How to be Houdini and escape the limits of CSS - Harnessing the power of some cool new APIs to unlock the power of the web

Written by Seth Corker on Benevolent Bytes

Using border-radius in Firefox (left), mask-image created with CSS Paint API in Chrome (right)

What is Houdini and why should I care? The CSS-TAG Houdini Task Force is working on defining specifications that will allow the extensibility of CSS. It will give developers access to the browser's CSS rendering pipeline. This is the future of CSS. The hope is that once Houdini starts gaining some momentum and becomes usable in more browsers, web developers will have access to some lower level APIs to create new styling rules. This could be something simple using the CSS Paint API like the example I'm showing today but eventually, it will allow for changing some fundamental properties of the web which haven't changed for a long time. Not only will developers have more fine-grained control over stages of the rendering pipeline but it should be performant. So let's take a look at what you can do today.

What will Houdini give me access to? There is great potential for more complex web styles while remaining performant. It gives more low-level control over the rendering stages that are undertaken when displaying a web page and could allow for new possibilities on the web without waiting as long for browser support. It may be possible in the future to define new layouts which today can take a lot of effort using a combination of counter-intuitive positioning and frameworks.

Allowing the browser to handle CSS related tasks will mean more performant execution. Looking at the direction of increasing rendering performance, Mozilla is working on Servo which utilises the GPU for performing rendering tasks and thus freeing up the CPU for script execution.

An example using the CSS Paint API I made a simple example to show how the CSS Paint API can be used to create a squircle mask.

If you've done anything using the Canvas API, you already have a head start. To create a worklet, we create a JavaScript class called `squircle` . We can then define out paint function which takes the canvas context, the dimensions of the element and any other properties we might want to use.

Finally, we call `registerPaint()` with a name and the class we just created. This allows us to use it in CSS with `paint(squircle)` and can be used anywhere we would normally use a `url()` in CSS.

The icons have a CSS class that uses the `mask-image` property to mask the background image. Thanks to https://picsum.photos/ for the easy to use image lorem ipsum.

```
class Squircle {
  paint(ctx, geometry, properties) {
    ctx.beginPath()
    ctx.lineWidth = 1

    const multiplier = -0.01
    const xRatio = multiplier * geometry.width
    const yRatio = multiplier * geometry.height

    ctx.moveTo(0, geometry.height / 2)
    ctx.bezierCurveTo(
      0 - xRatio,
      0 - yRatio,
      0 - xRatio,
      0 - yRatio,
      geometry.width / 2,
      0
    )

    ctx.bezierCurveTo(
      geometry.width + xRatio,
      0 - yRatio,
      geometry.width + xRatio,
      0 - yRatio,
```

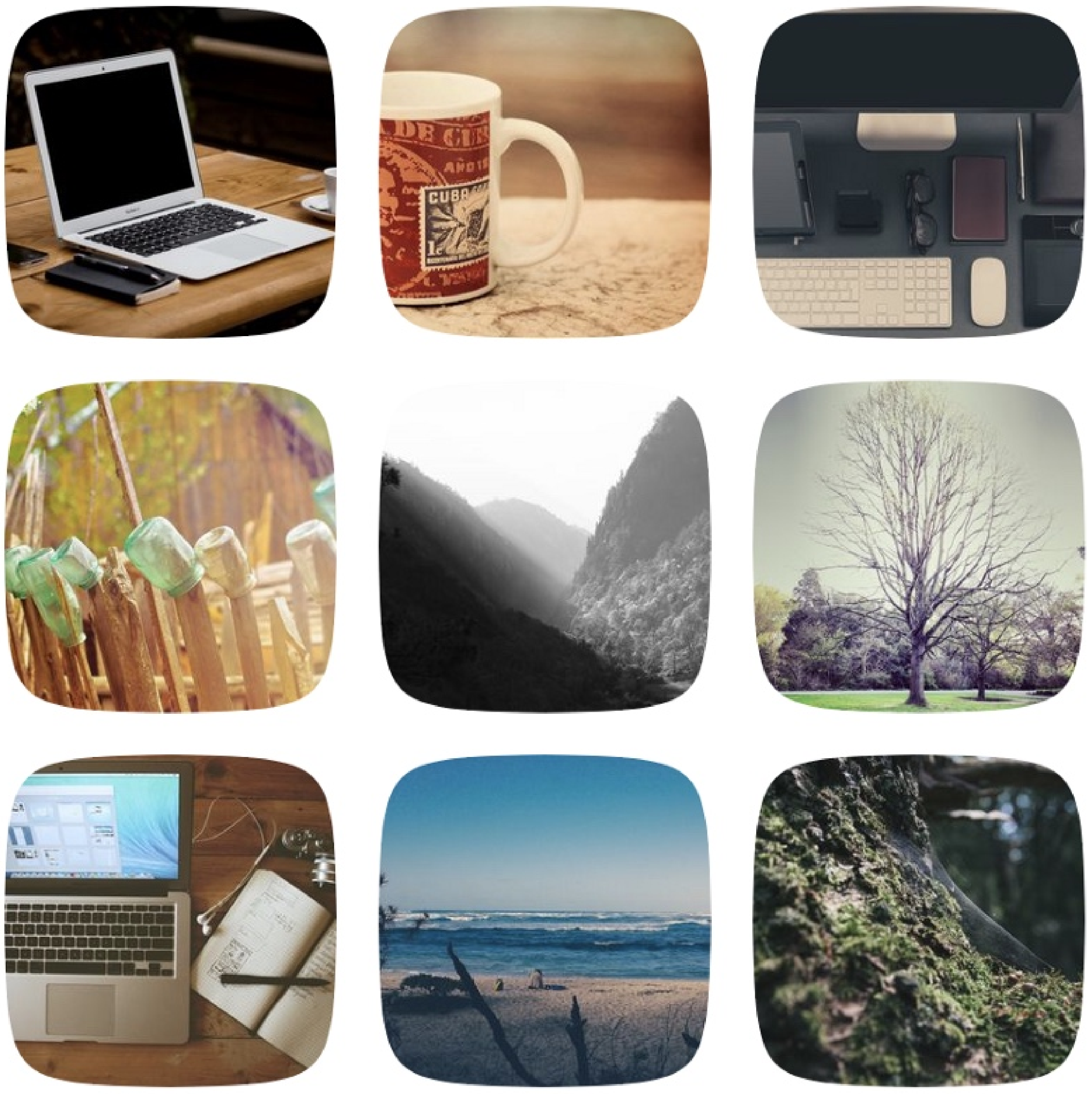
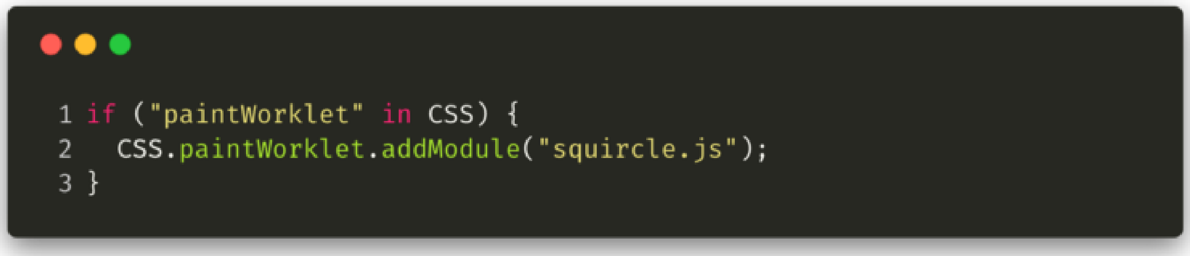


Figure 1: Grid of photos with custom rounded edges

```
      geometry.width,
      geometry.height / 2
    )

    ctx.bezierCurveTo(
      geometry.width + xRatio,
      geometry.height + yRatio,
      geometry.width + xRatio,
      geometry.height + yRatio,
      geometry.width / 2,
      geometry.height
    )

    ctx.bezierCurveTo(
      0 - xRatio,
      geometry.height + yRatio,
      0 - xRatio,
      geometry.height + yRatio,
      0,
      geometry.height / 2
    )

    ctx.fill()
  }
}

registerPaint("squircle", Squircle)
```

Checking for browser support can be done with a simple check like so:



```
if ("paintWorklet" in CSS) {
  CSS.paintWorklet.addModule("squircle.js");
}
```

Figure 2: a code snippet testing if(“paintWorklet” in CSS)

Similarly with CSS we can use the `@supports` rule to check if we can use our paint worklet or if we should just fallback to another method.

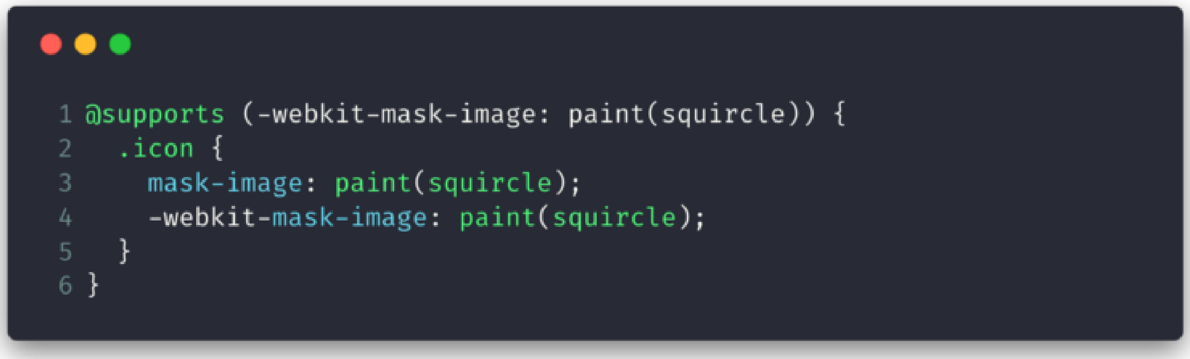
Take a look at the source code, it’s very brief and is an easy way to get started. The demo is also available, remember to use a browser that support the Paint API.

How can Houdini be used today, will it work in my browser?

If your browser is Google Chrome then probably. Houdini is very much a work in progress and thus browsers are still deciding how to approach the standards.

Browser support is almost non existent but it’s not all bad news. As of November 2018, Chrome and Opera have the best support with the CSS Paint and CSS Typed Object Model APIs. Chrome Canary is the best option for development as the Google Chrome team has been implementing draft W3C standards very quickly while the other vendors seem to be waiting for the standards to stabilize.

Apple is working on the Paint API for Safari and Mozilla have signaled intent to implement some of the APIs in Firefox.



```css
@supports (-webkit-mask-image: paint(squircle)) {
  .icon {
    mask-image: paint(squircle);
    -webkit-mask-image: paint(squircle);
  }
}
```

Figure 3: a code snippet showing feature detection using @supports in CSS

The best way to see what each browser is up to, check out https://ishoudinireadyyet.com/ by Surma. It's an easy to evaluate matrix showing major browser support and it's updated regularly.

Is Houdini Ready Yet? - *Overview over the current state of Houdini APIs in all major browsers.*

Should I use it?

I can't recommend using the APIs in production. It is very cool to tinker with, however, there is a lot of great material put out by https://developers.google.com/web/ which goes over some of the APIs more in depth. Once browser support is better, I think Houdini may be useful for improving performance of frameworks and getting into more low-level optimizations. The possibilities are really just opening up and it's difficult to predict what can be created until developers get some more time (and reason) to play with it.