Animations and Performance

By Paul Lewis

Paul is a Design and Perf Advocate

By Sam Thorogood

Evangelises Chrome and the mobile web in the Developer Relations team at Google.

Maintain 60fps whenever you are animating, because any less results in stutters or stalls that will be noticeable to your users and negatively impact their experiences.

TL;DR

- Take care that your animations don't cause performance issues; ensure that you know the impact of animating a given CSS property.
- Animating properties that change the geometry of the page (layout) or cause painting are particularly expensive.
- Where you can, stick to changing transforms and opacity.
- Use `will-change` to ensure that the browser knows what you plan to animate.

Animating properties is not free, and some properties are cheaper to animate than others. For example, animating the `width` and `height` of an element changes its geometry and may cause other elements on the page to move or change size. This process is called *layout* (or *reflow* in Gecko-based browsers like Firefox), and can be expensive if your page has a lot of elements. Whenever layout is triggered, the page or part of it will normally need to be painted, which is typically even more expensive than the layout operation itself.

Where you can, you should avoid animating properties that trigger layout or paint. For most modern browsers, this means limiting animations to `opacity` or `transform`, both of which the browser can highly optimize; it doesn't matter if the animation is handled by JavaScript or CSS.

For a full list of the work triggered by individual CSS properties, see CSS Triggers. You can find a full guide on creating High Performance Animations on HTML5 Rocks.

Using the will-change property

Use the `will-change` to ensure the browser knows that you intend to change an element's property. This allows the browser to put the most appropriate optimizations in place ahead of when you make the change. Don't overuse `will-change`, however, because doing so can cause the browser to waste resources, which in turn causes even more performance issues.

The general rule of thumb is that if the animation might be triggered in the next 200ms, either by a user's interaction or because of your application's state, then having `will-change` on animating elements is a good idea. For most cases, then, any element in your app's current view that you intend to animate should have `will-change` enabled for whichever properties you plan to change. In the case of the box sample we've been using throughout the previous guides, adding `will-change` for transforms and opacity looks like this:

```
.box {
  will-change: transform, opacity;
}
```

Now the browsers that support it, currently Chrome, Firefox, and Opera, will make the appropriate optimizations under the hood to support changing or animating those properties.

CSS vs JavaScript performance

There are many pages and comments threads around the web that discuss the relative merits of CSS and JavaScript animations from a performance perspective. Here are a few points to keep in mind:

- CSS-based animations, and Web Animations where supported natively, are typically handled on a thread known as the "compositor thread." This is different from the browser's "main thread", where styling, layout, painting, and JavaScript are executed. This means that if the browser is running some expensive tasks on the main thread, these animations can keep going without being interrupted.
- Other changes to transforms and opacity can, in many cases, also be handled by the compositor thread.
- If any animation triggers paint, layout, or both, the "main thread" will be required to do work. This is true for both CSS- and JavaScript-based animations, and the overhead of layout or paint will likely dwarf any work associated with CSS or JavaScript execution, rendering the question moot.

For more information about which work is triggered by animating a given property, see CSS Triggers.
