SVG Icons 3 ways

In this tutorial we will create SVGs to be displayed on a webpage in different ways and look at how control over placement and colour varies. We will look at three different methods:

1. SVG as image
2. SVG as background
3. SVG in line with interaction

Context

We'll create two SVG icons in a consistent style: a personal monogram (our initials) and a functional UI icon (external link arrow). Both could be used throughout our design portfolio.

A portfolio website does not necessarily need a logo at all and could merely typeset your name and job description or tagline nicely. A monogram however could be an alternative. Using your initials will make it personal and it gives you something to add to your work or possibly even use instead of a photograph for online profiles.

An arrow for links which go to external sites is an excellent addition to any portfolio site. It can be seen as an accessibility enhancement to show clearly which links will lead away from the website. And for a design or development portfolio, it is a fitting addition to your case studies, which will feature links to the real work online.

Part 1: creating the icons

1.1 Setting up your artboard

1. Create a new file: 512px x 512px for the monogram
   For the external link icon, use 24×24px (standard icon size)
2. Ensure that colour mode is set as RGB
3. Enable the grid (if you prefer): View → Show Grid — use 8px or 24px increments

1.2 Designing your monogram

For the sake of this exercise, work with a basic version first to learn the techniques. Take your time to refine your monogram design for its final use later on. It is then easily replaced ツ

Approach A: From letterforms

1. Type your initials using a bold sans-serif (e.g., Impact, Helvetica Bold)
2. Scale to fit the artboard neatly, leaving a few pixels margin, and centre-align

3. Convert to outlines: Cmd/Ctrl + Shift + O or
   Top menu → Type → Create Outlines
4. Experiment with overlapping, interlocking, or sharing strokes between letters

Approach B: Geometric construction

1. Build letters from basic shapes: rectangles, circles, arcs
2. Use the Pathfinder panel to unite, subtract, or intersect shapes
3. Align to grid points for clean geometry
4. Scale to fit the artboard neatly, leaving a few pixels margin, and centre-align

Design consideration: Your monogram should remain legible at a very small size like 16px. Test frequently at different sizes using View → Actual Size for full view and zoomin out to preview smaller version. You could also test by exporting a test PNG ~ the bitmap rendering is a good test preview.

1.3 Designing the external link icon

Use the same visual weight as your monogram (similar stroke thickness or shape style) for consistency.

Let's create this common icon ~ a square with a vertical line to hint at an arrow pointing to top right through the gap in the bottom left corner. Again, you can edit and update this later on to fit the final design.

1. Draw a square
2. Draw a diagonal line from bottom-left to top-right (the arrow shaft), leaving a small gap bottom left
3. Set stroke weight to match your monogram's visual weight
4. Apply rounded end caps for a friendlier feel: Stroke panel → Round Cap

The external link icon can take different shapes and forms ~ the main concept to include when designing this is an implied direction (top right) as a signal that the link is leading to an external page.
(Take a look at other icon design variations on icons8.com)

1.4 Finalising vector paths

SVGs should be a clean, simple and light as possible. A complex visual might weigh quite heavily on the overall file size if we do not optimise. The simpler the graphic, the lighter - the fewer points, the better.

Before export, let's clean up our paths for optimal SVG output:

1. Expand strokes: Object → Path → Expand (converts strokes to shapes)
2. Merge shapes: Select overlapping shapes → Pathfinder → Unite
3. Simplify paths: Object → Path → Simplify (reduce anchor points)
4. Check in outline view: Cmd/Ctrl + Y to see actual paths

Part 2: Export & Optimise

2.1 Export settings

From Illustrator: File → Export → Export As → SVG

- Styling: Presentation Attributes (not internal CSS)
- Font: Convert to Outlines
- Images: Embed (if any)
- Decimal places: 1 (keeps file small)
- Minify: No (we'll optimise properly next)
- Responsive: Yes (removes fixed width/height)

From Figma: Select the frame → Export panel → SVG

- Include "id" attribute: Off
- Outline text: Yes

A quick note on text/font considerations

While a SVG can embed fonts, this is still a very unpredictable feature. Some browsers will render everything perfectly, while others will substitute the font with a fallback or system font. This then can ruin the entire design.

This is the reason why we outlined text in our preparation for export, or during export. Outlined text will technically become a graphic and not change when added to our webpages.

For accessibility (+SEO) reasons we should never create full passages of text, be that headings or blockquotes, as SVGs (which might feel tempting). This will make the text illegible to screensreaders as well as search engines.

2.2 Optimise with SVGOMG

Regardless of how carefully we configure settings, we should always optimise our SVG code. Even well-exported files contain redundant data.

1. Open your exported SVG in a code editor

2. Save a copy as v2, eg monogram-v2.svg (keep original as backup)
3. Copy all code
4. Go to SVGOMG and paste
5. Review settings (defaults are usually fine)
6. Copy optimised code → paste into your v2 file → save
7. Compare file sizes — you'll typically see 30–60% reduction

2.3 Manual cleanup

SVGOMG does a great job and we should not have to do anything extra - but better safe than sorry ツ Open your optimised SVG and check for:

- Unnecessary attributes: Remove xmlns:xlink if not using <use>
- Editor metadata: Delete any comments like <!-- Generator: Adobe Illustrator -->
- Empty groups: Remove empty <g></g> elements
- viewBox present: Ensure viewBox="0 0 512 512" exists
- No fixed dimensions: Remove width and height from the root <svg> (CSS will control size)

Part 3: Method 1 — SVG as <img>

Use this method for: simple, scalable images without CSS interaction.

Demo: Monogram at different/smaller sizes

32px 64px 128px

Demo: External link icon at different sizes

32px 64px 128px

The code

HTML

```
<img src="icons/monogram.svg" alt="PS monogram"
  class="icon-img icon-img--md"
  width="64" height="64">
```

CSS

```
/* Base styles for all icon images */
.icon-img {
   display: block;
   /* prevent overflow */
   max-width: 100%;
    /* maintain aspect ratio */
   height: auto;
}

/* Size variations */
.icon-img--sm { width: 32px; }
.icon-img--md { width: 64px; }
.icon-img--lg { width: 128px; }
```

Always include width and height attributes in HTML. This prevents layout shift while the image loads. The CSS will override the rendered size ~ importantly, the browser knows the aspect ratio from the start of page load.

Key limitation

When SVG is loaded as an external file via <img>, you cannot style its internal elements with CSS. The SVG is treated as a flat image with fills, strokes, and colours are set in the SVG code itself.

This is fine for static images where the colour never changes. Most logos, for example, will not need any edits to colour and can easily be included as images. For interactive icons (hover states, theme switching), use Method 3: Inline SVG.

Part 4: Method 2 — SVG as CSS Background

Use this method for: decorative patterns, watermarks, or textures, i.e. purely as visual addition and without CSS interaction.

Demo: Watermark pattern

'Wallpaper'

This section is the demo of using the monogram as tiled watermark. The pattern sits behind the content using a pseudo-element with low opacity.

This look a little with a simple design such as this but we can always spend more time to create a more delicate and refined. Importantly, wallpaper/tiled background textures only work if subtle (as this demo), or if only used in the margins, for example.

The code

Option A: External file reference

```
.work-feature {
  position: relative;
}

.work-feature::before {
  content: "";
  position: absolute;
  inset: 0;
  opacity: 0.03;
  background-image: url("icons/monogram.svg");
  background-size: 80px 80px;
  background-repeat: repeat;
  /* allow click through */
  pointer-events: none;
}
```

Option B: Data URI (more portable)

Note on portability: SVG as Data URI is more portable because it's self-contained: no separate file, no path management, easier reuse across projects as well as fewer calls to the server.

```
.work-feature::before {
 /* ... repeat positioning as Option A ... */
 background-image: url("data:image/svg+xml,%3Csvg xmlns='...'%3E...%3C/svg%3E");
}
```

Converting SVG to data URI

To embed SVG directly in CSS, URL-encode special characters:

Character	Encoded
<	%3C
>	%3E
#	%23
"	Use ' instead

Or use a tool like URL-encoder for SVG.

Critical limitation: currentColor for fill (or stroke if used) does not work in background images. The SVG has no access to CSS inheritance when loaded as a background. We must set an explicit fill colour in the SVG code itself.

PNG fallback for older browsers

For browsers that don't support SVG backgrounds (increasingly rare), use this clever technique:

```
/* fallback */
background: url("icons/monogram.png");

/* SVG for compliant browsers */
background: url("icons/monogram.svg"),
        linear-gradient(transparent, transparent);
```

If a browser supports both multiple backgrounds *and* linear gradients, it supports SVG. The transparent gradient is added to trigger feature detection.

Part 5: Method 3 — Inline SVG with Interaction

Use this method for: icons that change colour on hover, focus, or with themes which might require different colour treatments for light/dark, for example.

The magic of currentColor

When SVG code is placed directly in our HTML, it can use currentColor as a fill or stroke value. This special keyword inherits the CSS color property from the parent element.

✗ External SVG file

fill="currentColor"

/ no effect — no CSS context */*

✓ Inline SVG

fill="currentColor"

/ CSS color property inherited from parent */*

Demo: Interactive icons

Design update for exhibition website
View project

Demo: Project card

The icon in included in the link and will change colour on interaction. It is not a clickable element per se but part of an accessible and easy-to-use link instead. This is a key pattern for clickable cards.

Exhibition Redesign

Complete UI overhaul and refresh of colour scheme.

The code

HTML structure

<a href="#" class="project-card">

<h4 class="project-card__title">Project Name</h4>

<!-- Inline SVG with currentColor -->

```
  <svg class="project-card__icon" viewBox="0 0 24 24" aria-hidden="true">
   <path stroke="currentColor" ... />
  </svg>
 </div>
 <p class="project-card__desc">Description...</p>
</a>
```

link styling

```
/* Default state: muted colour */
a.project-card__icon:link, a.project-card__icon:visited {
  width: 24px;
  height: 24px;
  color: #64748b;
  transition: color 0.2s ease;
}

/* colour change for icon */
a.project-card:hover .project-card__icon,
a.project-card:active .project-card__icon,
a.project-card:focus-visible .project-card__icon {
  color: #a2c90f;
}
```

Accessibility requirements

Scenario	Approach
Icon with visible text label	aria-hidden="true" on the SVG (text provides me
Icon-only button/link	aria-label="Description" on the parent element

Meaningful standalone image	role="img" + <title> inside the SVG

Quick Reference

Method	Best for	Limitations
<img>	Static images at various sizes	No CSS control over internal elements
CSS background	Patterns, watermarks, decoration	No currentColor; no interaction; requi
Inline SVG	Interactive icons, hover states, theming	More HTML; not cached separately; ca copy/paste

To do list ツ

As practice, apply what you learned to your current project, your portfolio ~ or a demo page ツ Learning by doing is often best.

download empty files:
HTML/CSS files - .zip, 4kb

1. Create your monogram using your own initials
2. Create an external link icon in the same visual style
3. Export and optimise both via SVGOMG
4. Implement Method 1: Monogram in your header as <img> at multiple sizes
5. Implement Method 2: Monogram as a subtle background watermark in a section
6. Implement Method 3: External link icon inline in your project cards with hover interaction

code tip

For an easy way to add icons to external link generally - use CSS to target links that link elsewhere, away from your site/domain.

```
a[href^="http"] {
  /* external link styles ` */
```

}

Testing checklist

- Does your monogram remain legible at 16px? 32px?
- Does the watermark add brand cohesion without distracting from content?
- Do the icons respond to hover and focus states?
- Are focus states visible for keyboard navigation?

Extra: SVG Clipping Path with Image Fill

The feature image at the top of this page uses a different technique altogether ~ the monogram has been used as a 'cookie cutter' to punch a hole into th e circle which is used as a clipping mask for a watercolour texture. This allows for the painterly, watercolour effect to shine while keeping the sharp vector edges of the outline and the monogram letterforms.

How it works

Instead of filling our paths with a solid colour, we define the shape as a clipPath and apply it to an image. The image is then "cut out" in the shape of our monogram.

The code

```
<svg viewBox="0 0 512 512" width="300" height="300"
xmlns="http://www.w3.org/2000/svg">
  <defs>
    <!-- define the monogram as a clipping mask -->
    <clipPath id="monogram-clip">
      <!-- monogram path(s) here -->
      <path d="M..." />
      <path d="M..." />
    </clipPath>
  </defs>

  <!-- image clipped to the monogram shape -->
  <image href="texture.png"
```

```
    width="512" height="512"
    clip-path="url(#monogram-clip)"
  />
</svg>
```

Step by step

1. Prepare your texture image
   any image works: watercolour, gradient, photograph, pattern. Match the dimensions to your SVG viewBox (512×512 in our case).
2. Export your monogram paths
   export as usual from Affinity, Illustrator or Figma, optimise + then open the optimised SVG and copy just the <path> elements.
3. Create the clipPath structure
   wrap your paths in <defs> and <clipPath> with a unique ID.
4. Add the image element
   reference your texture and apply clip-path="url(#your-id)".

Design considerations

Aspect	Recommendation
Image size	Match or exceed the viewBox dimensions to avoid gaps
Image format	PNG for transparency; JPG for photos
File weight	The image adds to total file size — optimise both SVG and image
Positioning	The image starts at 0,0 by default; use x and y attributes to offset if needed

Why use this technique?

This approach gives you effects that would be difficult or impossible with solid fills: photographic textures, complex gradients, or artistic media like watercolour and ink. It is particularly effective for header sections, portfolio headers, or social media graphics where your monogram needs extra visual impact.

The SVG remains scalable ~ the clipping path is vector ツ This means the edges stay crisp at any size. Only the image texture will show its pixels if scaled too large.

Further Reading

- SVG on MDN
- Using SVG (CSS-Tricks)
- Accessible SVGs
- A Complete Guide to SVG Fallbacks
- SVG articles by Sara Soueidan
- SVGOMG Optimiser