Modern Web Vector Graphics

fserb@chromium.org - July 2023

This document presents an exploration of the modern use cases for web vector graphics, the current solutions and their limitations, and potential future improvements. It's clear that as web design evolves, so too must the technologies that facilitate animations, interactivity, and responsiveness. The landscape of vector graphics is complex and demands solutions that can bridge the gap between developer desires, performance expectations, and user experiences.

Open Questions

Should we continue to enhance SVG or move towards a new vector format? With SVG's limitations to the new use cases and the potential for a more modern vector format, should we focus on enhancing SVG or devise a novel vector format altogether?

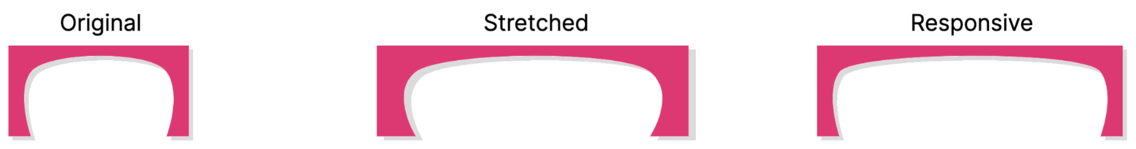
What are the constraints that we should focus on? Supporting better integration with CSS and allowing animations? Should we try to support non-web rendering? Especially if we consider text, is there a possibility of designing a format that doesn't depend on a web browser to render? Should this be a goal?

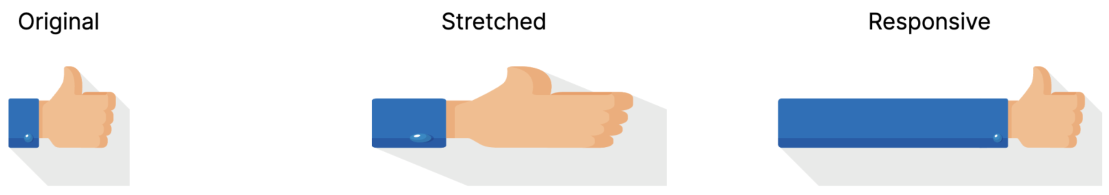
What is the best approach for text rendering in a new vector format? Given the complexity that comes with supporting text rendering, especially when considering internationalization and accessibility, it's vital to decide whether a new vector format should incorporate this feature.

Problem Space and Use Cases

Responsive Vector Graphics

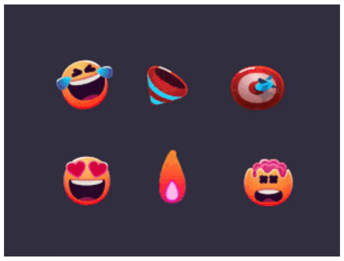
Responsive design became the expected baseline of modern UI accessible design. Vector graphics always played a role in this as "size-independent" graphics (for example, on icons). In spite of some solutions for level-of-detail responsiveness, there has never been support for truly responsive vector graphics that adapt in dynamic ways.





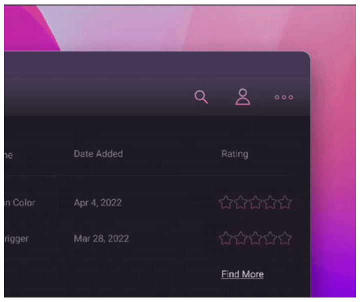
Animated Vector Graphics

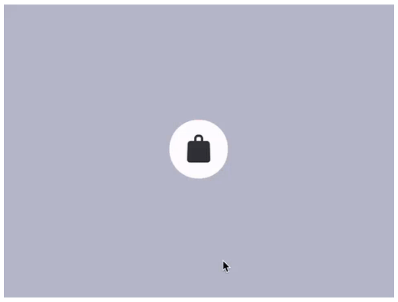
Vector graphics have been heavily utilized in creating smooth, complex graphics for the web. Currently, their usage in animations is mostly intermediated by runtimes, but they could potentially offer more flexibility and quality than traditional raster-based animations.



Interactive UI Elements

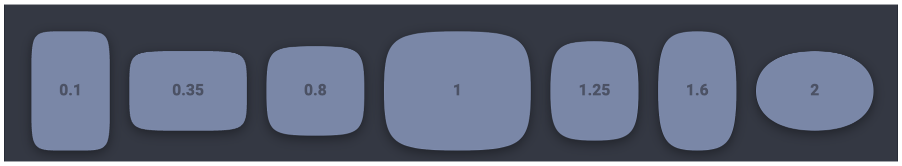
Interactive UI elements, such as buttons, icons, infographics, and data visualization components, are increasingly made using animations. Motion is considered an important part of UX. Being able to do them through vector animations provide scalability, resolution-independence, and the potential for interactivity and animation.

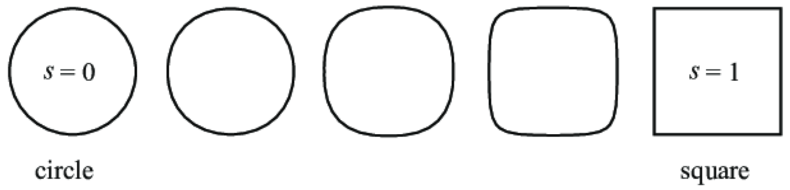




Parametric Shapes

Parametric shapes are vector graphics defined based on a set of parameters or equations. This makes them highly flexible and customizable, allowing for dynamic and responsive designs. They allow people to have access to easy-to-describe (like the curveness in squircles, described below) but hard to build shapes. It's an emerging design trend and we should make it easier for people to compose.





Current solutions & limitations

There's currently a set of tools and practices to address some of those use cases.

There are animation runtimes, like Greensock (GSAP), Lottie, and Rive. In the case of GSAP, it provides a JS library to build sequence or scroll based animations. Lottie and Rive provide more integrated solutions from animation tools that are exported as animations controlled by a JS runtime. Those can be very powerful and their end-to-end aspect can be very interesting, but they do suffer from potential performance issues and poor integration with the rest of the web platform.

The runtime solutions are close to the experience designers want, but they suffer from some run-time limitations that end up affecting end users. Animations are usually very performance demanding, requiring Canvas JS rendering or complex DOM manipulations. There's a fundamental disconnect here: developers want those experiences that are mostly declarative in nature ("animate this object" or "place this shape here"), and have to rely on imperative runtimes to support them, because the web doesn't provide good declarative primitives for them.

CSS and SVG have some declarative animation support that is able to address some of the same use cases, and because they are declarative they are often more performant. The complex cases, however, can be brittle and difficult to achieve.

There's a consistent movement towards more responsive and animated UI elements. Even simple things like checkboxes are expected to behave in more interesting ways than just state switching, with animations and motions being part of the overall UX experience. The current alternatives for it are either very complex and hack-y (using pseudo-elements to design a tick mark, or dot spacing for line animation, for example), or demand completely custom elements. Both of those solutions have downsides in terms of accessibility, performance and difficulty.

For shapes, there are external builders that allow people to generate one-off SVGs of the shapes they want, like Blobmaker or Squircle generator. Those are limited and not very flexible.

What the future could look like?

The current set of solutions are an indication that developers care about this space and, in spite of current limitations, are really trying to push Web UX design in those directions.

Usage of vector animations both as UI elements and overall graphics components should become 1st class citizens on the web, as easy as adding raster images to a page. Vector images should be fully integrated with CSS. Animations should be controllable via both declarative and imperative APIs.

Vector images should be parameterizable both in time/scroll (for animations), box dimensions (for responsiveness), and other parameters (for parametric shapes). Those parameters should be able to not only change image properties (like we currently have on CSS and SVG), but also change path control points and other animation parameters (such as squircle roundness).

Allowing vector formats to be declarative opens up a world of potential optimizations and performance improvements for UAs that would be able to choose how to handle performance tradeoffs and improve smoothness and remove jank. It also drastically reduces the barrier to use those features.

Challenges for vector image formats

There have been several attempts to design new vector image formats (like Flutter vector, IconVG) or animation specific ones (like Lottie, or Rive). They want to try to provide a cross-platform (including non-web uses) experience; they want a simple format that can be bridged directly to modern GPUs in a performant way; and sometimes they want to support features like full vector animation.

They usually try to stay away from SVG, mostly due to the implementation complexity (not only Text and ForeignObjects, but even style, and filters). And most of them tend to focus on implementation details like binary formats.

The one common trend among them is that they end up having to avoid text rendering, as the added complexity seems not solvable in a true cross-platform way. It seems that if you want to have text rendering (i18n, accessibility, native support) at the level people have grown to expect from the web, you need a web browser.

In the end, it would be possible to have a simpler cross-platform solution for vector graphics, but it would imply dropping text. One could argue that in some situations that would be useful (an icon vector format, for example), while in others less so (a diagram vector format).

If you must have text rendering, it is not clear that it's possible to design a "simple" format that doesn't depend on a chunk of the browser technology and would end up in the same design space as SVG.

That said, staying in the SVG space, while beneficial from lots of aspects, still has some potential shortcomings. Namely, to support the use cases listed above, we would need to consider some additions that would allow 1st class path support (allowing parametrization of control points) on SVG.