CSS-TRICKS

SVG Animation and CSS Transforms: A Complicated Love Story

BY **JACK DOYLE** ON NOVEMBER 10, 2014

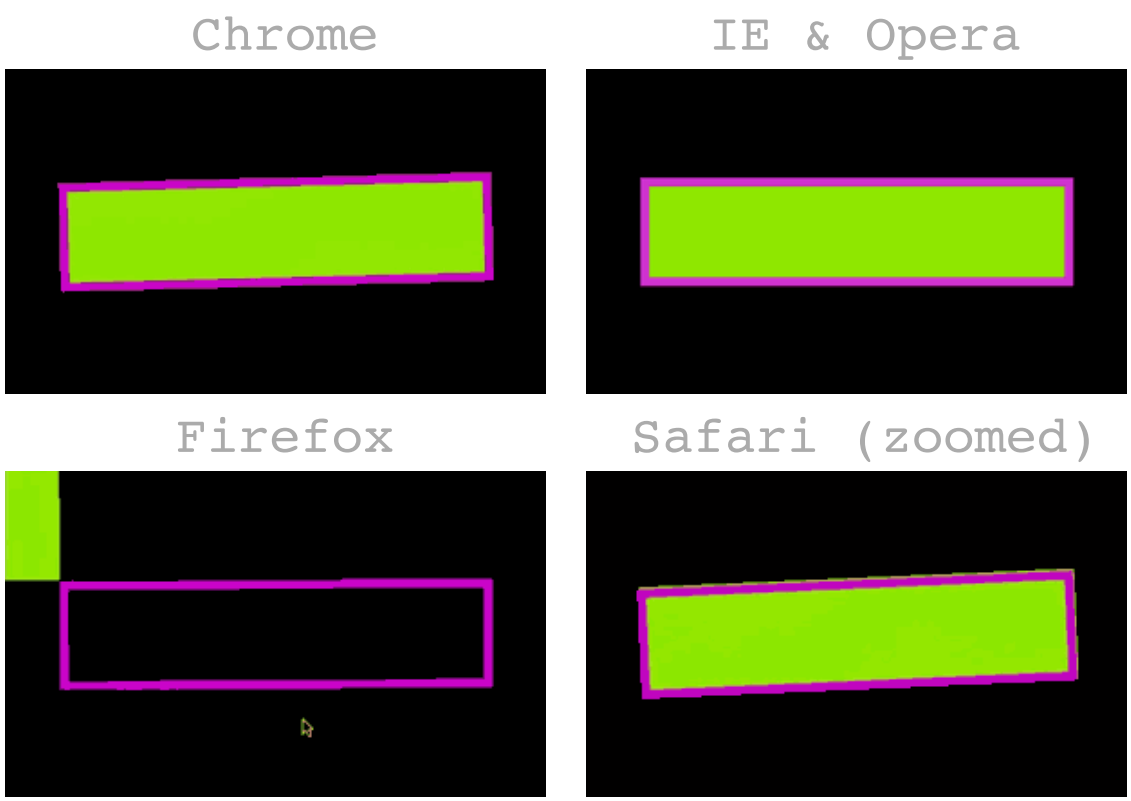
The following is a guest post by Jack Doyle, author of the GreenSock Animation Platform (GSAP). Jack has been deep in the woods of web animation for a long time, trying to make it easier and better. He's written here before, talking about how JavaScript animation can be the most performant choice (Google even recommends it). This time, he focuses on SVG animation, some pretty scary issues you may come across while manipulating them with CSS, and how you can solve those issues.

SVG is all the rage these days, and browser support is generally excellent...with one glaring exception: CSS transforms. This is particularly painful when it comes to animation because scale, position, rotation, and skew are so fundamental.

Buckle up, because we're in for a bumpy ride. But don't worry; this complicated love story has a happy ending. In this article, I'll walk you through some of the problems and then show you a technique that harmonizes behavior across browsers and is baked into the latest version of the GreenSock Animation Platform (GSAP).

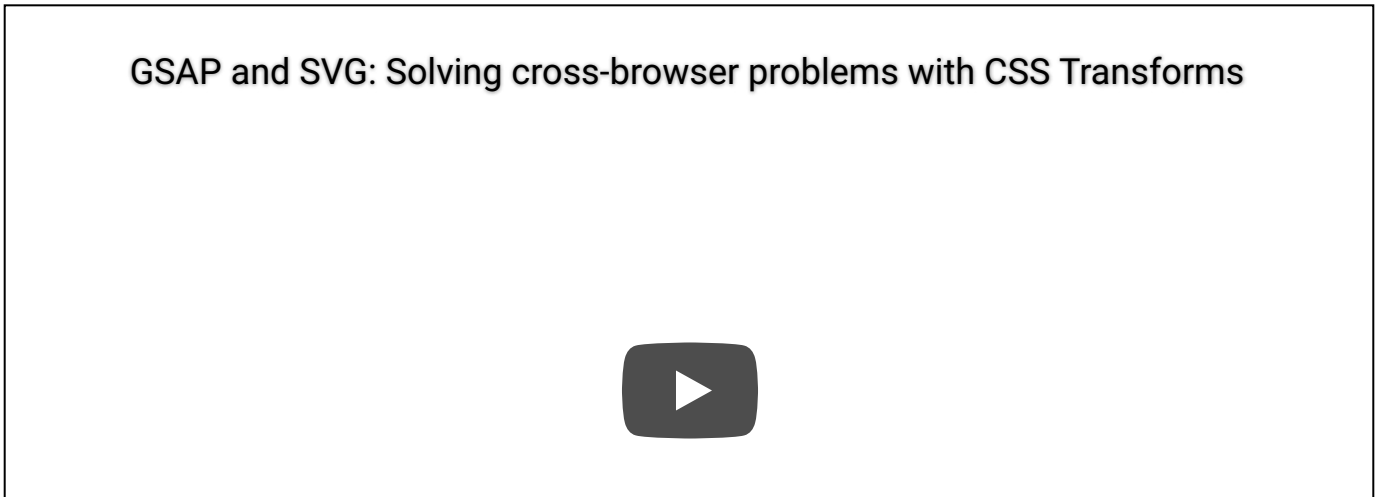
Browser bugs & inconsistencies

First, check out these animated GIFs showing the exact same CSS animation of two `<rect>` elements in various browsers (at least as of November 2014):



- **IE** and **Opera** don't honor CSS transforms at all on SVG elements. Instead, you must assign the value to the `transform` **attribute**.
- **Firefox** doesn't honor %-based origins.
- Zooming in **Safari** breaks the sync between %-based and px-based origins.
- **Firefox** doesn't recognize keyword-based origins like `"right bottom"`, and **Safari** alters them when the zoom is anything but 100%.
- In **all browsers**, px-based origins are measured differently for SVG elements than other DOM elements (see below).

Here's a video demonstrating those bugs:

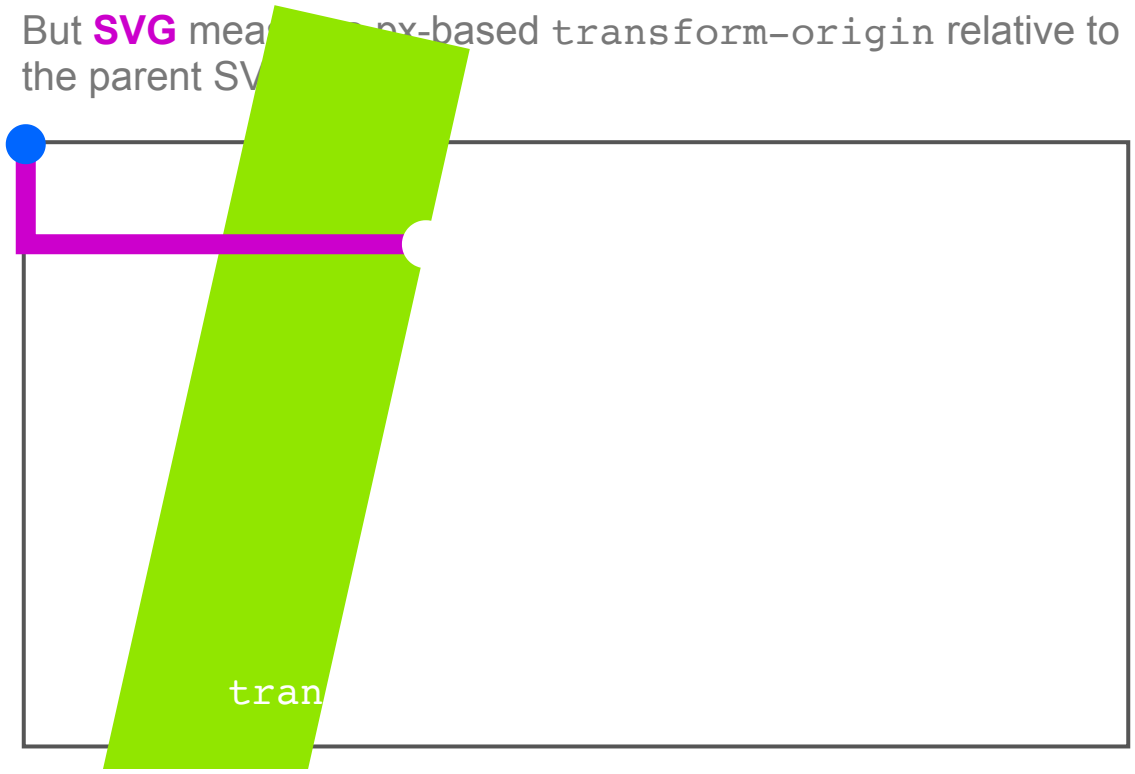


Think of the `transform-origin` as the point around which all rotating and scaling occurs, as if a pin was holding it in place there,

Where's my transform-origin?

Normally, if you want to rotate or scale from the center of an element, you'd set `transform-origin: 50% 50%` . Percentages are relative to the element's own native size, so 50% of its width and 50% of its height lands smack-dab in the middle. All major browsers support %–based origins for regular DOM elements, but only a few support them for SVG elements.

How about px-based values? Support is excellent, but there's another problem: as shown below, SVG measures px-based values relative to the **parent** SVG's 0,0 coordinate whereas every other DOM element (like a `<div>`) measures it relative to its **own** top left corner. So if you want to center it, you have to do the math to plot the coordinates in SVG (you can use `getBBox()` JS for this). Consequently, you cannot apply the same CSS to a regular DOM element and an SVG element and expect them to behave alike.



A solution

It'd be nice if I had a pure CSS trick that'd work here, but since IE and Opera completely ignore CSS transforms on SVG elements, and there are bugs in most other browsers, we'll rely on one of the incredible strengths of JavaScript: its flexibility. Even if every browser fixes their bugs next week, we still couldn't get around the fact that the SVG spec handles px-based transform origins very differently than other DOM elements, so CSS transforms are bound to that spec. As far as I can tell, the only viable longer-term solution is a JS-based one. Plus JS gives us a lot of other benefits, but that's a subject for another article.

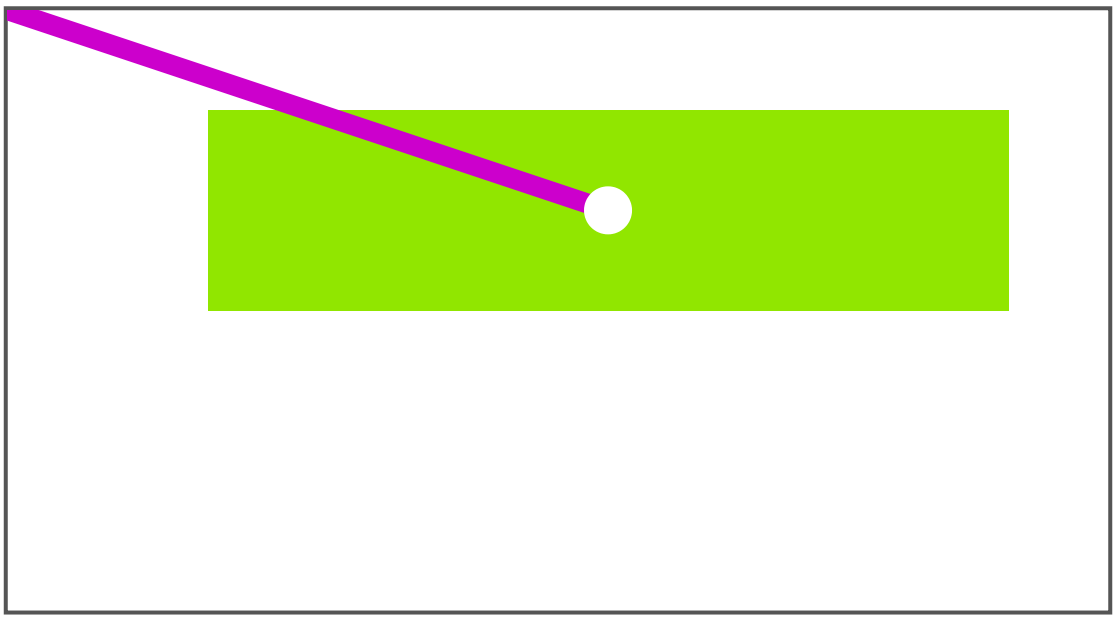
The goal: be able to animate various transform properties (rotation, scale, position, and skew) of SVG elements in the same way as "normal" DOM elements while delivering identical behavior across all major browsers. We should also be able to set the transform-origin using standard CSS-like values including percentages, px, or keywords like `"right bottom"`. Oh, and it must be *FAST*.

> In case you're not already familiar with GreenSock's GSAP, it's a high-performance, professional-grade JavaScript animation library with unmatched sequencing tools, runtime controls, and flexibility (animate literally anything JavaScript can touch). Google recommends it for JS-based animations, and it's up to 20x faster than jQuery.

Here's an overview of the technique employed under the hood in GSAP (we won't get into the matrix math because it's beyond the scope of this article). The demo below illustrates what GSAP does when you attempt to spin an SVG `<rect>` around its center by specifying a transform-origin of `"50% 50%"`.

Step 1: use `getBBox()` to calculate the x/y offsets from the SVG's 0,0 coordinates to the desired transform origin. In this case, it's x:300, y:100 (white dot).

NEXT



The resulting matrix() string should be fed to either the element's CSS style or the `transform` attribute, whichever is necessary for that particular browser.

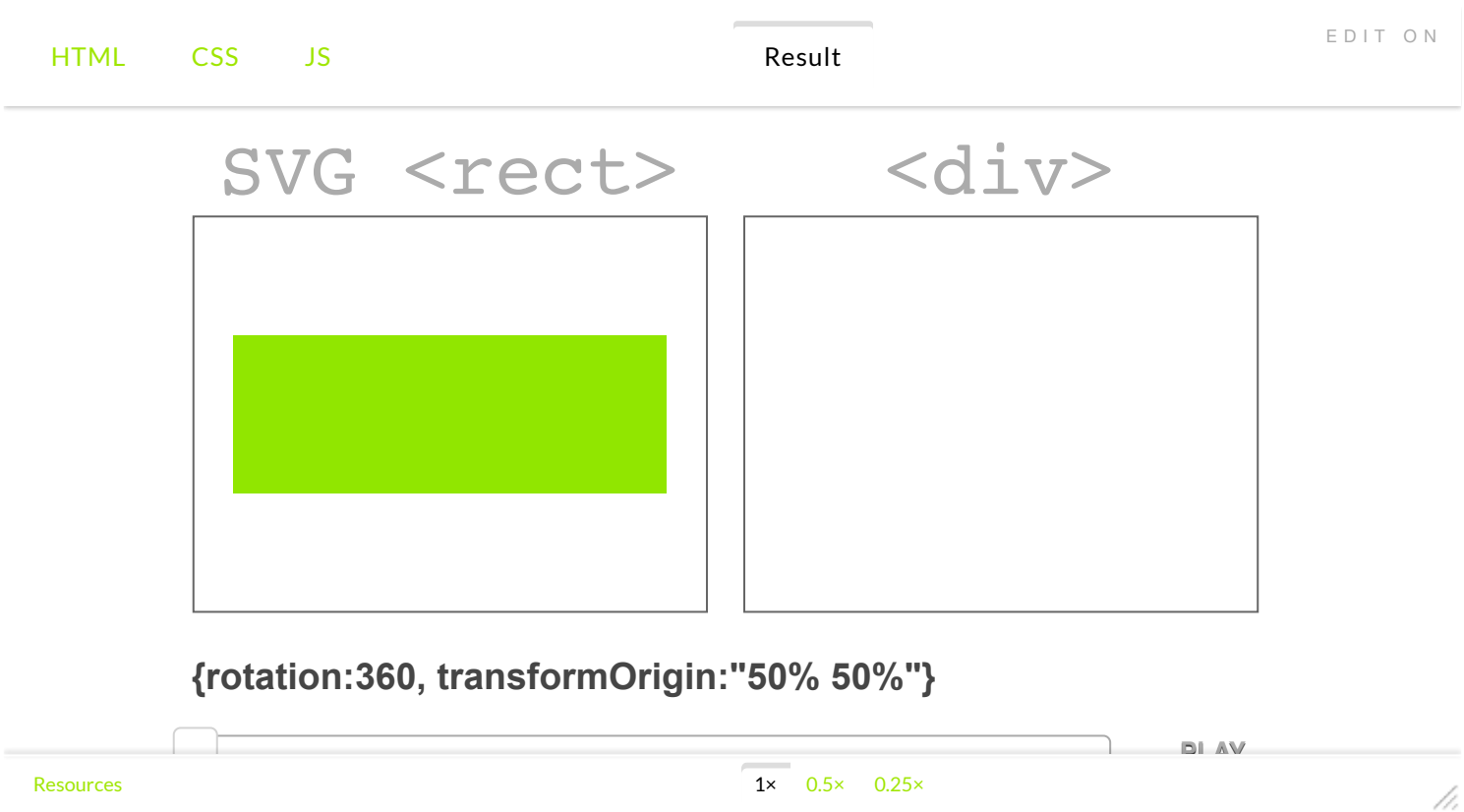
If that made no sense to you, that's okay - it all happens automatically for you under the hood in GSAP.

The result

Not only do we get harmonized behavior across browsers and a consistent animation API for SVG and non-SVG elements, but the animation code we have to write is tiny compared to the (broken) pure CSS animation.

```
Works the same for SVG and DIV

TweenMax.to("#svg, #div", 2, {
  rotation:360,
  transformOrigin:"50% 50%"
});
```

Here's a demo that shows a sequence of several origins. Notice everything works the same for the SVG `<rect>` and the `<div>`:



Tips for animating SVG with GSAP

I'm not going to explain GSAP's API here (see the getting started article for that), but I'll offer a few SVG-related tips. Snag GSAP on github or download it from GreenSock.com.

- Rather than combining all of the transform components into a single "transform" string like in CSS, GSAP lets you define each one independently, and you don't need to worry about order-of-operation (it's always consistent). Also keep in mind that GSAP uses `x` and `y` to refer to `translateX()` and `translateY()`, and `rotation` for `rotate()`. Everything else is the same (`scaleX`, `scaleY`, `skewX`, and `skewY`).

CSS (doesn't work consistently for SVG)

```
#yourID {
    animation-name: myAnimation;
    animation-duration: 2s;
    animation-timing-function: ease-out;
    transform-origin: right bottom;
}
@keyframes myAnimation {
    from {transform: none;}
    to {transform: rotate(270deg) scaleX(0.5) translateX(100px);}
}
```

Equivalent GSAP (works in all major browsers)

```
TweenMax.to("#yourID", 2, {
    rotation:270,
    scaleX:0.5,
    x:100,
    transformOrigin:"right bottom"
});
```

- You can pass in the raw element as the target (first parameter), or selector text, or an array of elements.
- To animate numeric **attributes** of SVGs (rather than CSS properties), use an `attr:{}` object like:

```
Animate numeric SVG attributes

TweenMax.to("#circle", 2, {attr:{cx:200, cy:300, r:20}, ease:Power2.
```

- You can also use **3D** properties like `z`, `rotationX`, `rotationY`, and `perspective` but some browsers like IE and Opera won't recognize those for SVG elements.
- It is best to set the `transformOrigin` directly via GSAP rather than in CSS because various browsers report them inconsistently via `getComputedStyle()`.
- You can animate just about any CSS value including color properties like `fill` and `stroke`.
- If you want to do much sequencing, I'd highly recommend watching the videos here and here. Sequencing is one of the most significant strengths of GSAP.

More resources

- Sara Soueidan wrote a fantastic article about animating SVG through SMIL.
- Mega list of SVG-related resources here on CSS-Tricks.
- Check out Joni Trythall's gorgeous Pocket Guide to Writing SVG.
- Myth Busting: CSS Animations vs. JavaScript

Comments

Brian Birtles

NOVEMBER 10, 2014

For Firefox, transform-origin is simply not supported for SVG. It was implemented but backed out due to performance issues. There's been some work recently to tweak the patch and reland it. If someone wants to look into it, see bug 923193.

Daniel O'Connor

NOVEMBER 10, 2014

I'd love to see more articles on the cross-browser SVG inconsistencies. I've been running into a bunch of them myself.

William T

NOVEMBER 11, 2014

Your way of explaining is great, as well easy for readers to understand. Wow it might have taken lot of time to make it o much attractive and reading friendly. Great work looking forward for more such work. I will try to make my work so much convincing as yours :)

m3lvin

NOVEMBER 11, 2014

Interesting digs on how browsers render SVG animation. Thanks Jack :)

David Kizler

NOVEMBER 12, 2014

Anyone know if these browser issues only apply to inline SVG? Or are there also concerns with img tags ending in .svg?

Amelia BR

NOVEMBER 13, 2014

I'm not sure how you want to animate your SVG images, so here's a quick breakdown of the two options:

If you want to animate the content within an SVG used as <img>, you have to use SMIL animation in the XML markup or CSS animation in an embedded stylesheet, both within the main .svg file. No browsers support external resources or JavaScript within image files.

If you want to move the entire <img> element around on your webpage, however, you can use the same CSS transforms as for any other type of image (or any other element of the webpage). Both CSS animation and JavaScript-based animation should work. SMIL is not an option because the <img> is an HTML element, not an SVG element. The coordinate system used for transform-origin would be based from the top left corner of the <img> element, according to the box-sizing property in effect, and the default origin would be the center of the image.

Otto

NOVEMBER 12, 2014

Nice work!

Was looking forward to seeing this!

about transformOrigin: can you make it behave as normal SVG? Cos at the moment, you are making it behave like html, but sometimes (to me, “this very moment”) I’d like to be able to rotate things from a single point.

Jack

NOVEMBER 12, 2014

Sure, Otto, you could use getBBox() to calculate the offset from the parent SVG canvas’s origin. Here’s a simple function you could reuse in your tweens:

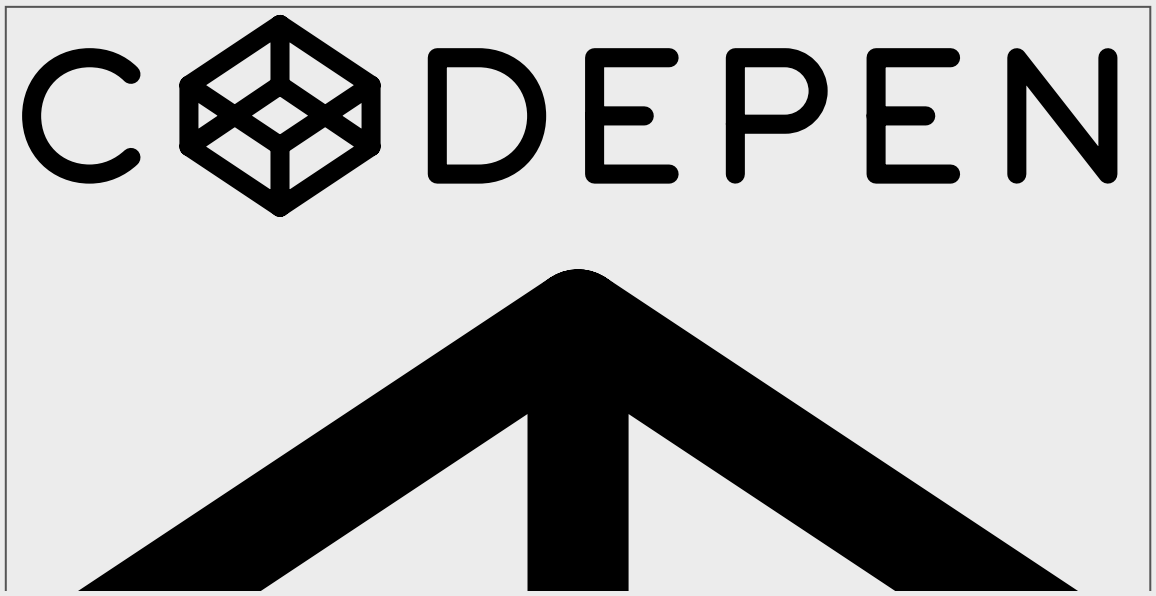
```
function normalizeSVGOrigin(element, offset) {
    var bounds = element.getBBox();
    if (typeof offset !== "object") {
        offset = {x:0, y:0};
    }
    return (offset.x - bounds.x) + "px " + (offset.y - bo
}

var rect = document.getElementById("svgRect"),
    point = {x:100, y:200};
TweenLite.to(rect, 3, {rotation:360, transformOrigin:norm
```

So you just feed in the SVG element and optionally an offset/point around which things should rotate/scale/skew and it returns the appropriate transformOrigin, like "-150px -75px". It'd obviously be different for each element which is why I wrapped it into a function, so you can reuse it and it does the work for you. Does that help?

Otto

NOVEMBER 13, 2014

Did not try it yet, but please have a look at this code example....



As you will see, my problem is a bit deeper: I need to animate elements with stuff in it.

In the example, I am adding a small to work as my pivot, so I would not need your function.

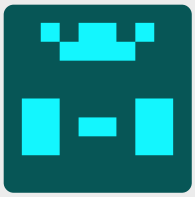
It works on Firefox and Chrome, but not on IE11.

Jack

NOVEMBER 13, 2014

Otto, this is probably something that should be handled in the GreenSock forums (http://greensock.com/forums/) rather than in the comments here, but the problem had to do with the fact that your center "g" element was an orphan when you set the x/y; browsers handle SVG elements differently when they don't have an SVG parent, thus it triggered the logic in CSSPlugin to evaluate it accordingly. I'll add code in 1.14.3 to work around that, but for now it should be as simple as flipping two lines in your code.

```
svg.appendChild(center); //append it first.
TweenMax.set(center, {x: 250, y: 250});
```



Eugene

NOVEMBER 13, 2014

I encountered this problem 2 weeks ago, solved with js too, but i was using snapSVG. It has the X and Y center in selected object properties so you can easily get center and animate.

Cheers.

Robin

NOVEMBER 20, 2014

Is there any way that greensock can animate colors of paths and rect's? I've tried animating individual properties with greensock but I've not not much luck.

Jack

NOVEMBER 20, 2014

Sure, it should be as simple as:

```
TweenMax.to("#rect", 3, {fill:"#900", stroke:"#0cf"})
```

Here is a basic codepen: http://codepen.io/GreenSock/pen/vEEjeQ

Paul Wilkins

NOVEMBER 20, 2014

I've been noticing that all of your headings below the main article seem have a negative bottom margin, resulting in an inconsistent look. When contrasted with the way headings are handled on the rest of your page, something seems to be off.

As you seem to be someone who cares greatly about the look and feel of your site, I'm left wondering if this is a deliberate design decision, or if it's a situation that's ripe for improvement at some stage.

wilfred

DECEMBER 18, 2014

Hey,
I've came across this cool interactive SVG Christmas animations.
Its a Google Chrome Experiment
Please Check it out.

https://ihatetomatoes.net/svg-christmas/

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