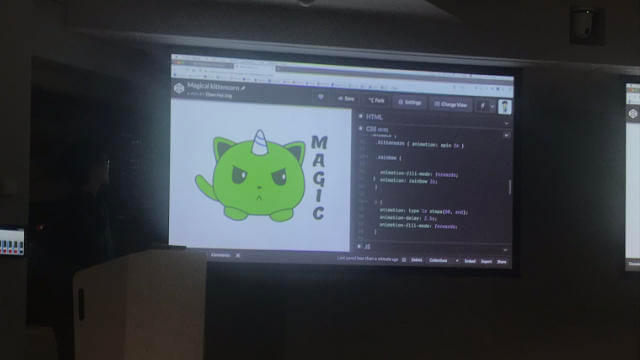
**Sub-pixel rendering and borders**

I was incredibly chuffed to have been able to speak at the inaugural [Talk.CSS](https://www.meetup.com/Melbourne-CSS/events/262802935/) in Melbourne recently, and after the event, I had a nice chat with one of the attendees about sub-pixel rendering issues for thin borders across different browsers.

Unofficial SingaporeCSS mascot making an appearance

During my talk, I had very briefly touched upon the topic of sub-pixel rendering while covering margin collapsing. If you have a completely empty box, its margins along the block direction will collapse with each other.

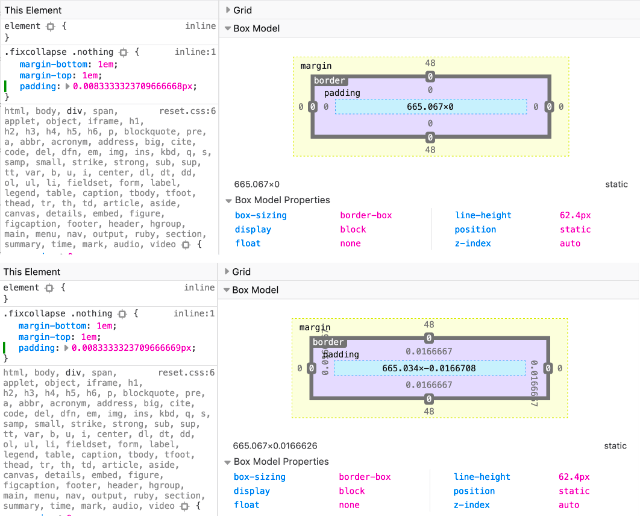
**Computed padding values**

However, adding anything at all to the empty box, even a border or some padding will prevent this. At that point, I was curious to how low a value of padding could go before the browser treats it as nothing, and I found that for Firefox, you could go down to 0.0083333323709666669px but not 0.0083333323709666668px.

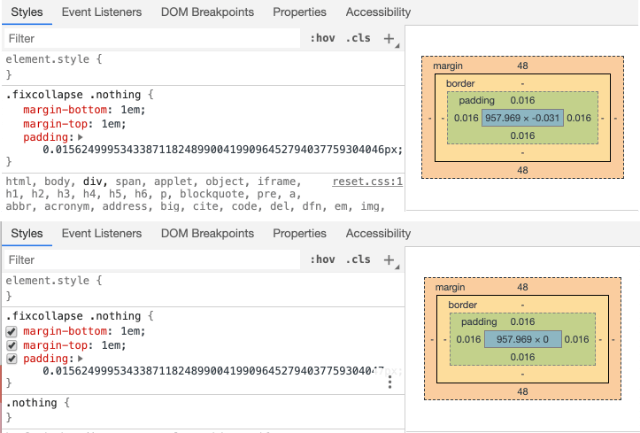
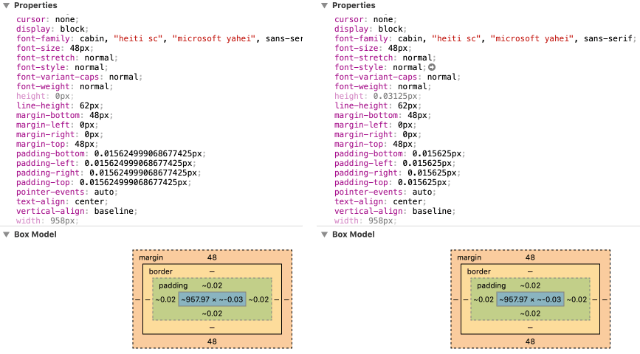
On Chrome, however, I could keep going on but I stopped at 0.015624999534338711824899004199096452794037759304047px. Feel free to go on if you want to and let me know how far you can go. Safari had the same decimal points all the way down, so I assume Blink did not change the calculation from Webkit.

Upon further inspection with DevTools for the 3 browsers I had on my laptop (who knows if browsers on Windows do something differently?), I noticed interesting things on how DevTools expressed the computed values.

Firefox’s box model diagram seems to express the padding effect correctly. Although there seems to be some magical content height going on there when padding gets rendered as a non-zero value.

Firefox DevTools' attempt at displaying excessive decimals

As for Chrome and Safari, it seems that the box model diagram doesn’t reconcile with what we see in the browser and there is also something going on with the computed values being shown.

Chrome's box model diagram seems confusedSafari's computed values seem to have some rounding somewhere

**On floating point numbers**

On a tangent, this reminded me of an issue with the display of calculation outputs consisting of floating point numbers. Floating point values are a small subset of rational numbers which have a very huge range and constant precision, which works well for computers.

[IEEE 754](https://ieeexplore.ieee.org/document/8766229), which is the technical standard for floating-point arithmetic defines arithmetic formats, interchange rules, rounding rules, operations and exception handling. I’m guessing that browsers conform to this standard when dealing with sub-pixel rounding, but I can’t be sure.

**On the rendering of borders**

Back to the discussion on borders. I use [reveal.js](https://github.com/hakimel/reveal.js/) a lot for my presentation slides and the default table style uses a 1px border between table rows. On certain resolutions, with different browsers, sometimes the border does not render.

A solution to this issue is to instead use the value of thin for the border-width property. This is a good time to talk about the CSS borders, as a refresher.

Borders have been around since [CSS1](https://www.w3.org/TR/CSS1/) as part of the box model. The main properties that define how a border looks like are border-width, border-style and border-color. All 3 properties are shorthands used to set the border properties for all 4 sides of the box.

For border-width, you could use keyword values like thin, medium or thick, as well as standard CSS length values. The thing about these keyword values is that [the specification](https://www.w3.org/TR/css-backgrounds-3/#the-border-width) does not define exactly how the browsers should resolve them.

*The lengths corresponding to thin, medium, and thick are not specified, but the values are constant throughout a document and thin ≤ medium ≤ thick. A UA could, e.g., make the thickness depend on the medium font size: one choice might be 1, 3 & 5px when the medium font size is 17px or less. Negative <length> values are not allowed.*

I hadn’t tested every single browser, just Chrome, Firefox and Safari, but all 3 of them do indeed use the suggested 1px, 3px and 5pxrespectively.

It is highly probable that the reason using thin seems to resolve the disappearing border issue is that each browser resolves the keyword in a manner that fits its own rendering engine’s calculations. So even though thin and 1px should have the same result, using thin is a better option.

**Wrapping up**

Sub-pixel rounding affects many aspects of browser rendering and it was interesting to read [John Resig’s observations](https://johnresig.com/blog/sub-pixel-problems-in-css/) on his test case involving child <div>s sized with percentage values.

Another sub-pixel rounding issue was documented by the engineers at [Symbiote](https://www.symbiote.com.au/blog/yes-even-firefox-gets-it-wrong/), who encountered the issue of rounding the calculated height of text when using em or rem values on the line-height property.

This seems like a long-standing issue with browser rendering but there doesn’t seem to be a clear solution. If anyone has any insight on the state of sub-pixel rendering, or can point me to the right people to ask, I’d greatly appreciate it. 🙏