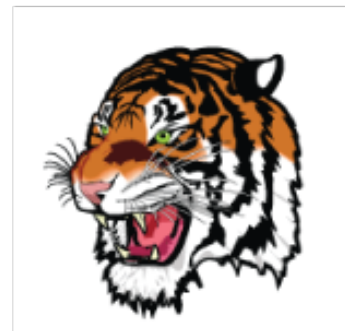
tiger.svg



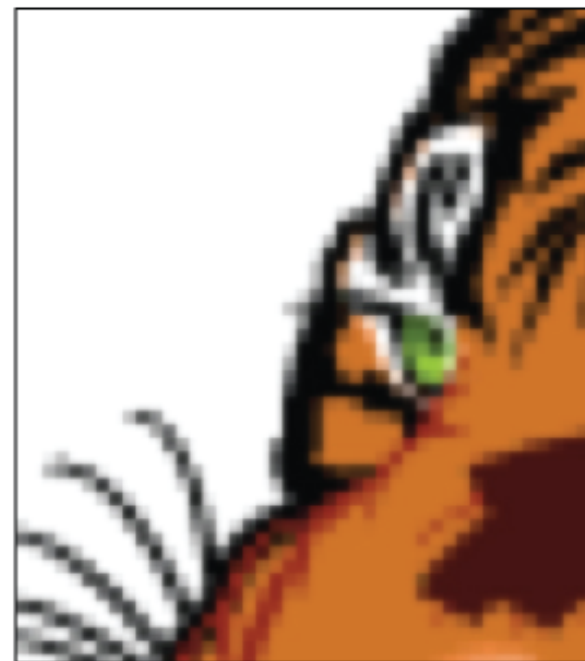
10x



tiger.png

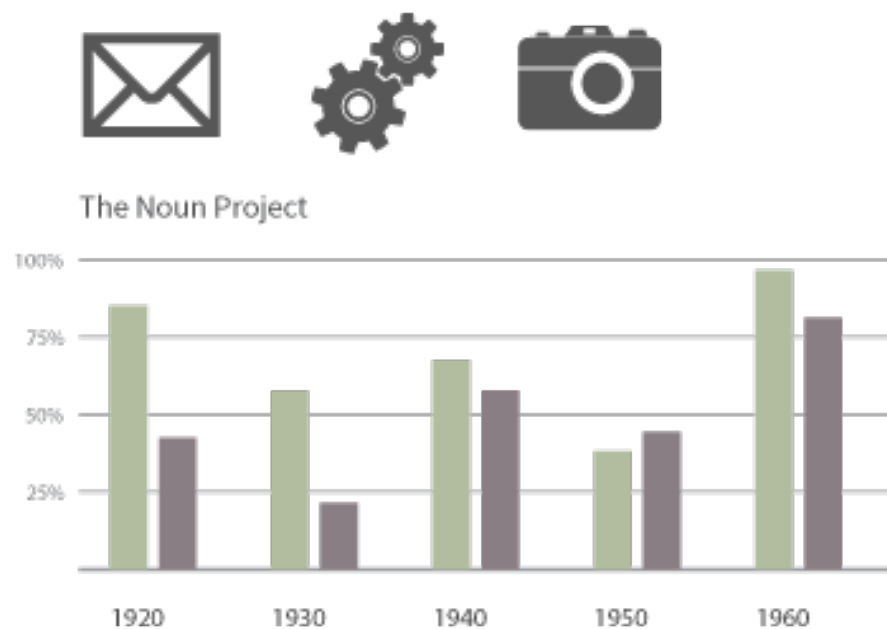


10x



Intro to SVG (cont'd)

- SVG is good for line drawings like icons, logos, charts, etc.
- The file size of an SVG is often significantly smaller than the same image in a bitmapped format.



SVG as an XML Language

- SVG is an **XML language for describing two-dimensional graphics** including paths, shapes, text, and filter effects.
- An SVG is just a text file that has been marked up with SVG elements and styles.
- Because they are text files with their own DOM, elements in **SVGs can be accessed and manipulated with CSS and JavaScript.**

Intro to XML

XML (eXtensible Markup Language) is a set of rules for creating other markup languages.

A few web-related languages written in XML:

- **XHTML:** HTML with the stricter syntax rules of XML
- **RSS (Really Simple Syndication):** Shares content as data to be read with RSS feed readers
- **MathML:** Used to describe mathematical notation
- **SVG:** Scalable Vector Graphics

Intro to XML (cont'd)

XML has strict syntax requirements (so languages can be used together in a single document), including the following:

- Element and attribute names must be lowercase.
- All elements must be closed (terminated). Empty elements are closed with a slash before the closing bracket: **
**.
- Attribute values must be in quotation marks with no extra white space.
- All attributes must have explicit values (example: **checked="checked"** instead of simply **checked**).
- Proper nesting is strictly enforced.
- Special characters must be represented by numeric character entities.
- Scripts must be contained within a **CDATA** section.

SVG Elements

The SVG language includes **elements for 2-D graphics**, including the following:

- Lines and shapes (**circle**, **rect**, **ellipse**, **path**, **line**, **polyline**, and **polygon**)
- A **text** element for text content
- Organization elements: **g** for grouping elements, **use** and **symbol** for reusing drawing
- Elements for clipping (**clipPath**) and masking (**mask**) images
- Elements for raster effects (**linearGradient** and **filter**)

Sample SVG Document

This SVG image is drawn with the following SVG document (*simple.svg*):

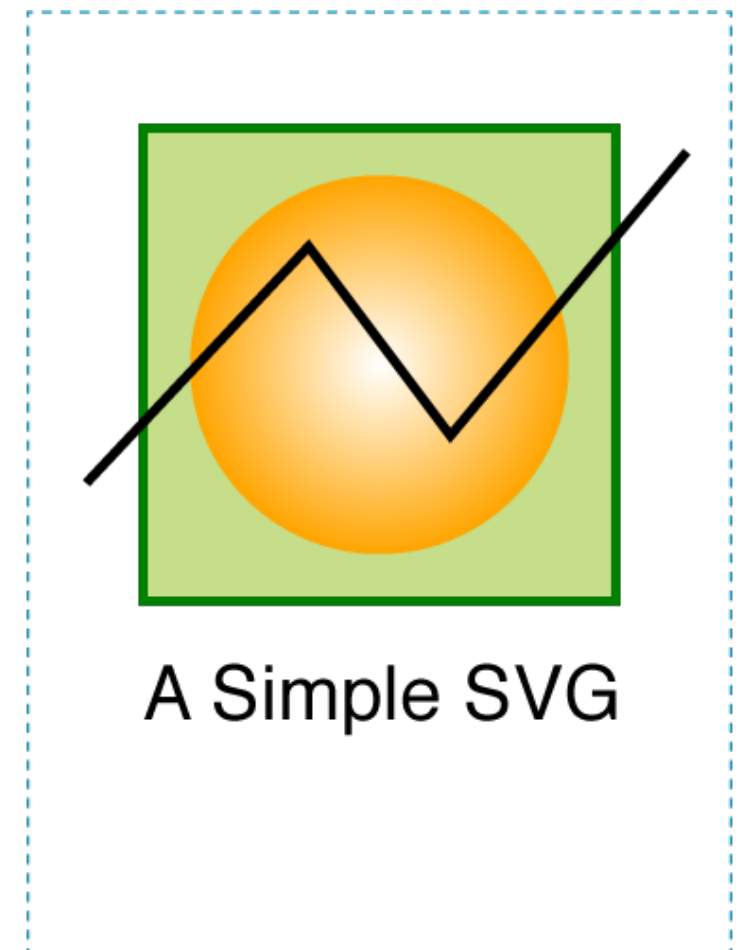
```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="150" height="200" viewBox="0 0 150 200">

  <defs>
    <radialGradient id="fade">
      <stop offset="0" stop-color="white"/>
      <stop offset="1" stop-color="orange"/>
    </radialGradient>
  </defs>

  <g id="greenbox">
    <rect x="25" y="25" width="100" height="100" fill="#c6de89"
      stroke-width="2" stroke="green"/>
    <circle cx="75" cy="75" r="40" fill="url(#fade)"/>
    <path d="M 13 100 L 60 50 L 90 90 L 140 30" stroke="black"
      stroke-width="2" fill="none"/>
  </g>

  <text x="25" y="150" fill="#000000" font-family="Helvetica"
    font-size="16">A Simple SVG</text>

</svg>
```



NOTE: The code will be discussed in sections over a series of slides.

Sample SVG Document (cont'd)

```
<?xml version="1.0" encoding="utf-8"?> (A)
<svg version="1.1" (B)
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  width="150" height="200" viewBox="0 0 150 200"> (C)
...
```

- (A) An XML declaration identifying the file as XML and identifying the character encoding
- (B) The root element for SVG files is **svg**. The version number is added by most drawing programs but is not required. The **xmlns** attributes declare the **namespace**, which tells the browser to use the SVG element vocabulary. The **xlink** attribute allows links and references to external files.
- (C) **width** and **height** attributes establish the **viewport** (the window in which the SVG is displayed).

Sample SVG Document (cont'd)

```
...  
<defs> (D)  
  <radialGradient id="fade"> (E)  
    <stop offset="0" stop-color="white" />  
    <stop offset="1" stop-color="orange" />  
  </radialGradient>  
</defs>  
...
```

(D) The **defs** element defines elements that will be referenced later by their **id** values. Here it's used to define a radial gradient that could be reused throughout a document.

(E) The **radialGradient** element contains two color **stop** elements, one for white, one for orange.



A Simple SVG

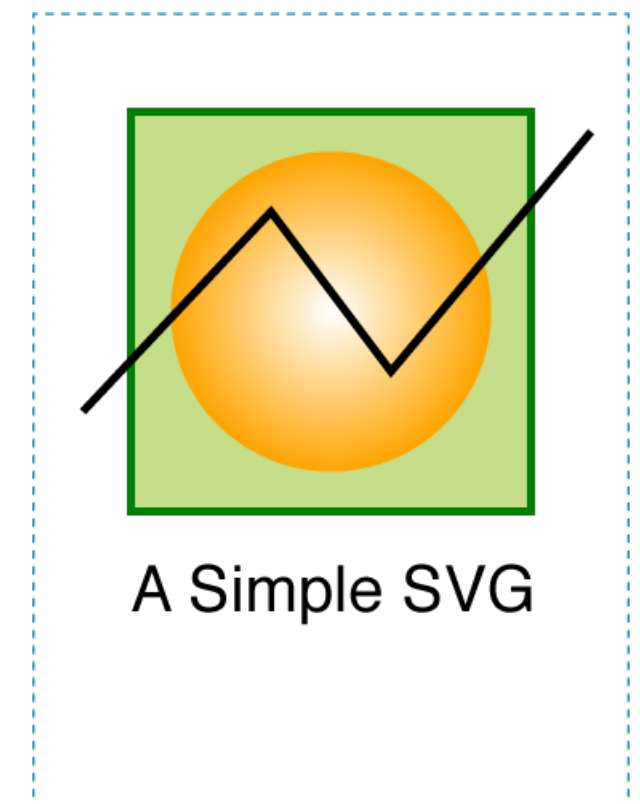
Sample SVG Document (cont'd)

```
...  
<g id="greenbox"> (F)  
  <rect x="25" y="25" width="100" height="100"  
fill="#c6de89" stroke-width="2" stroke="green"/> (G)  
  <circle cx="75" cy="75" r="40" fill="url(#fade)"/>  
  <path d="M 13 100 L 60 50 L 90 90 L 140 30" stroke="black"  
stroke-width="2" fill="none"/> (H)  
</g>  
...
```

(F) The shape elements **rect**, **circle**, and **path** are grouped with the **g** element and given the name **greenbox**.

(G) The **rect** element is given a size, fill, and stroke with attributes. The **circle** is assigned the "fade" radial gradient defined earlier. Empty elements are terminated.

(H) The crooked line is defined by **path**. The **d** (data) attribute provides x,y coordinates. The path starts with **M** (move to), and each **L** draws a "line to" the next set of coordinates.



Sample SVG Document (cont'd)

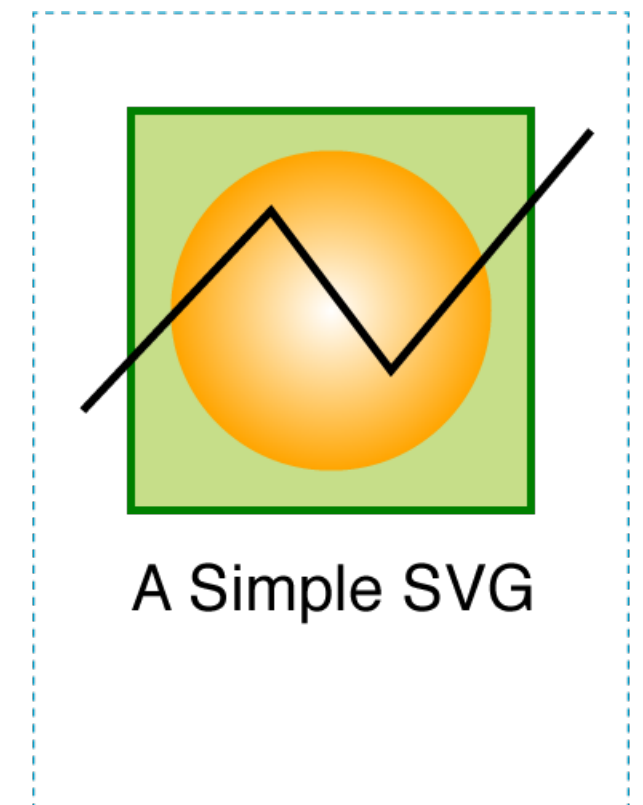
...

```
<text x="25" y="150" fill="#000000" font-family="Helvetica"
font-size="16">A Simple SVG</text> (I)
```

```
</svg>
```

(I) The **text** element contains the text. Style attributes set the font size and font family. There are many similarities between SVG attributes and CSS styles

The root **svg** element is closed at the end of the document.



Embedded Bitmap Images

SVGs may contain **embedded bitmapped content**:

```
<image xlink:href="starrysky_600.jpg" x="45" y="0"  
width="100" height="150"/>
```

You might do this to add a behavior or interactivity that a JPEG or PNG can't do on its own.

Clipping

The **clipPath** element defines a vector path that cuts out a section of the image. Parts outside of the path are hidden:

```
<defs>
  <clipPath id="star">
    <polygon points="390,12 440,154 590,157 470,250 513,393 390,307 266,393
310,248 189,157 340,154 390,12" style="fill: none"/>
  </clipPath>
</defs>
<image xlink:href="starrysky_600.jpg" width="600" height="400"
  style="clip-path: url(#star)"/>
```



The star-shaped path positioned over the image.



The image is clipped to the path (dotted border indicates the SVG viewport but is not part of the SVG).

Masking

Masking is like clipping, but it uses a bitmapped overlay (a gradient or another image) to determine the opacity levels of the image. A mask is created with the **mask** element:

```
<defs>
  <linearGradient id="blend">
    <stop offset="0%" stop-color="#ffffff" />
    <stop offset="100%" stop-color="#000000"/>
  </linearGradient>

  <mask id="star" x="0" y="0" width="400" height="381">
    <polygon points="390,12 440,154 590,157 470,250
513,393 390,307 266,393 310,248 189,157 340,154 390,12"
style="fill: url(#blend)" />
  </mask>
</defs>
```

```
<image xlink:href="starrysky_600.jpg" width="600"
height="400" style="mask: url(#star);" />
```



The star-shaped path is filled with a gradient.



The gradient works as a mask in which the light areas allow more of the image to show through. The darker the mask, the lighter the masked image.

Filter Effects

- SVG includes over a dozen Photoshop-like **filter effects** for manipulating images.
- Effects can be used alone or combined.
- The original image is untouched because effects are applied when the image is rendered.



Original image



Gaussian blur



Color matrix: saturate



Morphology



Turbulence + Displacement map

Reusing SVG Drawings

Define a shape once and reuse it throughout the document with the **symbol** element. Content in **symbol** elements does not get rendered:

```
<symbol id="iconA" viewBox="0 0 100 100">  
  <!-- all the paths and shapes that make up the icon -->  
</symbol>
```

When you want to use the symbol, call it with the **use** element that triggers it to render in place:

```
<svg class="icon">  
  <use xlink:href="#iconA" />  
</svg>
```

NOTE: You can create SVG sprites with **symbol** and use them to reduce the number of calls to the server for SVG images.

Styling SVGs

There are four ways to apply styles to the elements in an SVG:

- **Presentation attributes**

Attributes defined in the SVG language:

```
<rect x="25" y="25" width="100" height="100"  
fill="#c6de89" stroke-width="2" stroke="green" />
```

- **Inline styles**

With the inline **style** attribute (similar to HTML):

```
<rect x="25" y="25" width="100" height="100"  
style="fill:#c6de89; stroke-width:2; stroke:green;" />
```

Styling SVGs (cont'd)

- **Internal style sheet**

A **style** element at the top of the **svg** source document:

```
<svg>
  <style>
    /* styles here */
  </style>
  <!--drawing here -->
</svg>
```

- **External style sheet**

Importing a .css document with an @import rule in the SVG source for inline SVGs or SVGs placed with **object** or **iframe** elements:

```
<svg>
  <style type="text/css">
    @import "svg-style.css";
    /* more styles */
  </style>
  <!-- svg code here -->
</svg>
```

NOTE: Inline **svg** elements can also be styled by style sheets linked to the HTML source.

JavaScript and SVG

- SVGs can be scripted with JavaScript because all of its element and attribute nodes are accessible in the DOM.
- This allows events like loading and mouseovers to trigger changes in the SVG.
- Effects range from adding a little motion to a UI element to creating complex game interfaces.
- For inline SVGs, scripts must be contained in an **XML Character Data Block**:

```
<script><![CDATA[  
    /* script here... */  
]]></script>
```

Animation

SVG can be used to add animation effects to a web page. There are several options for animating SVGs:

- **JavaScript**

JavaScript is currently the most reliable way to create complex animations in SVG.

- **CSS Animation**

SVG elements can be animated with CSS transitions and keyframes. Browser support is currently spotty, and you can't animate SVG attributes.

- **SVG/SMIL**

There are animation effects built into SVG based on the SMIL XML language (Synchronized Multimedia Integration Language), but they're not well supported.



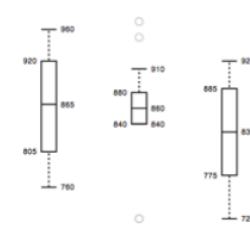
Animated SVG by Chris Gannon

Data Visualization

SVG is a popular format for data visualization because images can be generated dynamically with real data.

Example: Make the temperature level on an SVG thermometer rise and fall with real weather data.

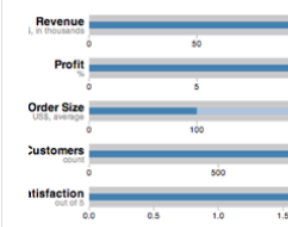
Box Plots



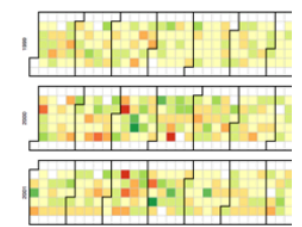
Bubble Chart



Bullet Charts



Calendar View



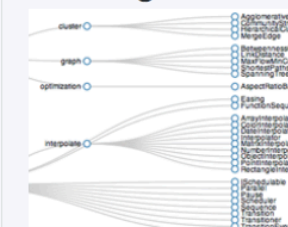
Non-contiguous Cartogram



Chord Diagram



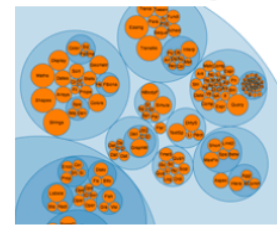
Dendrogram



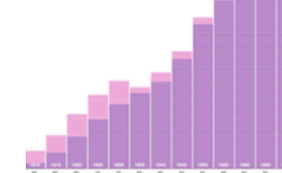
Force-Directed Graph



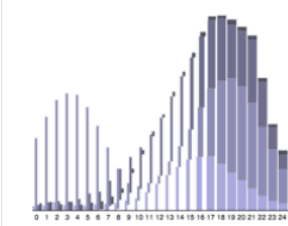
Circle Packing



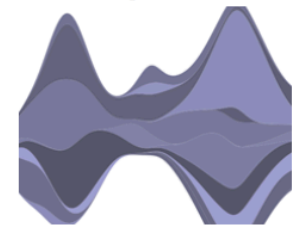
Population Pyramid 2000



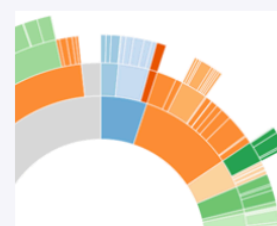
Stacked Bars



Streamgraph



Sunburst



Node-Link Tree



Treemap



Voronoi Diagram



SVG Tools

You can write SVG by hand, but having it exported from a drawing program is easier. Some SVG tools include

- Adobe Illustrator (adobe.com)
- Inkscape ([free from inkscape.org](https://inkscape.org))
- Boxy (boxy-svg.com)
- SVG-Edit (github.com/SVG-Edit/svgedit)
- Interface design tools like Sketch, AdobeXD, and Affinity Designer

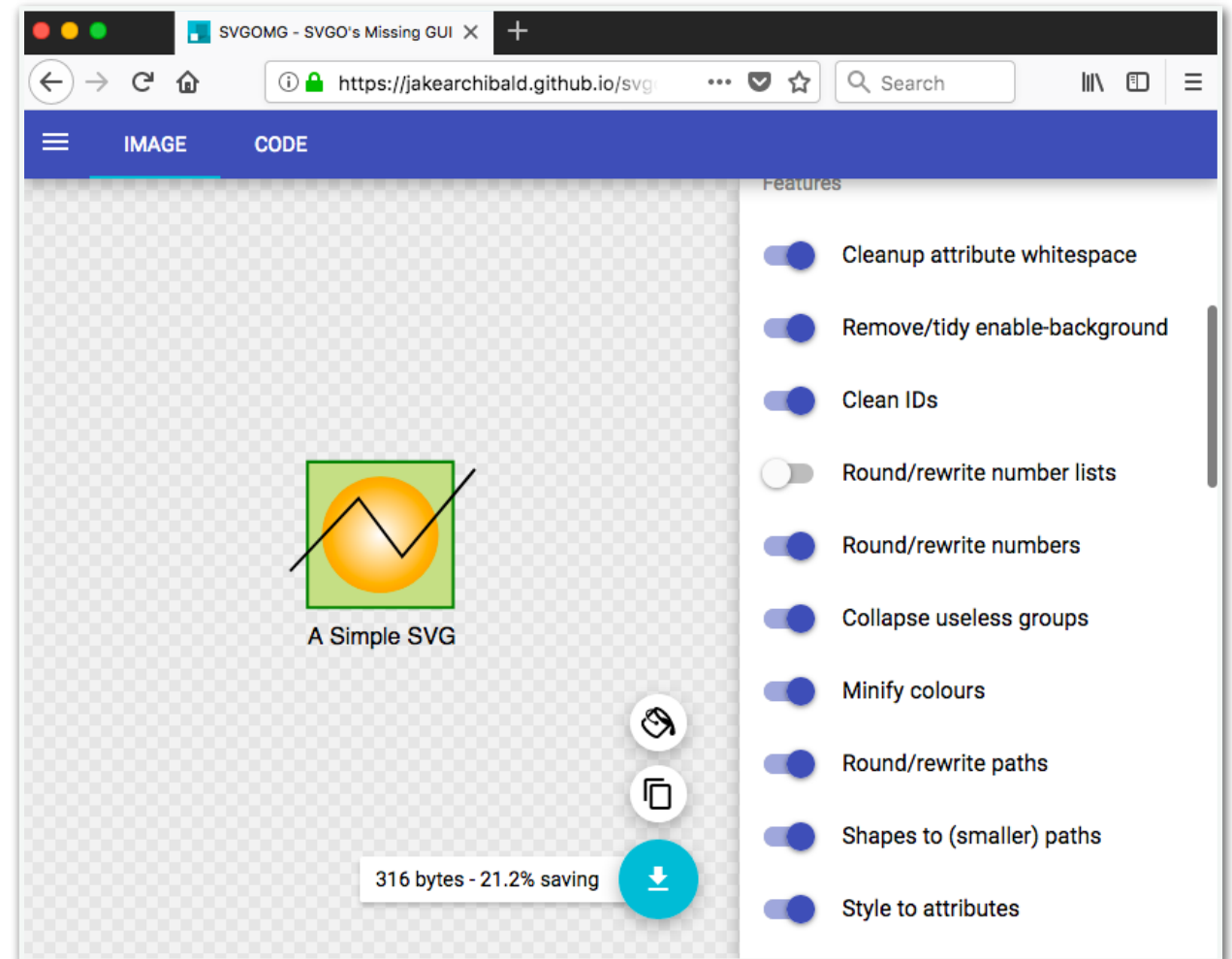
SVG Production Tips

When designing, follow these tips for optimizing SVG images before exporting them:

- Define the dashboard or drawing size in pixels.
- Use layers to group elements logically.
- Give elements and layers meaningful names.
- Simplify paths.
- Be aware of decimal places (more decimals = larger files).
- Avoid raster effects.
- Use centered strokes.
- Pay attention to fonts (external web fonts may not load).

Optimizing SVGs with SVGO

- Image editing programs usually export bloated code, so **SVGs should be optimized with the SVGO tool after export** (github.com/svg/svgo).
- There are SVGO plug-ins for many graphics programs and Grunt or Gulp task runners.
- **SVGOMG** (shown on right) provides a graphical interface for SVGO (github.io/svgomg).



File Compression

SVG files can be compressed with text-compression utilities:

- **Gzip**

A utility on the server that compresses text files with algorithms. A gzipped SVG uses the suffix .svgz.

- **Brotli**

An open source compression algorithm from Google

Responsive SVGs

- SVGs are great for responsive layouts because they can resize larger or smaller without any loss of detail.
- The SVG must be marked up in a way to preserve its **aspect ratio** (the ratio of its width to height) when the image resizes.
- This requires an understanding of the **viewport** and the **viewbox**.

Responsive SVGs (cont'd)

The viewport

- Defined by **width** and **height** attributes in the **svg** element:

```
<svg width="400" height="500">  
  <!-- drawing content here -->  
</svg>
```

- It's like a window through which you view the drawing area.
- The viewport coordinate system starts at 0 in the top left corner and increases to the right and downward.

Responsive SVGs (cont'd)

The viewBox

- The viewBox controls the **dimensions of the drawing itself** in the user space.
- It is defined with the **viewBox** attribute. The x and y values determine the position of the top left corner in the viewport:

`viewBox="x y width height"`

- In this example, the viewBox matches the viewport:

```
<svg width="400" height="500" viewBox="0 0 400 500">  
<!-- drawing content here -->  
</svg>
```

- **The viewBox preserves aspect ratio by default.**

Responsive SVGs (cont'd)

For SVGs embedded with `img` or `object`:

- Make sure the **svg** element includes the **viewBox** attribute.
- Do not include **width** and **height** if you want the SVG to fill the width of its container.
- Set the **width** and **height** of the **img** or **object** element to fill the width of its containing element (see **NOTE**):

```
img {  
    width: 100%;  
    height: auto;  
}
```

NOTE: This approach is necessary to support Internet Explorer. Modern browsers don't require **width** and **height** attributes for proportional sizing.

Responsive SVGs (cont'd)

The SVG markup (flowers.svg):

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" viewBox="0 0 160 120">
  <!-- flower drawing -->
</svg>
```

The HTML markup:

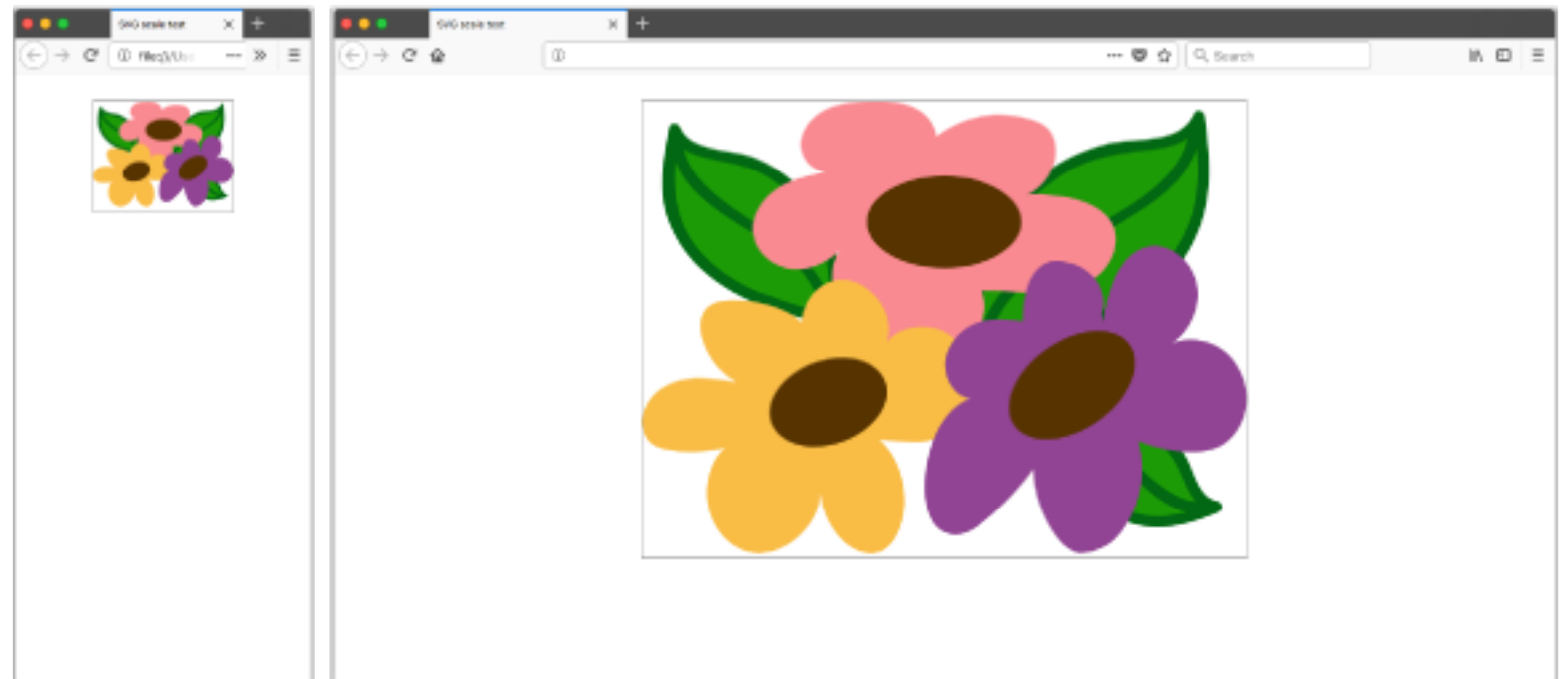
```
<div class="container">
  
</div>
```

The SVG image scales with
the .container div:

The styles:

```
.container {
  width: 50%;
  outline: 1px solid gray;
  margin: 2em auto;
}

/* IE fix */
img {
  width: 100%;
  height: auto;
}
```



Responsive SVGs (cont'd)

For inline SVGs (added with the `svg` element):

Inline SVGs scale proportionally in modern browsers, but old browsers and Internet Explorer require an elaborate "padding hack":

- Include **viewBox** and omit **width** and **height** attributes on the **svg** element.
- Put the **svg** in a container **div** and set its height to 1 pixel.
- Apply an amount of **padding** to the top of the **div** to give it the same aspect ratio as the SVG image.
- Once the **div** is expanded with padding, position the **svg** element in the top left corner of the **div**.

Responsive SVGs (cont'd.)

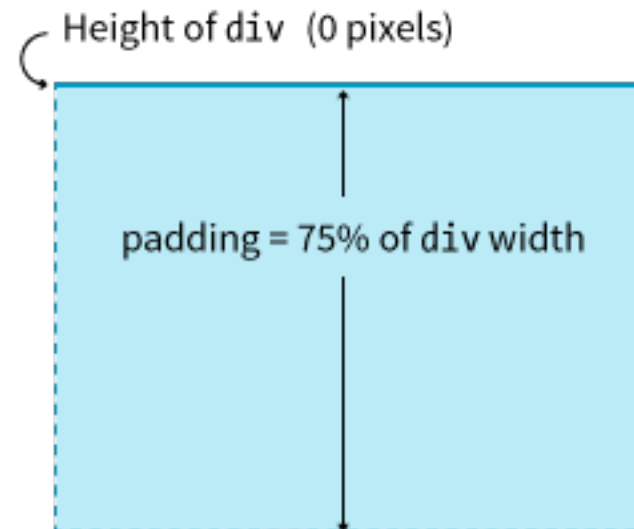
The markup

```
<div class="container">
  <svg version="1.1"
  viewBox="0 0 160 120">
    /* drawing contents */
  </svg>
</div>
```

The styles

```
.container {
  width: 100%;
  height: 0;
  padding-top: 75%;
  /* (120/160)*100% */
  position: relative;
}
```

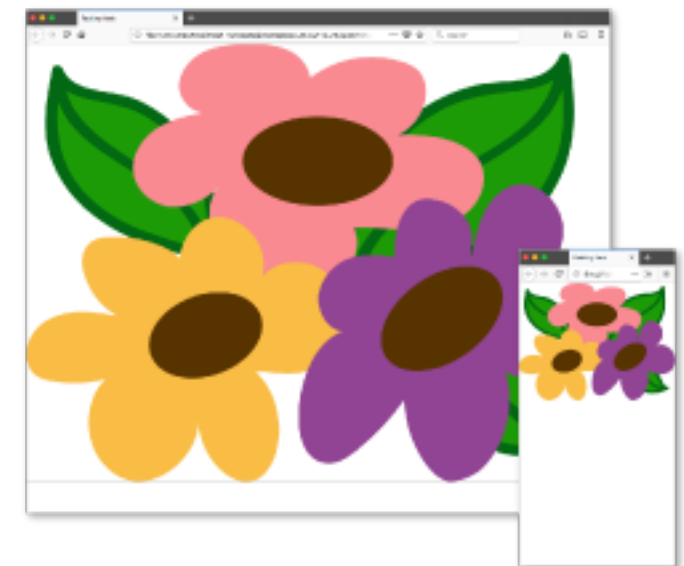
```
svg {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```



svg element gets pushed down.



svg is absolutely positioned in top-left corner of div.



The container is set to 100% width, and the svg scales with it.