Using SVG in the Web Browser

Aumüller Thomas, Heider Martin, Ramadan Abdelrahman

706.057 Information Visualisation 3VU SS 2024
Graz University of Technology

15 May 2024

Abstract

Scalable Vector Graphics (SVG) are a powerful option to display two-dimensional graphics and images. They support animations, and a high degree of interactivity, whilst also being scalable without loss of quality. These and many more features put SVGs in a quite unique spot when dealing with icons, graphics, and images on the web. Although they are already well-known and widely used, SVGs have been somewhat opaque and convoluted in the way they can be and should be used. The aim of this survey is to shine a light on some of the most important aspects of SVG graphics, their attributes, and how they are validated and also take a closer look at how they are used in web browsers.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

This survey paper takes an in-depth look at Scalable Vector Graphics (SVG) and how they are used in web browsers. One of the most recurring annoyances while using the web is the resolution and scaling of videos and images. Many people are familiar with the widley used image file formats JPEG and PNG. Unfortunately, not as many people are aware of Scalable Vector Graphics (SVGs), which resolve many annoyances and problems. For many years, use of SVGs in web browsers was rather opaque, because of a multitude of different combinations of standards and support.

SVGs are incredibly powerful two-dimensional graphics and, thanks to their XML-based nature, they posess a multitude of quality-of-life features such as interactivity, scalability without loss of quality, and animation. In addition to these main features, SVGs are searchable, indexable, scriptable, compressible and most web browsers support and render SVGs. To create or edit these immensely powerful two-dimensional graphics, one only has to open any text editor, write simple SVG code like that in Listing 1.1, and save it as an SVG file with the extension `.svg`. It can be displayed by opening the file in any modern web browser. SVGs can also be embedded inside HTML files, as can be seen in Listing 1.2. The output is shown in Figure 1.1.

```
<svg width="10rem" height="10rem">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 1.1: Basic SVG code to draw a red circle.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="sizing.css">
</head>
<body>

<svg width="10rem" height="10rem">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>

</body>
</html>
```

Listing 1.2: Basic SVG code to draw a red circle, embedded inside an HTML file.

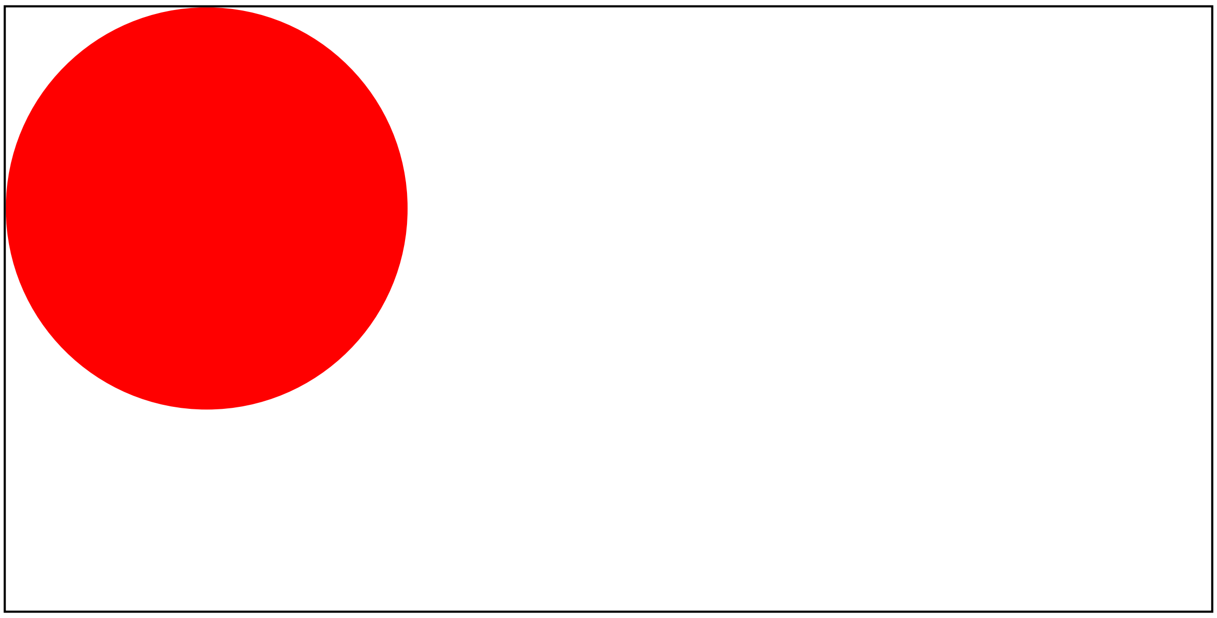

Figure 1.1: The output of Listing 1.2, showing a basic red circle. [Drawn by Thomas Aumüller.]

Chapter 2

Including SVG

2.1 Inline `<svg>` Element

The most straightforward way to include an SVG graphic is by embedding an `<svg>` element directly within an HTML file, as can be seen in Listing 2.1. SVG 1.1 [W3C 2011] is supported by all modern browsers. Some browsers support parts of SVG 2 [W3C 2023].

SVG content embedded directly within the HTML document becomes part of the page DOM allowing for easy editing and manipulation of SVG elements using HTML and CSS. Additionally, SVGs included in this way can be cached by the browser, reducing the number of server requests and improving page loading performance [Buckler 2021].

The main disadvantage is that the SVG must be embedded into every page which requires it, cluttering the HTML file. This can make the document structure more complex and harder to maintain, especially for larger projects. Embedding SVG content directly in HTML increases the size of the HTML file, especially for large or complex SVGs, leading to longer loading times [Buckler 2021].

2.2 `<img>` Element with External SVG File

SVGs can also be added to a a web page like any other image via the `<img>` element and specifying an external SVG file. Attributes like `width`, `height`, and `alt` can also be added, as can be seen in Listing 2.2.

Including SVGs via external files makes the HTML code easier to read and maintain. Furthermore, the browser can cache external SVG files, increasing performance if the same graphic is used in multiple places. The HTML `<img>` element is well-supported across all modern browsers ensuring consistent rendering of the SVG content.

The biggest trade-off when including SVGs with the `<img>` element from an external file, is that the browser treats the SVG like any other regular image. For security reasons, any scripting, external styling via CSS, links, and other SVG interactivity is completely disabled [Buckler 2021].

2.3 `<img>` Element with Data URI

The `<img>` element can also be used with a data URI rather than pointing to an external file. Encoding an SVG inside a data URI is practical for including smaller regularly used SVGs, as can be seen in Listing 2.3.

Using data URIs, small files can be embedded directly into HTML files, reducing the number of HTTP requests and therefore speeding up load times. Linking to external resources always comes with a risk the resources might be moved or deleted, resulting in a broken link. Data URIs circumvent this risk because the data is embedded directly into the document.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<!-- Inline SVG -->
<svg>
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>

</body>
</html>
```

Listing 2.1: An `<svg>` element embedded inline within an HTML file.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<!-- Using img with svg file -->
<img width="10rem" height="10rem" src="file.svg"/>

</body>
</html>
```

Listing 2.2: An `<img>` element with an external SVG file used within an HTML file.

However, the embedded resource cannot be cached separately, and is repeated inline wherever it is needed, increasing the size of the HTML file.

2.4 CSS Background Image

HTML elements can be assigned a background image through CSS. SVGs can be used as a CSS background image, either by specifying an external SVG file, or encoded directly in a data URI. Listing 2.4 illustrates the former and Listing 2.5 the latter.

CSS provides a wide array of styling options for SVG background images, including positioning, sizing, repeat behaviour, blending modes, and many more. This allows for greater flexibility in styling and integrating SVG graphics into web pages. SVG background images can be interactive through CSS pseudo-elements or JavaScript event handling. Developers can add hover effects, animations, or interactive elements to SVG backgrounds, enhancing user engagement.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<!-- Using img with data URI -->
<img src="data:image/svg+xml,%3Csvg xmlns="http://www.w3.org/2000/svg"
  viewBox="0 0 100 100"%3E%3Ccircle cx="50" cy="50"
  r="50" fill="red" /%3E%3C/svg%3E"/>

</body>
</html>
```

Listing 2.3: An <img> element with an SVG fgraphic included inline as a data URI.

```
.demo {
  width: 10rem;
  height: 10rem;
  background-image: url('file.svg')
}
```

Listing 2.4: CSS background image with an external SVG file.

```
.demo {
  width: 10rem;
  height: 10rem;
  background-image: url('data:image/svg+xml,%3Csvg
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100"%3E
%3Ccircle cx="50" cy="50" r="50" fill="red" /%3E%3C/svg%3E')
}
```

Listing 2.5: CSS background image with SVG encoded in a data URI.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Injecting SVG into DOM</title>
</head>
<body>

<!-- SVG DOM injection -->
<div id="svg-container">
</div>

<script>
// Create SVG element
var svgNS = "http://www.w3.org/2000/svg";
var svg = document.createElementNS(svgNS, "svg");
svg.setAttribute("width", "200");
svg.setAttribute("height", "200");

// Create SVG circle element
var circle = document.createElementNS(svgNS, "circle");
circle.setAttribute("cx", "100");
circle.setAttribute("cy", "100");
circle.setAttribute("r", "50");
circle.setAttribute("fill", "red");

// Append circle to SVG
svg.appendChild(circle);

// Append SVG to container
var container = document.getElementById("svg-container");
container.appendChild(svg);
</script>

</body>
</html>
```

Listing 2.6: Injecting SVG elements into the page DOM with JavaScript.

Styling SVG background images with CSS may introduce complexity, especially for intricate graphics or animations. Achieving desired effects or fine-tuning styles may require advanced knowledge of CSS and SVG.

2.5 SVG-DOM Injection

The most advanced way to include SVGs into a web page it to inject SVG elements directly into the web page's DOM via JavaScript (SVG-DOM). An example can be seen in Listing 2.6.

JavaScript injection allows for the dynamic creation and manipulation of SVG elements based on user interactions, data changes, or other events. This enables dynamic and interactive visualizations on web pages. Developers can programmatically create complex SVG graphics, animations, or interactive elements tailored to specific requirements. Injecting SVG on demand can help optimize performance by reducing initial page load times. For example, SVG elements can be generated on demand and injected only when needed.

Generating SVG content with JavaScript can be complex, especially for intricate graphics or animations. Actually handling SVG attributes, transformations, and event handling programmatically requires advanced knowledge of both SVG and JavaScript. Dynamically injected SVG may pose accessibility challenges compared to conventional static SVG content. Proper handling of accessibility features such as alt text, ARIA attributes, and keyboard navigation requires additional effort. DOM injection comes with the highest complexity and maintenance effort of all options for including SVGs.

2.6 Deprecated Options

For completeness, here are some of the outdated and deprecated ways to include SVGs, in addition to the aforementioned possibilities:

- `<iframe>`: used to embed documents within the HTML document.
- `<embed>`: defines a container for an external resource.
- `<object>`: defines a container for an external resource.

All of these options can be used to embed SVG content into a web page, but the use of these options is discouraged [Buckler 2021; W3Schools 2024].

Chapter 3

Styling SVG

There are four ways to apply styleing to obejcts inside an SVG graphic, in precedence order: inline style, internal style sheet, external style sheet, and SVG attibutes directly. These can be seen in Listing 3.1. Precedence is applied in that order, i.e. an inline style attribute overrides all other kinds of styling.

3.1 Inline Style

A `style` attribute is applied directly to an object inside an SVG, as shown in Listing 3.2. Inline style applied to an object overrides all other styling.

3.2 Internal Style Sheet

Styles are defined using a `<style>` element within the SVG, as shown in Listing 3.3. Styles in the internal style sheet override those specified in an external style sheet or directly as SVG attributes. This method keeps styles within the SVG file, but allows for better organization compared to inline styles.

3.3 External Style Sheet

Styles are defined ouside of the SVG and are applied to SVG objects using CSS selectors. The styles might be in a `<style>` block, as shown in Listing 3.4, or in an external CSS file. This method promotes better maintainability and reusability of styles across multiple SVG files.

3.4 SVG Attributes

SVG elements have specific styling attributes such as fill, stroke, and stroke-width, which can be directly applied to control their appearance. This method provides fine-grained control over individual SVG elements, but has the lowest precedence.

```
<!-- Styling Precedence -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
rect {
  fill: purple;
  stroke: orange;
  stroke-width: 6;
}
</style>
</head>

<body>
<svg width="100" height="100">
<style>
  rect {
    fill: rgb(0, 128, 0);
    stroke: yellow;
    stroke-width: 4;
  }
</style>
<rect x="10" y="10" width="80" height="80"
  fill="cyan" stroke="black" stroke-width="3"
  style="fill: red; stroke: blue; stroke-width: 2;" />
</svg>

</body>
</html>
```

Listing 3.1: Applying inline style directly to an SVG object with the `style` attribute. Inline style applied to an object overrides all other styling.

```
<!-- Inline Style -->
<svg>
<rect x="10" y="10" width="80" height="80"
  style="fill: red; stroke: blue; stroke-width: 2;" />
</svg>

</body>
</html>
```

Listing 3.2: Applying inline style directly to an SVG object with the `style` attribute. Inline style applied to an object overrides all other styling.

```
<!-- Internal Style Sheet -->
<svg>
<style>
  rect {
    fill: rgb(0, 128, 0);
    stroke: yellow;
    stroke-width: 4;
  }
</style>
<rect x="10" y="10" width="80" height="80"
  fill="cyan" stroke="black" stroke-width="3"/>
</svg>
```

Listing 3.3: An internal style sheet, with a `<style>` element inside the SVG. The stroke width will be 4, since styles in an internal style sheet override those provided as SVG attributes.

```
<!-- External Style Sheet -->
<style>
  rect {
    fill: purple;
    stroke: orange;
    stroke-width: 6;
  }
</style>
```

Listing 3.4: An external style sheet uses a style block outside of the SVG, shown here, or styling in an external CSS file.

```
<!-- SVG Attributes -->
<rect x="10" y="10" width="80" height="80"
  fill="cyan" stroke="black" stroke-width="3"/>
```

Listing 3.5: Styling applied via SVG attributes directly on an object.

Chapter 4

Sizing SVG

SVGs can be sized in many different ways such that they fit neatly in the desired space or frame, rather than using the browser default each time.

4.1 Browser Defaults

In the beginning when SVG were integrated and supported by web browsers, there were two competing sets of browser defaults. Different browsers picked different combinations of these sizing defaults, which resulted in unpredictable behavior when no intrinsic sizing was given.

Fortunately, modern browsers have settled on more concrete and predictable defaults:

- *SVG embedded in HTML*: Default object size of `width=300px` and `height=150px`.
- *SVG file*: Attribute default =100%.
- *SVG with viewBox*: default `width=100%` and `height` according to aspect ratio.

4.2 Browser Default Sizing

In Listing 4.1, the SVG element does not specify any width or height attributes. Therefore, the SVG will be sized based on the current browser default, depending on whether the SVG is part of an HTML file or a standalone SVG file.

4.3 SVG Element Attributes

Here, in Listing 4.2, an SVG element embedded inside a HTML file specifies width and height attributes directly. Giving the SVG element a width and height overrides the browser default.

4.4 External Style Sheet

In Listing 4.3, the SVG element has a class attribute (`class="demo"`) that allows CSS styling for the SVG. The CSS rule `.demo` sets the width to `10rem` and height to `10rem` for SVG elements with this class. Setting width and height in an external style sheet will override the browser default and the SVG width and height attributes.

```
<!-- 1. Browser Default Sizing -->
<svg>
  <circle cx="50" cy="50" r="50" fill="red"/>
</svg>
```

Listing 4.1: If no sizing given at all, the browser default is used.

```
<!-- 2. SVG Element Attributes -->
<svg width="10rem" height="10rem">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.2: Width and height are specified as attributes of the SVG element.

4.5 Internal Style Sheet

In Listing 4.4, a `<style>` block is used inside the SVG element to define the sizing of the SVG. These CSS rules apply to SVG elements within the same SVG document. In this example, the CSS rule sets the width to `10rem` and height to `10rem` for all SVG elements. Assigning a size to the SVG via an internal style sheet overrides external style sheet sizing (at the same specificity), SVG sizing attributes, and the browser default.

4.6 Inline Style

The SVG element in Listing 4.5 has a style attribute to directly apply inline CSS styling. This sizing takes precedence over any other sizing attributes.

4.7 viewBox Only

In SVG, the `viewBox` attribute defines the coordinate system and aspect ratio of the SVG content, but does not directly affect its size on the web page. The viewBox specifies the internal coordinate system used for objects inside the SVG. These are then mapped to fit the viewport allocated by the web page [Bellamy-Royds et al. 2017].

The SVG element in Listing 4.6 specifies a viewBox attribute without any explicit width or height attributes. Without explicit width and height attributes, the SVG will scale to fit the available space within its container, while preserving the aspect ratio defined by the viewBox. Any content outside the viewBox will be clipped.

4.8 viewBox with Sizing

In Listing 4.7, the SVG element specifies both a viewBox and explicit width and height attributes. In this case, the viewBox determines the aspect ratio and internal coordinate system, while the width and height attributes determine the size of the SVG container on the web page.

```
<!-- 3. External Style Sheet -->

<svg class="demo">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>

// CSS File
.demo {
  width: 10rem;
  height: 10rem;
  border: 0.1rem solid black;
  background-color: lightgray;
}
```

Listing 4.3: Sizing via an external style sheet.

```
<!-- 4. Internal Style Sheet -->
<svg class="demo">
<style>
.demo {
  width: 10rem;
  height: 10rem;
  border: 0.1rem solid red;
}
</style>
<circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.4: Sizing via an internal `<style>` element inside the SVG.

```
<!-- 5. Inline Style -->
<svg style="width: 20rem; height: 20rem; background-color: lightblue;">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.5: Inline style applied directly to the SVG element with a `style` attribute.

```
<!-- 6. viewBox Only -->
<svg viewBox="0 0 100 100">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.6: An SVG element with a viewBox, but no other sizing.

```
<!-- 7. ViewBox with Sizing -->
<svg viewBox="0 0 100 100" width="20rem" height="20rem">
  <circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.7: An SVG element with a viewBox and width and height attributes.

4.9 Sizing Precedence

The sizing of SVGs follows the following precedence rules, highest precendence first:

1. Inline Style.
2. Internal Style Sheet.
3. External Style Sheet.
4. SVG Element Attributes.
5. viewBox (aspect ratio and scaling).
6. Browser Default Sizing.

This example in Listing 4.8 demonstrates the precedence of the aforementioned sizing methods for SVGs. The SVG element has a viewBox, width, height attributes, and inline style. The CSS rule inside the `<style>` element further modifies the sizing and styling. In this case, the inline style takes precedence over the width and height attributes, and the CSS rule takes precedence over all other sizing methods.

In summary, the precedence of sizing and styling for SVG elements follows the cascade of CSS styles, where inline styles take precedence over internal and external CSS rules, and explicit attributes like viewBox, width, and height can further influence the sizing and aspect ratio of SVG content.

```
<!-- Sizing Precedence -->

// Outside of SVG element
<style>
svg {
  width: 10rem;
  height: 10rem;
  border: 0.1rem solid black;
  background-color: lightgray;
}
</style>

<svg class="demo" viewBox="0 0 100 100" width="100" height="100"
  style="width: 10rem; height: 20rem; background-color: lightblue;">

<style>
  svg {
  width: 20rem;
  height: 10rem;
  border: 0.1rem solid black;
  background-color: lightgreen;
  }
</style>

<circle cx="50" cy="50" r="50" fill="red" />
</svg>
```

Listing 4.8: Sizing precedence.

Chapter 5

Animating SVG

Animations bring elements to life by creating dynamic and engaging visual effects. They can be used for various purposes such as highlighting important content, guiding users through a process, or simply adding visual appeal to a web site.

In particular, transitions can be used to smoothly change property values over a specified duration. They can enhance user experience by adding fluidity to UI elements, for instance by transitioning between different states of a button when hovered over or clicked rather than abruptly switching.

The following techniques can be used to animate SVG graphics:

- *SVG Animations*: The SVG elements `<animate>`, `<animateMotion>`, and `<animateTransform>` can be used to animate SVG objects. SVG animations are ideal for creating vector-based animations such as logos, icons, and illustrations directly within the HTML markup. An example with `< animateTransform>` is shown in Listing 5.1.
- *CSS Animations*: The CSS `@keyframes` rule can be used to define the trajectory of an animation. It provides a declarative way to animate CSS properties without JavaScript, making them efficient for creating simple to complex animations directly in the stylesheet. An example is shown in Listing 5.2.
- *JavaScript*: JavaScript is a versatile programming language used for creating interactive and dynamic web experiences. It can be employed to manipulate HTML, CSS, and SVG elements, enabling complex animations, interactivity, and dynamic content generation.
- *SMIL (Synchronized Multimedia Integration Language)*: SMIL is an XML-based markup language used for creating multimedia presentations, including animations and interactive content. Although less commonly used compared to CSS and JavaScript, SMIL offers precise control over timing and synchronization of multimedia elements.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>SVG Animation</title>
</head>

<body>
<svg width="300" height="200">
  <rect x="50" y="50" width="100" height="100" fill="blue">
    <animateTransform attributeName="transform"
      type="rotate" dur="6s" from="0 50 90" to="360 0 0"
      repeatCount="indefinite" />
  </rect>
</svg>
</body>
</html>
```

Listing 5.1: Animation using the SVG <`animatetransform`> element.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>SVG Animation</title>
</head>
<body>

<style>
#myRect {
  animation: scale 2s ease infinite alternate;
}
@keyframes scale {
  0% { transform: scale(1); }
  50% { transform: scale(1.5); }
  100% { transform: scale(1); }
}
</style>

<svg width="300" height="200">
  <rect id="myRect" x="50" y="50" width="100" height="100" fill="blue"/>
</svg>
</body>
</html>
```

Listing 5.2: Transition using a CSS `@keyframes` rule.

Chapter 6

Use Cases for SVG

There are many use cases for SVG graphics in web pages and applications, including for icons, charts, and diagrams. A sample web project was created to showcase some of these use cases [Heider 2024]. For this project, some code and icons were borrowed from the Rslidy project [Rslidy 2024].

6.1 Converting SVG Files to Inline TypeScript Strings

One of the ways to include SVG icons into a web project is to embed them inline into a data URI. In a Node.js setting, this procedure can be autpmated, for example by creating a TypeScript string with the data URI version of each SVG icon. This can be achieved conveniently with a Gulp task. The Gulp script also strips any inline style from the SVG files, so that styling can be set using CSS from the main the code.

6.2 Sample Web Page for Use Cases

The sample web page [Heider 2024] shows the following use cases:

- Changing the colour of icons.
- Scaling icons to create the illusion of flipping a switch.
- Changing the font of text inside an SVG.
- Hover effects for enabled and disabled icons.

Most these changes are achieved with CSS, but can be manipulated by JavaScript code as well.

For the switch, enabled and disabled attributes show if the switch is flipped on or off. The icon is coloured green when on and grey when off. In addition, it is scaled by -1 in the x direction when on to visually move the button to the right. The font change inside the SVG is done with a button which calls a JavaScript function, changing the font to the currently selected value. Hover effects are often used to show interactability. In the example web page, a disabled icon is shown in grey and changes the cursor to the “not allowed” variant when hovered.

6.3 Other Possible Use Cases

6.3.1 SVG Sprites

With SVG sprites, multiple different icons can be included in a single SGV file. They can be used individually without having to load additional content from a server. Further explanation can be found in Imoh [2023].

6.3.2 Masking content

An SVG can be used as a mask to cover or show (part of) an image. Since SVGs can be very dynamic, this allows for a great deal of freedom in shaping or styling content.

6.3.3 Data-Driven Charts

SVGs can also be used to display charts and visualisations based on (possibly changing) external data.

6.3.4 Background Images

As SVGs are very efficient in terms of file size, sometimes it can be worth using them as images in general on a web page. It is also possible, for example, to create textures in SVG and use them as a background image.

6.3.5 Situation-Based Icons

SVG icons can be manipulated internally in order to create dynamic iconography for showing things like a clock with the current time, the current weather, or current moon phase as icons.

Chapter 7

Styling Attributes and Styling Properties

SVG uses *styling properties* to control various aspects of rendering its objects. These are expressed as *presentation attributes* when used directly as attributes of an SVG element, for example `stroke-width ="3"`. In CSS or JavaScript, the styling properties sometimes have a slightly different syntax, for example `strokeWidth: 3` in CSS or `svg.setAttribute('strokeWidth','3');` in JavaScript.

Some more examples for SVG 1.1. are shown in Table 7.1. A detailed description of SVG styling properties and attributes can be found in W3C [2011, Chapter 6] for SVG 1.1 and W3C [2023, Chapter 6] for SVG 2.

Styling Attributes	**Styling Properties**
width / height	*
x / y	*
fill	fill
cursor	cursor
font-family	fontFamily
font-size	fontSize
opacity	opacity
stroke	stroke
stroke-width	strokeWidth
transform	transform

Table 7.1: Styling attributes and styling properties for SVG 1.1. In SVG 1.1, the `width`, `height`, `x`, and `y` attributes cannot be directly accessed via CSS or JavaScript. To manipulate position or rotation, transforms should be used.

Chapter 8

Validating SVG

Most SVG viewers attempt to recover from any syntax errors or malformed SVG, in the same way a browser handles errors in HTML. However, just like it is good practice to validate HTML before deployment, it is also possible to validate SVG. Validation of any XML-based format, including SVG and HTML, can be done by validating against a corresponding Document Type Definition (DTD). There are also vaidators which do not use a DTD.

8.1 SVG 1.1

The W3C Markup Validation Service [W3C 2024a] uses a Document Type Definition (DTD). Validation works well for SVG 1.1 files. For example, using Validate by File Upload, under More Options, manually set Character Encoding: utf-8 and Document Type: SVG 1.1.

The alternative W3C Nu Html Checker [W3C 2024b] is not DTD-based. Validation also works well for SVG 1.1 files. For example, using Check by: file upload, and uploading a file with a .svg extension, the validator with validate according to the root namespace specified in the file.

Both validators find small errors such as a missing closing tag, or `viewbox` being spellt with a lower case `b` rather than upper case.

8.2 SVG 1.2

There is no DTD for SVG 1.2 yet. There is also no need to specify the DOCTYPE. Identification is instead done by SVG namespace and `version` and `baseProfile` attributes. In SVG Tiny 1.2, validation can be performed using the RelaxNG schema for SVG Tiny 1.2.

8.3 SVG 2

As stated in W3C [2023, Section K.2.27], "SVG 2 is not defined in terms of a DTD.", which makes validation by DTD impossible. Efforts to create a schema for SVG 2, as was done for SVG Tiny 1.2, are still ongoing, according to the most recent update about the issue:

> "The IEC 61850-6-2 Task Force are making good progress on the SVG2 xsd schema, and are looking for any interested SVG experts to review/contribute in any capacity." [Tessier 2021]

Chapter 9

Conclusion

Scalable Vector Graphics (SCGs) are very powerful and in some ways rather straightforward to include and use in modern web browsers. They can be dynamically styled and animated, giving them many potential use cases. Although more advanced techniques of using SVGs on the web require a higher degree of maintenance and expertise, there are numerous resources online that provide the help and information necessary for successfully working with SVGs.

Bibliography

Bellamy-Royds, Amelia, Kurt Cagle, and Dudley Storey [2017]. *Using SVG with CSS3 and HTML5: Vector Graphics for Web Design*. O'Reilly, 05 Nov 2017. 818 pages. ISBN 1491921978. https://oreilly.com/library/view/using-svg-with/9781491921968/ (cited on page 14).

Buckler, Craig [2021]. *How to Add Scalable Vector Graphics to Your Web Page*. 01 Feb 2021. https://sitepoint.com/add-svg-to-web-page/ (cited on pages 3, 7).

Heider, Martin [2024]. *SVG on the Web - Use Cases*. 12 May 2024. https://gitlab.tugraz.at/95FD77DBCF078A32/svg-on-the-web-use-cases/ (cited on page 21).

Imoh, Ifeoma [2023]. *How to Use SVG in React*. 17 Oct 2023. https://telerik.com/blogs/how-to-use-svg-react (cited on page 21).

Rslidy [2024]. *Rslidy*. 30 Jan 2024. https://github.com/tugraz-isds/rslidy (cited on page 21).

Tessier, Dustin [2021]. *Building a schema for SVG, Issue #699*. 11 Mar 2021. https://github.com/w3c/svgwg/issues/699#issuecomment-796860551 (cited on page 25).

W3C [2011]. *Scalable Vector Graphics (SVG) 1.1*. W3C Recommendation. World Wide Web Consortium, 16 Aug 2011. https://w3.org/TR/SVG11/ (cited on pages 3, 23).

W3C [2023]. *Scalable Vector Graphics (SVG) 2*. Editor's Draft. World Wide Web Consortium, 08 Mar 2023. https://svgwg.org/svg2-draft/ (cited on pages 3, 23, 25).

W3C [2024a]. *Markup Validation Service*. World Wide Web Consortium, 12 May 2024. https://validator.w3.org/ (cited on page 25).

W3C [2024b]. *W3C Nu Html Checker*. World Wide Web Consortium, 12 May 2024. https://validator.w3.org/nu/ (cited on page 25).

W3Schools [2024]. *HTML Element Reference*. 10 May 2024. https://w3schools.com/tags (cited on page 7).