CSS-TRICKS

A Comparison of Animation Technologies

BY **SARAH DRASNER** ON MAY 2, 2016

ANIMATION, CANVAS, GREENSOCK, SHAPE MORPHING, WEBGL

The question I am asked most frequently: what animation tool do you recommend?

Having worked with a slew of them, I can tell you **there is no right answer**. It's a complicated question and complicated answer. This post serves to clarify what to use, and when, to get you working with the right tool for the job.

If you're here for React, we've got you covered! Jump down to the React section below and we'll break down what to use and how to use it.

There's no possible way to go over every single animation library around, so I'm going to stick with the ones that I've either used or that interest me. Please keep in mind that these are my recommendations based on my own experiences, and the web has a lot of grey area, so you may have a different experience or opinion. That's OK.

Native Animation

Before we talk about any libraries, let's go over some native animation implementations. Most libraries are using native animation technologies under the hood anyway, so the more that you know about these, the more you'll be able to negotiate what is happening when it becomes abstracted.

CSS

CSS remains one of my favorite ways to animate. I've been using it for years and continue to be a fan due to its **legibility and excellent performance**. CSS animations make it possible to transition between different states using a set of keyframes.

Pros:

- You don't need an external library.
- The performance is great, especially if you do things that are inherently hardware accelerated (offloaded to the GPU). Here's an article about the properties that will help you do so.
- Preprocessors (like Sass or Less) allow you to create variables (for things like easing functions or timings) that you'd like to remain consistent, along with `:nth-child` pseudo classes in functions to produce staggering effects.
- You can listen for `onAnimationEnd` and some other animation hooks with native JavaScript.
- Motion along a path is coming down the pipeline which will be rad.

- It is easy to use for responsive development because you can modify your animation with media queries.

Cons:

- The bezier easings can be a bit restrictive. With only two handles to shape the bezier, you can't produce some complex physics effects that are nice for realistic motion (but not necessary very often).
- **If you go beyond chaining three animations in a row, I suggest moving to JavaScript.** Sequencing in CSS becomes complex with delays and you end up having to do a lot of recalculation if you adjust timing. Check out this Pen from Val Head that illustrates what I mean. Staggers are also easier to write in JavaScript. You can hook into the native JavaScript events I mentioned earlier to work around this, but then you're switching contexts between languages which isn't ideal, either. Long, complex, sequential animations are easier to write and read in JavaScript.
- The support for motion along a path isn't quite there yet. You can vote on support for Firefox here. Voting for support in Safari, as far as I can gather, is a little more individual. I registered to fill out a bug report and requested motion path module in CSS as a feature.
- CSS + SVG animation has some strange quirkiness in behavior. One example is that in Safari browsers, opacity and transforms combined can fail or have strange effects. Another is that the spec's definition of transformation origin, when applied sequentially, can appear in a non-intuitive fashion. It's the way that the spec is written. Hopefully SVG 2 will help out with this but for now, CSS and SVG on mobile sometimes requires strange hacks to get right.

requestAnimationFrame

`requestAnimationFrame` (rAF) is a native method available on the window object in JavaScript. It's really wonderful because under the hood, it figures out what the appropriate frame rate is for your animation in whatever environment you're in, and only pushes it to that level. For instance, when you're on mobile, it won't use as high a framerate as on desktop. It also stops running when a tab is inactive, to keep from using resources unnecessarily. For this reason, `requestAnimationFrame` is a really performant way of animating, and most of the libraries we'll cover use it internally. `requestAnimationFrame` works in a similar fashion to recursion, before it draws the next frame in a sequence, it executes the logic, and then calls itself again to keep going. That might sound a little complex, but what it really means is that you have a series of commands that are constantly running, so it will interpolate how the intermediary steps are rendered for you very nicely.

Canvas

Canvas is great to get those pixels moving! So many of the cool things you see on CodePen are made by canvas magicians. Like its name suggests, canvas offers a visual space scripting, in which you can create complex drawings and interaction, all with high performance rendering. We're working with pixels here, which means they are raster (as opposed to vector).

Pros:

- Since we aren't moving things around in the DOM, but rather a blob of pixels, you can make a ton of complex things happen without weighing down performance.
- **Canvas is really nice with interaction**. You're already in JavaScript which offers a lot of control without switching contexts.
- The performance is very good. Especially considering what you can make.
- The sky's the limit on what you want to create.

Cons:

- It's difficult to make accessible. You'd have to use something like React to create a DOM for you, which Flipboard managed to do a little while ago, though in their case study it seemed they had more work to do. There's probably an opportunity there, though, if you're interested in taking up the charge.
- I find it pretty unintuitive to make responsive. By this, I don't mean making a small version work on the phone, because that it does well. I mean moving and shifting and reorganizing content based on the viewport. I'm sure there are some really great devs out there who do it easily, but I would say it's not terribly straightforward.
- It's not retina-ready. Unlike SVG animation which is resolution-independent, most of the canvas animations I see are not crisp on retina displays.
- It can be difficult to debug. When what you are building breaks, it breaks by showing nothing, unlike animating in the DOM, where things are still there but just behaving weirdly.

SMIL

SMIL is the native SVG animation specification. It allows you to move, morph and even interact with SVGs directly within the SVG DOM. There are a ton of pros and cons to working with SMIL, but the biggest con will lead me to omit it entirely: **it's losing support**. Here's a post I wrote on how to transfer over to better-supported techniques.

Web Animations API

The Web Animations API is native JavaScript that allows you to create more complex sequential animations **without loading any external scripts**. Or will, anyway, when support grows. For now you'll probably need a polyfill. This native API was created as a backfill from all of the great libraries and work that people were already making with JavaScript on the web. This is part of a movement to align the performance of CSS animations and the flexibility of sequencing in JavaScript under one roof. If you're interested in hearing more, there is a ShopTalk episode all about it.

Pros:

- Sequencing is easy and super legible. Just check out this example from Dan Wilson which compares CSS keyframes and the web animation API.
- Performance seems to be on track to being very good. I haven't tested this myself well yet but plan to. (You can and should be running your own performance tests as well).

Cons:

- Support is not great right now. It's also still changing a bit so while the spec is moving I would be cautious about running it in a production environment.
- There are still a lot of things about the timeline in GSAP (which we'll be covering in a moment) that are more powerful. The important ones for me are the cross-browser stability for SVG and the ability to animate large swaths of sequences in a line of code.

WebGL

There are some extraordinary things you can do with WebGL. If anything is blowing your socks off, there's a good chance it was made with WebGL. What it's really good at is interactive and 3D effects. Check out some of these demos. Technically it uses canvas, but I like making the distinction because, I don't know, names are cool when they work to identify slight variance.

Pros:

- **Amazing visual effects**
- Three dimensions means a whole new world for interaction
- Hardware-accelerated
- Possibilities for VR

Cons:

- Tougher to learn than the other native animations we've covered thus far. The first two animations I made I thought weren't working because I had the camera pointed the wrong way (womp womp). There are also less really good articles and books explaining how to work with it than the other technologies I've mentioned thus far, but they're growing.
- Hard to make responsive. Most websites I've seen that use WebGL fall back to a "please view this site on desktop" screen.

External Libraries

GreenSock (GSAP)

GreenSock is far and away my favorite library to work with. Please understand that this bias comes from working, playing around with, and bumping my head badly on a lot of different animation tooling so when I give my strong endorsement, it's through blood, sweat and tears. I especially like it for SVG. The GreenSock Animation API has almost too many cool benefits to list here without missing something, but they have both docs and forums you can explore.

Pros:

- It's **extraordinarily performant** for something that's not native. As in, performs just as well, which is major big deal.
- The sequencing tools (like timeline) are as easy to read as they are to write. It's very simple to adjust timing, have multiple things fire at once with relative labels, and even speed your animations up or down with one line of code.

- They have a ton of other plugins if you want to do fancy things like animate text, morph SVGs, or draw paths.
- Their motion along a path in the bezier-plugin has the longest tail of support available.
- **It solves SVG cross-browser woes**. Thank goodness for this one. Especially for mobile. There's some good info about that in this post they wrote.
- They offer really nice, very realistic eases.
- Since you can tween any two integers, you can do cool stuff like animate SVG filters for some awesome effects, and even works in canvas. Sky's the limit on what you can animate.
- GSAP is the only animation library, including the native SMIL, that lets you morph path data with an uneven amount of points. This has opened new opportunities for motion and movement. (If you'd like to it in action, go the logo at the top of this page on desktop and click the orange bit in the dash.)

Cons:

- You have to pay for licensing if you're re-selling your product to multiple users (apps and paid-access sites). That's probably not most cases. I'll also put in that supporting work on this isn't the worst thing in the world.
- As with any external library, you have to watch which versions you are using in production. When new versions come out, upgrading will involve some testing (like any library). You'll also need to consider the size of the library.

VelocityJS

VelocityJS offers a lot of the sequencing that GreenSock does, but without a lot of the bells and whistles. I no longer really use Velocity due to the cons below. Velocity's syntax looks a bit like jQuery, so if you've already been using jQuery, it will be familiar.

Pros:

- Chaining a lot of animations is a lot easier than CSS.
- For refinements in eases there are step-eases where you can pass an array.
- The documentation is pretty nice so it's easy to learn and get up and going.

Cons:

- **The performance isn't great**. Despite some claims to the contrary, when I ran my own tests I found that it didn't really hold up. I suggest you run your own, though, as the web is always moving and this post is frozen in time.
- GSAP has more to offer, for performance and cross-browser stability without more weight. I would also say that jQuery used to lose performance tests but that might have changed since their RAF adoptions, so Velocity isn't bad by any means, but it loses out in comparison.
- From what I can tell, it's not actively being maintained anymore. I've heard that Julian Shapiro has passed the project off to someone else, so it may make a resurgence sometime in the future.

jQuery

jQuery isn't solely an animation library, but it has animation methods and a ton of people use it, so I thought I would cover it.

Pros:

- A lot of sites already have it loaded, so it's usually readily available.
- The syntax is really easy to read and write.
- They recently moved over to `requestAnimationFrame`, so in versions 3.0.0 and greater, the performance will be better than before.

Cons:

- Versions earlier than 3.0.0 performance is not great and not recommended. I haven't tested this in a very formal way, but aside from looking at repaints and painting in Chrome DevTools, you can also see it with your eyes.
- **Animations aren't supported on SVG in any version**.
- jQuery is limited to the DOM, unlike tools like GSAP which work on anything.

Snap.svg

A lot of people think of Snap as being an animation library, but **it was never intended to be one**. I was going to run performance benchmarks on Snap, but even Dmitri Baranovskiy (the incredibly smart and talented author of Snap.svg, and it's predecessor, Rafael), says here on the SVG Immersion Podcast that it's not the correct tool for this, and in a message to me said:

> Just a note: Snap is not an animation library and I always suggest to use it with GSAP when you need a serious animation.

That said, jQuery is not great with SVG, though they do now support class operations. If I need to do any DOM manipulation with SVG, Snap is really awesome for this.

A library called SnapFoo was released recently, and it extends Snap's realm to animation. I haven't played with it myself yet, but it looks pretty cool and is worth a go. If you try it, report back or comment below!

Mo.js

This is a library by an awesome fellow that goes by LegoMushroom. He's an extremely talented animator, and has already made some great demos for this library that have me really excited. This library is still in beta it's getting very close to being released now. I wrote an introductory guide to working with it here if you're interested in learning more.

Pros

- There are things like shapes, bursts, and swirls, which are really easy to work with and spin things up for you- so you don't need to be the world's best or most creative illustrator to get something nice going.

- Mo.js offers some of the best and most beautiful tooling on the web, including players, timelines, and custom path creators. This in and of itself is one of the most compelling reasons to use it
- There are a few different ways to animate- one is an object, one is plotting a change over the course of an ease- so you can decide which way you feel more comfortable.

Cons

- It doesn't yet offer the ability to use an SVG as a parent for the custom shapes, (I believe LegoMushroom is working on this), so working with coordinate systems and scaling for responsive is less intuitive and harder to make work on mobile. This is a fairly advanced technique, though, so you might not need it.
- It doesn't correct cross-browser behavior like GreenSock does yet, which means you might need to write hacks like you do with CSS. He mentioned he's also working on this as well.

Three.js

Three.js is a beautiful three dimensional animation tool! Check out some of these examples. I haven't worked with this a lot so I don't feel comfortable reporting on it, but I plan to and will update the post when I know more. People usually talk about this and the native WebGL in tandem.

Bodymovin'

Bodymovin' meant for building animations in Adobe After Effects that you can **export easily** to SVG and JavaScript. Some of the demos are really impressive. I don't work with it because I really like building things from scratch with code (so this defeats the purpose for me), but **if you're more of a designer than a developer**, this tool would probably be really great for you. The only con I see to that part is that if you need to change it later, you'd have to re-export it. That might be a weird workflow. Also, the outputted code is usually kind of gross, but I haven't seen that affect the performance too much in the case of Bodymovin', so it's probably fine.

React-Specific Workflows

Before we talk about React, let's cover why we have to treat animations in React differently at all. DOM stands for Document Object Model. It's how we create a structured document with objects, the nodes of which we talk about as a tree. React interacts with a virtual DOM, which is unlike the native browser DOM implementation, it's an abstraction.

React uses a virtual DOM number of reasons, the most compelling of which is the ability to figure out what's changed and update only the pieces it needs to. **This abstraction comes at a price**, and some of the old tricks that you're used working with will give you trouble in a React environment. jQuery, for instance, will not play nice with React, being that it's primary function is to interact and manipulate with the browser's native DOM. I've even seen some strange CSS race conditions. Here are some of my recommendations for nice animations in a React workflow.

React-Motion

React-Motion is a very popular way to animate in React, and the community has adopted it over the old `ReactCSSTransitionGroup` that used to be most frequently used. I like it a lot, but there are some keys points to working with it that if you don't get, will have you banging your head.

React-Motion is pretty similar to game-based animation, where you give an element mass and physics and send it on it's way, and it gets there when it gets there. What this means is that you aren't specifying an amount of time like you do with CSS or any other traditional web-based sequential motion. This is a pretty different way of thinking about animation. This can mean that the motion has the **ability to look really realistic** which can be very beautiful. But UIs have different requirements than games, so working with this can get tricky. Let's say you have two things that have to fire independently and get there at the same time that are different. It can be tough to line them up exactly.

Up until recently, there was no sequencing available. Just last week, Cheng Lou added in `onRest`, which allows for this kind of callback workflow. It's still not easy to write a loop without writing an infinite loop (bad). You might not need that but I thought I would call it out. As an FYI to the performance-sensitive, it uses re-rendering.

Pros:

- The animation can look really beautiful and realistic, and the spring options are nice.
- The staggering effect is pretty nice. It is also available in most JS libraries, but this one is created based on the last motion, not by duplicating the last one, so there are some nice motion possibilities.
- The addition of `onRest` is a big deal. It allows for some sequencing that wasn't available before, so if you look at the demos, you're not even seeing all it has to offer yet.
- I'd consider this currently the animation tool that plays with React the best.

Cons:

- It's not super plug-and-play like some other libraries or native are, which means you end up writing quite a bit more code. Some people like this about it, some people don't. It's not kind on beginners this way, though.
- The sequencing, due to the complex nature of the code, are not as straightforward or legible as some of the alternatives.
- `onRest` doesn't yet work for staggering parameters.

GSAP in React

GreenSock has so much to offer that it's still worth using in a React environment. This is particularly true for more complex sequencing, which is where React-Motion doesn't offer a slim solution and GreenSock shines. It takes a bit more finessing than usual, and some things that should work and do with the DOM, don't in React. That said, I've gotten React and GSAP to play nice a few different ways now so here's how I do it.

1. Hook into the native React component lifecycle methods.
   Here's how I do that. When I want it to fire right off the bat, I use `componentDidMount`.

2. Create a function that has a timeline and then call that function within something you use for interaction. I can have an inline event handler like `onClick()` , and within the custom function I have for it, I'll call another function that has a timeline within it. In a jQuery or vanilla JS setup, I would pause the timeline initially and hit replay, but I don't have to do that here.
3. Here's a nice post that Chang Wang wrote about how to hook them into `ReactTransitionGroup()` , which is a pretty cool way of doing it.
4. You can use a library like React-Gsap-Enhancer. I haven't worked with this one myself too much but there are really sweet demos. One of these is a fork of one of my own demos and I will say, though, that when I looked at the code it seemed much more verbose that what I wrote. I'm not sure if that's due to the library or React itself, or their style.

CSS in React

CSS animations has had a bit of a resurgence lately likely because it's the easiest way to go for animations in React. I like working with them for little things like UI/UX interactions. I have seen them behave a little funny if you try to chain things using delays. Other than that, they're pretty great, especially for small UI adjustments. **In terms of real-world animations on the web, sometimes CSS is Occam's Razor.**

Conclusion

This is an opinion post by someone who feverishly works with animation all the time and this is what I've gathered from pure brute force of working with these technologies a lot. My opinions might not be shared by everyone, and that's fine. I don't have any particular reason to hold allegiance with any technology except, as a developer, considering its versatility, performance, and ease of use. Hopefully, this information can save you some time.

Comments

Chris Coyier

MAY 2, 2016

This is how my brain processes why animation is weird in React:

1. React has a virtual DOM, but ultimately what you see on the screen is still the real DOM.
2. Animations are happening in the real DOM.
3. React doesn't tell you (and you don't tell it) when it's going to update the real DOM. It just does it based on state changes.
4. Any given DOM node might be being animated (e.g. it's new, so it's being slide in from the left)
5. React might decide to rip out and re-render that node. Or ones around it. Or the parent. It doesn't know about your animations built using some other tool.

So I think the best you can do is either...

1. Use an animation tool that is built specifically to work in React
2. Build animations that are resilient to abrupt DOM changes (e.g. it doesn't matter if they finish early or start late)
3. Or, attempt to be really smart/predictive about when React re-renders things in the real DOM

Kyle

MAY 2, 2016

I'm a bit of an React newbie, but this matches my thinking on the matter to a T. Some of the component lifecycle functions, particularly shouldComponentUpdate(), should provide a method for option #3.

Casey

MAY 2, 2016

Obviously it depends on the use case, but wouldn't an option be to use Canvas within React? Canvas is just a single node, so any drawing you do would not affect React's update cycle, correct?

Sarah Drasner

MAY 2, 2016

Very good point, Casey. I mentioned React + Canvas a tiny bit in the Canvas section but it's worth a mention below. I'll update the post in a bit.

Brad Westfall

MAY 3, 2016

The only issues I have animating are related to new DOM created — but this was always weird with regular 'ol JS. As Sarah mentioned, there are tools that solve this.

As far as DOM that already exists at the moment you want the animation to occur, such as a slide-in menu or something (again, assuming the menu is already there) — one can simply use the 'ol class state techniques. i.e.: The sudden presence of a class name triggers a CSS selector which is animated.

Michael Jackson

MAY 3, 2016

"React might decide to rip out and re-render that node. "

React's reconciliation algorithm is really good, and it will usually perform only the minimum # of changes on the real DOM that are required to accurately reflect the stuff you return from your render function, so it's not quite as unpredictable as you're making it sound. If you *do* have problems with React re-rendering nodes mid-animation, you are probably missing a `key` on one of your nodes. That helps React to be smarter about reconciling things.

Jimothy

MAY 10, 2016

Great article! Basically boils down to: use css for basic stuff, greensock for chaining animations, and a framework based on webgl for 3d. :)

Emanuele

MAY 2, 2016

SMIL is the only true way to make SVG animations. Everything else is just a hack. I've used it for 10 years and it works like a charm. You can create your animation in an app and play it on the browser or on a mobile phone, and it will behave the same. No scripts or css hacks needed. This is hardly true for all other methods, usually a pain to port and mantain. SMIL is solid like a rock and there are polyfills for any browser so even if one day someone is going to drop SMIL support (Chrome announced the intention to deprecate, but did not yet because the usage of SMIL is growing, especially on mobile. Even very popular apps like Viber are making their animated stickers using SMIL on Android).

Using a polyfill for SMIL is not different from using a javascript library, so why did you include GSAP but not SMIL? I surely prefer something standard, something I can export from a popular animation tool like Adobe Animate CC for example (with its plugin "flash2svg" you can export any animation you create in a single svg file smil animated, no code or css required). And the open source version of Toonz, the powerful animation software used by Ghibli studios, have in its development road map the option to export the animations in SVG SMIL. SMIL is going to live, as a native standard or as an external library like GSAP, but it is going to stay forever.

MaxArt

MAY 2, 2016

I feel you're too optimistic about SMIL's future.
Chrome developers made a solid job on Web Animation API, and on that basis they're eager to drop SMIL to make their codebase significantly lighter. And they will, be sure of that.
Microsoft won't even begin to develop SMIL support because of this new direction. You should know, because you too commented on it:
https://wpdev.uservoice.com/forums/257854-microsoft-edge-developer/suggestions/6509024-svg-animation-elements

In short, SMIL is basically dead.

fmuaddib

MAY 2, 2016

@MaxArt: maybe you missed my point. SMIL is not going away just because some browser vendor decides to. In the worst case scenario, it would require the use of an animation library, exactly like the others reviewed here.
There are already tree popular javascript libraries for running SMIL animations on browsers that do not support it natively: Eric Willigers polyfill, SIE and Fakesmile. If you are going to use a javascript library for your animations, using a SMIL library is the best choice you can do. Independently from the native support or not, SMIL is the best for SVG animations. And you can create SMIL animations in real animation tools, and exchange your svg animated assets with any graphic artist, because they are encoded in a standard format, not written in custom javascript code. Believe me: SMIL is not going anywhere. At least until a new standard format for vector animations cames out.

MaxArt

MAY 3, 2016

And in fact there's a new standard: it's called Web Animation. In the meanwhile, SMIL is a goner, together with its awkward and clunky declarative syntax. We already have a declarative way to define animations, and it's CSS. More complex and dynamic animations are simply no good for any declarative language. Everything becomes a huge pile of more and more obscure lines almost impossible to maintain.
SMIL offers no performance advantage whatsoever over CSS or Web Animation, so none is going to use a polyfill for something that no browser sooner or later will support anymore. It just wouldn't make a grain of sense.
Professional animations on the web are mostly made with GSAP anyway, which has great

performances and a nice and actually usable API. For other use cases, designers simply don't care of what's under the hood of their vectorial animations, because they use tools that take care of that for them.

Emanuele

MAY 3, 2016

@MaxArt : I love the GSAP guys, and when many years ago I used Actionscript their libs were of great help. But I'm not talking about programming here. I' talking about animations made by artists with their pen and tablets on software like OpenToonz or Animate CC, or CACANi or Anime Studio, or any other. Artists use those apps to create vector animations, they not use javascript. So how are those vector animations going to be displayed and reproduced on the web? In what format you can export them? It must be a descriptive graphic animation format, like SVG and SMIL. A standard estabilished by W3C that anybody can use, not a proprietary format that some company can use to lock people in and force them to use their libraries or their tools. SMIL is FREEDOM. It is something that makes vector animations files data usable, exchangeable and understandable by everybody. SMIL is the PNG of the vector animations. Do you remember how PNG was born? I do. The motivation for creating the PNG format was when in early 1995 it became known that the Lempel–Ziv–Welch (LZW) data compression algorithm used in the Graphics Interchange Format (GIF) format was patented by Unisys. Gif used proprietary technology. PNG was FREE. Today we have only ONE open standard vector animation format, SMIL. Exporting an animation in "GSAP" dependent code? A proprietary library that can change at any time and lock developers and artists for ever? Or even breaking compatibility, making all vector animations suddenly "obsolete" and "unplayable", just because the company decides to force an upgrade to users to make more money? Or even dictate specifications? No thank you. If you don't understand the importance of using an open standard, then you should study the story of the web, and of all information technology for that matter. Because proprietary formats for Data turn out always bad and damage the entire ecosystem, including the proprietary of the format. Just look at the failure that Silverlight/XAML was. Or Flash. They were proprietary, and then they become walled gardens. And walled gardens always dry up in the end. The more they fight for staying closed, the more their walls isolates them from the world, until nobody uses them anymore. This is why I store all my vector animations files in SVG SMIL format. Software changes, browsers and libraries changes, but an .SVG file will always be the same, because it is a standard, like PNG. This is why in the end SMIL will win. It would win even if it was not supported by browsers but only by javascript libraries like SIE or Fakesmile. Because it is a standard. And being a standard you can have not one library like GSAP, but THOUSANDS of independent javascript libraries to play it. You can save it from AnimateCC or OpenToonz and reproduce it in a Mobile App you are developing like Viber, or in a web page, or in a videogame for a console using something like Unity SVG Importer. As you can see there is a clear advantage in having the code separated from the data. And unfortunately SVG is the only data animation format you can save. GSAP or any other library are code, and they are dependent on the platform and on the specific version of software, like the browser. A data format for animation instead it is universal. Clear separation between code and data is impossible with GSAP or other animation libraries written in javascript. After all, artists do not draw animations in code. They draw them with a pen and a graphic tablet. SVG SMIL is the only true data format they can save in. And this is not gonna change because some blind manager at Google decided to pull the plug on SMIL in Chrome to save some money.

Jack

MAY 4, 2016

@Emanuele you make a lot of valid points. I understand the fear of using anything that doesn't have an official standards body endorsing it. However, please consider that:

1) Libraries like GSAP are written in standards-based tech (JS), so it's not like in 5 years GSAP-based animations would suddenly stop working. Any vanilla JS you write yourself has the same risk, as do the SMIL polyfill libraries you mentioned.

2) Just because something was ratified by a standards body doesn't mean it won't be phased out (or ever implemented at all – look at IE which has zero support for SMIL). Ultimately browsers decide which tech they support regardless of "standards".

3) GSAP has been around for a decade and has ***never*** changed a free tool to paid, or raised prices on anything or pulled any underhanded tactic as you theorized could happen. Trust is super important to us. But it's fair to point out possibilities.

4) In some ways, a library like GSAP is "safer" because it's far more nimble and can react to browser bugs and inconsistencies much faster than standards bodies or browser vendors. I think GSAP's track record certainly bears that out.

The web would be crippled without standards...yet imagine how limited the web would be today if we ***only*** relied on standards (no libraries/frameworks/plugins)! I also don't think it's fair to characterize standards as so impervious to change or deprecation. I'd guess that in 5 years, GSAP-based animations will run more consistently across browsers than any SMIL animation created today, and that's ultimately what matters to a lot of developers.

Emanuele

MAY 5, 2016

@Jack You are right about all, Jack. But you still miss the most important point: all you said makes sense for programmers. But how can an artist work with GSAP? How can he save and store an animation he created in OpenToonz or Animate using a specific javascript library? You need something that works indipendently from a specific library. And Data should be separated from Code because you cannot be dependent on the implementation. Also proprietary libraries like GSAP are cool on the web, but there is no GSAP for mobile apps. If I want to reproduce my animation on an iOS app, how I'm supposed to do it? Even if I buy some library that is capable to playback the messy package (html+javascript+css) on iOS, it should emulate a javascript VM, and that it is slow as hell (Also on iOS VMs are not allowed at all, but that is another matter). The alternative would be for someone to code a "GSAP" specific viewer for iOS, but that would be against your copyright, because such library would have to rewrite natively the same GSAP library functions.
As you can see, no matter how good is GSAP for programmers, it is not useful for artists and animators. Artists that make animated SVG icons for a living have only one option, and that is SMIL. Of course here everybody is a programmer, me included, so we don't care about artists and graphicians. But I assure you that if those people are kept away from this technology, you'll never see the quality of animations improve much beyond the very little that you can do creating animations programmatically. It would be like forcing people to make bitmap images on the web with a bunch of .putpixel() function via javascript instead of using JPGs and PNGs created in Photoshop. I doubt that the web

would be a pretty sight.
This is why a data standard for saving vector animations is NEEDED. SVG SMIL does that, and does it well. There is no other standard to replace it. Maybe one day another one will be born. But until that day, SMIL will be the only serious option for professional artists and animators.

Emanuele

MAY 6, 2016

Here is an example SVG SMIL file with a cartoon episode with embedded audio in synch of the kind that only an artist can do:

https://imgh.us/LoveDota.svg

The animation with the audio is reproduced correctly on Chrome and Firefox.

On Safari there is no audio, but the bug is currently under review:
https://bugs.webkit.org/show_bug.cgi?id=155778

Do you think that the animator that did this with Animate CC is going to doing the same using GSAP or any other javascript library? I don't think so.
Without SMIL you just lose the option of having complex animations like that on the web.
This is why SMIL is going to stay.

Sarah Drasner

MAY 6, 2016

If you're worried about automated output, you might consider bodymovin', one of the libraries listed above. It outputs to SVG and JavaScript. Thanks!

Emanuele

MAY 6, 2016

Thank you @Sarah, I love the Bodymovin add on for After Effects. But unfortunately it still doesn't export a data file that you can exchange like a standard format between different graphic applications. It is not a file format you can load like SVG. It is an entire web site, composed of HTML, Javascript, CSS and SVG files put together for a very specific platform and technology. It is not platform independent like SVG or PNG is. You cannot just save in that format your vector animations, and using it as an animated icon on a Mobile app for example. Of course you can use the native After Effects project file as the animation format, but it is a proprietary format. So it has all the incoveniences of a proprietary format, the most important is that you cannot make a tool for edit such file without paying royalties to Adobe. Also Adobe dictates the format, and it is in their best interest to keep it locked, obscure and linked only to their tools.
With SVG SMIL instead an artist can create an animation in any software tool, export and load it in any another different tool for compositing or for building an UI for a mobile app, or a web site, or a videogame UI. An SVG SMIL animation is a standard and because of that it allows artists to save animations in it and use them everywhere, from vidogames in Unity with the SVG Importer, to professional TV video productions using OpenToonz or

AfterEffects, to Adobe Experience Design or Invision for designing applications UIs with animated components, in a ePub eBook school textbook, or in a mobile app. You can use that SVG file along the entire production chain, efficiently and with the reliability of a PNG file, on any platform present and future. You do not have to worry about proprietary lockin or interpreting some javascript code written for a browser and trying to convert it in some native code. An animated button made with SVG SMIL works perfectly both on a web page that in a mobile app, that in a epub ebook, that in a videogame UI, or in any video. Can you say that of an Adobe After Effects project file? Or of a web package exported from the Bodymovin plugin? No, you can't. There is no way to make that work in a production chain. Bodymovin is a very specific and custom solution. It is not PNG. SVG SMIL is the PNG of vector animations. And every developer out there can just add support for it in his software like it can for PNG images. We should defend the treasure that SVG SMIL standard is. It is something that was very hard to achieve. Many greedy corporations fought hard to avoid that for years, and they succeeded in slowing down SVG adoption for years and in crippling it vetoing the addition of basic functionalities like text wrapping. Only now, thanks to the great work of extraordinary people like you, like Sara Soueidan or Dmitry Baranovskiy, SVG is starting to get its revenge. It is time for people to know that a better web was already possible 10 years ago, but was deliberately stopped on its track because of the greed of some private companies. But you can't stop a good thing for too long. Eventually people will find about it. This is why even the latest move by Google to kill SMIL is going to fail in the end, like the move to destroy SVG by Microsoft first and by Adobe later, failed in the end. Openness and standards always win against closedness and proprietary. The web itself is a banner of truth proving this.

MaxArt

MAY 7, 2016

Honestly, Emanuele, you sound a little like a broken record. I don't see why you're trying to buy everyone's support with your words, but that's not going to happen. Studio Ghibli is exporting using SMIL, Viber is using SVG with SMIL... eh, honestly, who cares? I bet ***they*** don't care either.

You saying you're not standing for a developer point of view, and that's fair, but the point is that every tool that's not for developers, should never burden its users with technical details like what's used to animate an SVG. For a designer/animator, is their tool is exporting using SMIL of Web Animation should make no difference, should be completely transparent. I shouldn't remind you that SVGs can include Javascript code, should I?

It's just that from a development point of view SMIL is terrible, so that's why it's seeing the sunset. It's as simple as that. And this post deals with development, here's the catch. GSAP, in fact, has been talked about because it's great for development. Even the raw Web Animation API is waaay better than SMIL.

Now, we're kind of in the middle of a transition and maybe that's why you're worried/confused/disoriented, but trying to keep your world stuck in the past is no good. It will take a while, but ultimately everything will be fine, since the industry demands it should.

Emanuele

MAY 7, 2016

@MaxArt First of all, SMIL is not terrible at all. It's the most elegant animation description language ever designed. Ask any IT academic. Second: the companies care about SMIL

because it is a data description language. In other words: it is something that contains the informations about the vector animation. It doesn't tell you how to run it. It doesn't contain CODE that need to be RUN in their application. Believe me: even if you define a standard way for saving vector animations in a SVG+JAVASCRIPT file, no one software developer company is going to load it in its application. Because loading and running untrusted code inside your application it is NOT safe. Do you really think that Adobe, Serif or Autodesk, Corel, Microsoft, Google, or any other company is ever going to choose a file format that contains potentially malicious code? Code that you must run to play the animation? It would be the worst software design decision ever. No software engineer is going to define a file format with code to run in it. The troubles are not only in the danger of untrusted code, but also in something that you need to RUN to decode it. You don't know the code inside it, so the only way to know what is going on at frame 291 is to play the entire animation. A data description format instead provides you with the informations, not with the code. You don't need to run anything. You know exactly what the frame 291 is gonna be, because that frame is defined by those informations in an unique, predictable and standard way.
Also, interpreting a language every time you load a file it is slower than using your native app code. Performance wise it is a killer. It is like the MPEG standrd forced you to run a Javascript piece of code contained in the .mpeg file at runtime, and that for each rendered frame. Only the most stupid engineer would define a video standard like that. And yet you think it is a good thing for SVG animations? I don't believe you. I think that you know ho inefficient and stupid is using that way to store animations, and I think you know that the only reason because we are doing it is because the Web is a mess. There is no rational vision behind it, but only a bunch of business competitors that are making everything in their power to avoid that a single standard, one that they do not own, will affirm itself. They don't care about a true standard for the web, and even less about a rational standard. All they care is to lock developers in, force their libraries and tools on us, and using the web standards war to win and control the mobile app ecosystem. Why do you think that javascript is still the language of the web? Javascript is not even a true programming language. It is a mess created by Netscape in a week! It is the worst language ever, and yet we run the world wide web on it! There are many modern and wonderful languages that the W3C could approve tomorrow morning as a new web standard. So why they don't do it? Because they need the approval of browser vendors, and they don't want to agree on something well designed that works best. They want to fight each other. And this is why artists that today need to create a simple animated icon are now forced to use 4 different language standards (HTML, CSS, Javascript, SVG) to make something that a rational designer would make using just SVG. Lucky for us in many other fields people are increasingly using pure SVG for vector animations. Mobile apps, videogames, video editing... in all those fields people are using SVG. Not mixed with HTML, Javascript and CSS. Just SVG. And it does its job beautifully. But on the Web there is a browser war, and so a simple standard file format for vector animations is heresy for the greedy browser vendors. Well, they are not going to win. Because the human mind is capable of doing something so much better than the mess that is the web now, and if we follow our reason instead of obstructing it for blind greed, we can create amazing things.

Jeffrey Sweeney

MAY 2, 2016

While it's true that older canvas graphics/animations might not render in high DPI, the capability to do so has been around for awhile. It's just a matter of scaling the canvas and

context dimensions up while keeping a constant CSS width/height.
https://jsfiddle.net/yokf2qmk/

This old article describes what's going on in better detail:
http://www.html5rocks.com/en/tutorials/canvas/hidpi/

Rachel Smith

MAY 2, 2016

Speaking from personal experience – I still often create my canvas demos in non-high DPI just for a better frame rate. I prefer to have smooth animations over crisp animations on canvas.

Kyle

MAY 2, 2016

Thanks for the helpful post, Sarah!

In case others need more examples of React + GSAP, my site has some: https://kylegach.com, and the full source is available on GitHub.

Sarah Drasner

MAY 2, 2016

Awesome! Thanks.

Bool

MAY 2, 2016

Um, GIF is not on this list. What gifs? (NB: plural of "gif" is pronounced like "gives")

James Fhrencho

MAY 2, 2016

Bool, listen pal. Gifs clearly are not W3C CSS3 spec, I suggest you move on and ADAPT to the TIMES.

Bool

MAY 2, 2016

Au contraire. https://www.w3.org/Graphics/GIF/spec-gif89a.txt

Your move. Or should we say, "checkmate?"

MaxArt

MAY 2, 2016

GIFs aren't something you can programmatically handle and have no place in this discussion.
(And if you want to talk about animated images, at least don't pick a format that's abysmally old and inefficient like GIF.)

OSUblake

MAY 2, 2016

I know you can't compare every tool out there, but I think a big one you didn't include is Pixi.js. I'm not even sure which of your categories it would fall under. It's a 2D canvas library, but it also has full WebGL support. It's like Flash, but without the GUI.

Like most canvas libraries, Pixi uses a full scene graph, so it's not just a blob of pixels. There is an actual hierarchy, which means you can map HTML elements and events to it. What does this mean? You can make canvas accessible.

Check it out. You can tab through all the buttons, and interact with them by pressing space/enter.
http://www.goodboydigital.com/pixijs/accessibility/accessibility.html

Sarah Drasner

MAY 2, 2016

Haven't really heard very much about this one. Looks cool, Blake! Yeah, I kept it to technology that I've either worked with or am excited about, as I stated in the article. Nice find, though.

Sale

MAY 3, 2016

You can have motion path with SVG – using getPointAtLength() on Path element.

Arthur Stolyar

MAY 3, 2016

> It's not retina-ready. Unlike SVG animation which is resolution-independent, most of the canvas animations I see are not crisp on retina displays.

Because nobody of authors care about it. As all in land, retina support is also manual, but it's ready.

Sarah Drasner

MAY 3, 2016

For clarification, retina-ready means resolution-independent. You can author high pixel densities in Canvas, but it it requires thinking about and authoring. That's the distinction we're making here, against vector where that isn't required.

The pros and cons are not meant to be a dig on the technology, but rather information that helps people decide on what to use. That said, I clarify that section with an extra sentence since people seem confused.

Manuel Matuzovic

MAY 4, 2016

WOW! Thank you!

Thomas Mak

MAY 4, 2016

> It's not retina-ready. Unlike SVG animation which is resolution-independent, most of the canvas animations I see are not crisp on retina displays.

It is not too difficult. Make the canvas dimension 200% in content (or 300% on 3x screen) and make it 100% in CSS. Now your canvas look sharp in retina displays.

Sarah Drasner

MAY 4, 2016

Please see above comment, thanks.

Emanuele

MAY 5, 2016

This is not easy to do because of the following reasons:
1 – You don't know the true resolution of the current display.
2 – There are now too many display with too many resolutions to be handled by some switch statement.
3 – Very few mobile browsers provide an API to query for the true display resolution
4 – Retina or higher density displays are now becoming common even on desktop computers
5 – Many users want to pinch-zoom on the page, because their display is too small (disabling the zoom on a web page makes all 3.5 inches devices owners angry, and it is a bad practice). Zooming makes no difference in sharpness for SVG elements and icons, but it makes canvas elements ugly with pixelization and aliasing.

This, combined with the fact that canvas elements are not indexable by search engines like SVG is, makes canvas pretty much useless for the majority of the web sites, with the exceptions of online web games, where search engine indexing is not needed and the users rarely read text that need the page to be pinch zoomed.

Thomas Mak

MAY 5, 2016

@Sarah

Thanks. I read it after I post this comment.

> retina-ready means resolution-independent.

I would say, canvas is not resolution-independent, but it is retina-ready.

> You can author high pixel densities in Canvas, but it it requires thinking about and authoring.

All you need is a generic utility function to enlarge the canvas and call it on every canvas. (It's CreateJS code, but the same concept applies to all canvas API). It's a bit planning at the beginning but the same fix applies to all projects without further thinking.

Thanks @Emanuele Also.

> On point 1, 2, 3, 4 about true resolution

We don't need the resolution. We aren't talking about responsive, are we?

What we need is the density per pixel. (per point, technically)
`Window.devicePixelRatio` provides the 2x, 3x information, and most browsers support it on both mobile and desktop.

> Many users want to pinch-zoom on the page, because their display is too small

You're right. Canvas isn't vector. It's bitmap and zooming the page with canvas is the same of zooming a page with PNG/JPEG image. But it doesn't void the point that canvas supports retina. Zooming is about vector vs. bitmap / layout strategy, but not the point of Retina.

> This, combined with the fact that canvas elements are not indexable by search engines like SVG is, makes canvas pretty much useless for the majority of the web sites, with the exceptions of online web games, where search engine indexing is not needed and the users rarely read text that need the page to be pinch zoomed.

It depends from project to project. Assuming you have added the best accessibility practice to canvas, search engine— and disabilities—shouldn't find it difficult to understand the canvas' content. That applies the same to having text in `<img>` and trying to provide the meaning of the image to search bot or visual impaired people.

> makes canvas pretty much useless for the majority of the web sites.

I'm not bias on canvas, or any technology. They are designed for a purpose. if they are not yet obsoleted, they're still providing their value. So I wouldn't agree that canvas is "useless for the majority of the web sites". It is just you haven't find a way to use it in your project.

Thanks again :-)

JSteunou

MAY 5, 2016

I am surprised people still bring velocityjs which was nice a few years ago but drop clearly far behind others like animateplus https://github.com/bendc/animateplus which is my favorite animation library waiting for waapi to be usable everywhere : es6 first, umd build, npm published, good perfs, less weight...

yoonsun lee

MAY 10, 2016

I'm in the SMIL camp. Why wrap up renders in Javascript, when you have SVG/XML with fallbacks (HTML5, Web GL, CSS linked to locally loading symbol libraries) made with Greensock/Animate CC? You can actually design with branded content and preview what you build with your client. There is tremendous value in being able to change what you present pre/post transpiling. I'm an ex Flasher. I work with pictures.

Brian Birtles

MAY 10, 2016

This was a good summary! I'd like to clarify some of the performance information, however.

Firstly, the primary advantage of CSS Animations/Transitions in terms of performance is not offloading to the GPU. GPU composition of animated layers can be applied to scripted animations too (provided they are animating the same kinds of properties: transform, opacity). `will-change` helps here, but many browsers automatically detect script-generated animations and apply this optimization after a couple of frames.

The performance advantage of CSS Animations/Transitions is that these animations that can be composited on the GPU, can also be delegated to a separate thread or process. This allows them to continue running smoothly even when the main thread is busy. That's something that scripted animation simply cannot do unless they use CSS Animations/Transitions or the Web Animations API under the hood. (And it's a very significant optimization on low-end mobile devices!)

As for the performance of the Web Animations API, it is identical to CSS Animations/Transitions. It is exactly the same code running both. The whole purpose of Web Animations API is to be a lower-level API on top of which CSS Animations/Transitions run. (And SMIL too, for that matter.)

Also, regarding React, be careful of things like `react-router` and `ReactCSSTransitionGroup`. If you're trying to animate in the next bit of content in response to a URL change, this setup will end up adding the content to the DOM at the point when you want to animate. That's going to introduce significant delay to the start of the animation. You're better off to have the content you want to animate already in the DOM but simply moved offscreen or made transparent (and marked `will-change: transform` or whatever). Then you'll get an instant response when you navigate.

Emanuele

MAY 10, 2016

Thanks for the precious infos, Brian.
You are the one currently responsible for writing the most important web standard EVER: SVG 2.0 Animations (link: https://svgwg.org/specs/animations/).

Considering your key role for the future of the web, here are some questions:
1 – Will the specs be completed in time for the official SVG 2.0 finalization?
2 – Are the SVG 2.0 Animations specs going to be backward compatible with current SMIL animations?
3 – According to your document, the aim of the SVG 2.0 Animations specs is to be more similar to CSS and then making possibile for browsers vendorsa to use the same code for both CSS and SVG Animations. Does that mean that we can see ping-pong animations (i.e. animatable attribute defining the animation direction) added to the SVG Animations specs?
Thank you! :)

This comment thread is closed. If you have important information to share, please contact us.