Canvas-driven background images

By Eric Bidelman
Engineer @ Google working on web tooling: Headless Chrome, Puppeteer, Lighthouse

Programmatically animating background images

There are two **primary ways people animate background images**:

1. Use CSS sprites to update a `background-position` in JS .
2. Hacks with `.toDataURL()` .

The first works great if you have the image ahead of time, but what if your source needs to be programmatically generated, say, by a `<canvas>`? The solution to #1 is to use `.toDataURL()` on the canvas and set the background to the generated URL:

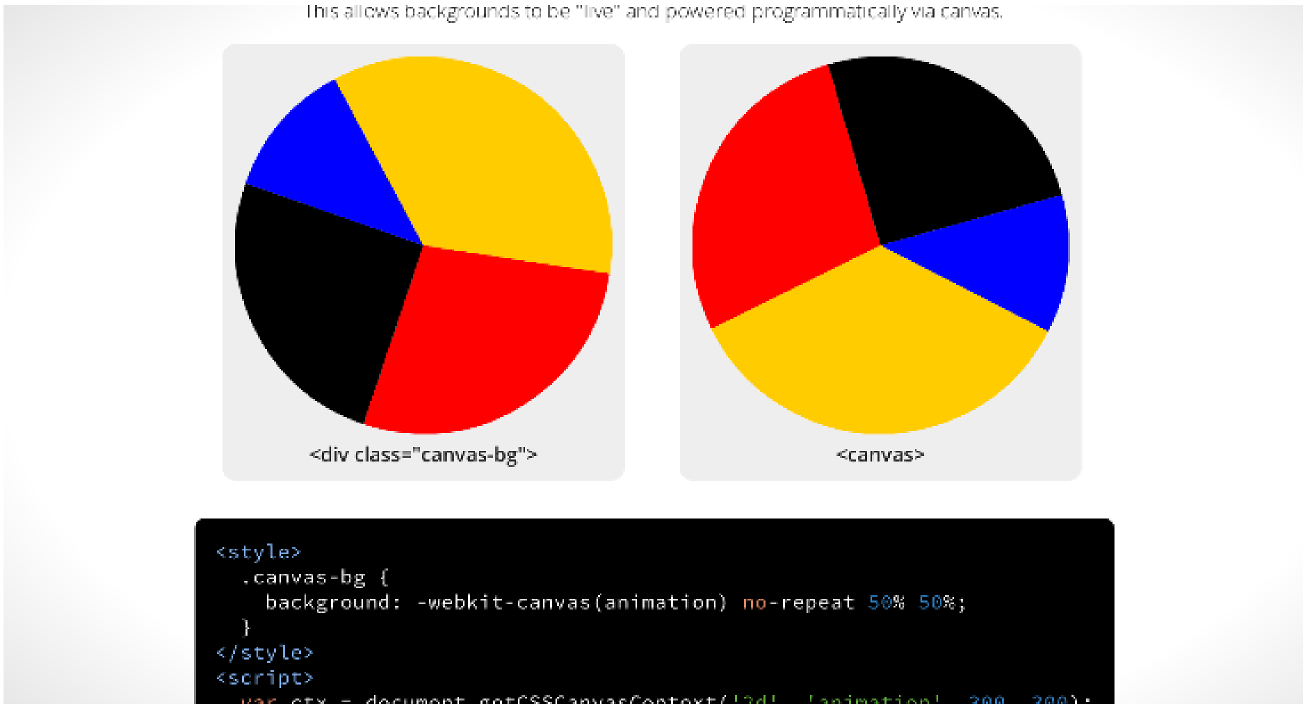
```
while(1) {
  var url = canvas.toDataURL('image/jpeg');
    el.style.background = 'url(' + url + ')';
}
```

There are two problems with this:

1. `data:` URLs add a ~33% size overhead to the resulting image.
2. A ton of DOM touching (`el.style`).

Both of these methods are inefficient: unacceptable for an always-silky-smooth 60fps web app.

Using 2d canvas as a background


This allows backgrounds to be "live" and powered programmatically via canvas.

```
<style>
  .canvas-bg {
    background: -webkit-canvas(animation) no-repeat 50% 50%;
  }
</style>
<script>
  var ctx = document.getCSSCanvasContext('2d', 'animation', 300, 300);
```

DEMO

Turns out, there's a non-standard API which WebKit has had for years that can take canvas as the source for a background. *Sadly however, there's no published spec for this feature.*

First, instead of specifying a URL for the back:

```
.bg {
  background: url(bg.png) no-repeat 50% 50%;
}
```

use `-webkit-canvas()`, referencing a string identifier to a canvas context:

```
.canvas-bg {
  background: -webkit-canvas(animation) no-repeat 50% 50%;
}
```

Next, we need to create the 2d context with a special version of `.getContext()`:

```
var ctx = document.getCSSCanvasContext('2d', 'animation', 300, 300);
```

Note: this method is on the `document` object and not the canvas. The second argument is the same name used in `-webkit-canvas()`.

Further information from Dave Hyatt:

> There is only one CSS canvas for a given global identifier per document. When you obtain the drawing context you also specify the size. The canvas will not be cleared as long as you repeatedly request the same size. Specifying a new size is equivalent to the kind of behavior you'd expect if you resized a `<canvas>` element, and so the canvas buffer will be cleared.
>
> All objects that observe a CSS canvas of the same name are sharing that canvas. This means that (similar to how animated GIFs work), you can do animations and have them happen in lockstep on all the clients of the canvas. Drawing changes will be propagated to all clients automatically.

Animations

As seen in the demo, we can reuse `requestAnimationFrame()` to drive an animation. This is great, because once things are hooked up, **the association between CSS and the canvas element is preserved**. There's no need to fiddle with the DOM.

Demo not animated in Chrome?

The current stable channel of Chrome (version 23) has crbug.com/161699, which prevents a `requestAnimationFrame()` animation from updating the background properly. This has been fixed in Chrome 25 (currently Canary). The demo also should work well in the current Safari.

Performance benefits

We're talking canvas here. **Hardware accelerated animations** are now fully in play (at least for the browsers this feature works in). And just to reiterate, **there's no need to molest the DOM** from JS.

Using webgl as a background

Hold on a sec. Does this mean we can power a CSS background using webgl? Of course it does! WebGL is merely a 3d context for canvas. Just swap in "experimental-webgl" instead of "2d", and voila.

```
var gl = document.getCSSCanvasContext('experimental-webgl', 'animation', 30
```

Here's a proof of concept that contains a div with it's background drawn using vertext and fragment shaders: DEMO

Other approaches

It's worth noting that Mozilla has had `-moz-element()` (MDN) for quite some time. This is part of the CSS Image Values and Replaced Content Module Level 4 spec and allows you to create an image generated from arbitrary HTML: videos, canvas, DOM content,...you name it. However, there are security concerns with having full access to snapshot images of the DOM. This is primarily why other browsers have not adopted said feature.