How I Built “The Monospace Web”

Oskar Wickström — Sept 26, 2024

Recently, I published The Monospace Web, a minimalist design exploration. It all started with this innocent post, yearning for a simpler web. Perhaps too typewriter-nostalgic, but it was an interesting starting point. After some hacking and sharing early screenshots, @noteed asked for grid alignment, and down the rabbit hole I went.

This morphed into a technical challenge, while still having that creative aspect that I started with. Subsequent screenshots with the fixed grid and responsive tables sparked a lot of interest. About a week later I published the source, and since then there’s been a lot of forks. People use it for their personal web sites, mostly, but also for apps.

I’d like to explain how it works. Not everything, just focusing on the most interesting parts.

The Fixed Grid

This design aligns everything, horizontally and vertically, to a fixed grid. Like a table with equal-size cells. Every text character should be exactly contained in a cell in that grid. Borders and other visual elements may span cells more freely.

The big idea here is to use the unit in CSS. I actually did not know about it before this experiment. The unit is described in CSS Values and Units Module Level 4:

> *Represents the typical advance measure of European alphanumeric characters, and measured as the used advance measure of the “0” (ZERO, U+0030) glyph in the font used to render it.*

Further, it notes:

> *This measurement is an approximation (and in monospace fonts, an exact measure) of a single narrow glyph’s advance measure, thus allowing measurements based on an expected glyph count.*

Fantastic! A cell is thus wide. And the cell height is equal to the . In order to refer to the line height in calculations, it’s extracted as a CSS variable:

So far there’s no *actual* grid on the page. This is just a way of measuring and sizing elements based on the width and height of monospace characters. Every element must take up space evenly divisible by the dimensions of a cell; if it’s top left corner starts at a cell, then its bottom right must do so as well. That means that all elements, given that their individual sizes respect the grid, line up properly.

The Font

I’ve chosen JetBrains Mono for the font. It looks great, sure, but there’s a more important reason for this particular choice: it has good support for box-drawing characters at greater line heights. Most monospace fonts I tried broke at line heights above 110% or so. Lines and blocks were chopped up vertically. With JetBrains Mono, I can set it to 120% before it starts to become choppy.

I suspect Pragmata Pro or Berkeley Mono might work in this regard, but I haven’t tried them yet.

Also, if you want to use this design and with a greater line height, you can probably do so if you don’t need box-drawing characters. Then you may also consider pretty much any monospace font. Why not use web-safe ones, trimming down the page weight by about 600kB!

To avoid alternate glyphs for numbers, keeping everything aligned to the grid, I set:

And, for a unified thickness of borders, text, and underlines:

This gives the design that thick and sturdy feel.

The Body

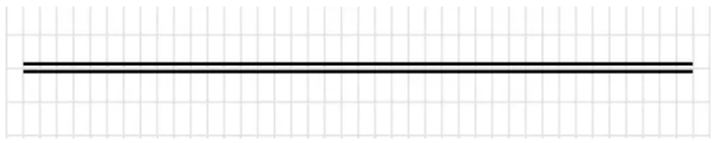
The element is the main container in the page. It is *at most* 80 characters wide. (Huh, I wonder where that number came from?)

Now for one of the key tricks! I wanted this design to be reasonably responsive. For a viewport width smaller than (and of padding), the body width needs to be some smaller width that is still evenly divisible by the cell dimensions. This can be accomplished with CSS rounding:

This way, the body shrinks in steps according to the grid.

The Horizontal Rule

Surprisingly, the custom horizontal rule styling was a fiddly enterprise. I wanted it to feel heavy, with double lines. The lines are vertically centered around the break between two cells:



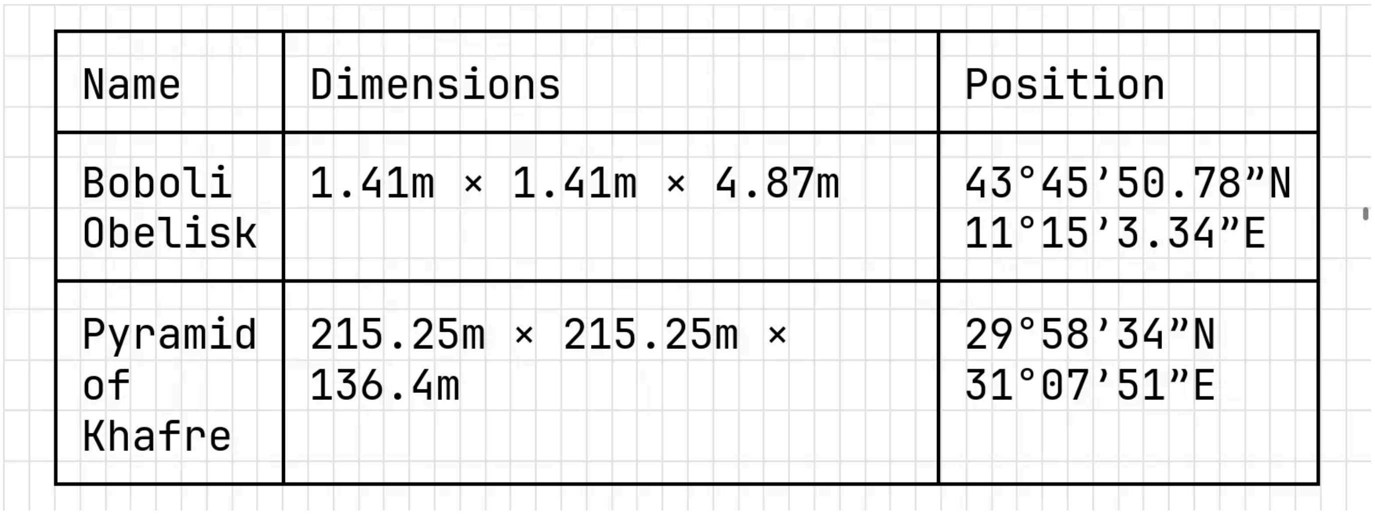
The horizontal ruler

To respect the grid, the top and bottom spacing needs to be calculated. But padding won’t work, and margin interacts with adjacent elements’ margins, so this required two elements:

The itself is just a container that takes up space; 4 lines in total. The pseudo-element draws the two lines, using , at the vertical center of the .

The Table

Table styling was probably the trickiest. Recalling the principles from the beginning, every character must be perfectly aligned with the grid. But I wanted vertical padding of table cells to be *half* the line height. A full line height worth of padding is way too airy.



Name	Dimensions	Position
Boboli Obelisk	1.41m × 1.41m × 4.87m	43°45’50.78”N 11°15’3.34”E
Pyramid of Khafre	215.25m × 215.25m × 136.4m	29°58’34”N 31°07’51”E

This requires the table being horizontally offset by half the width of a character, and vertically offset by half the line height.

Cell padding is calculated based on cell size and borders, to keep grid alignment:

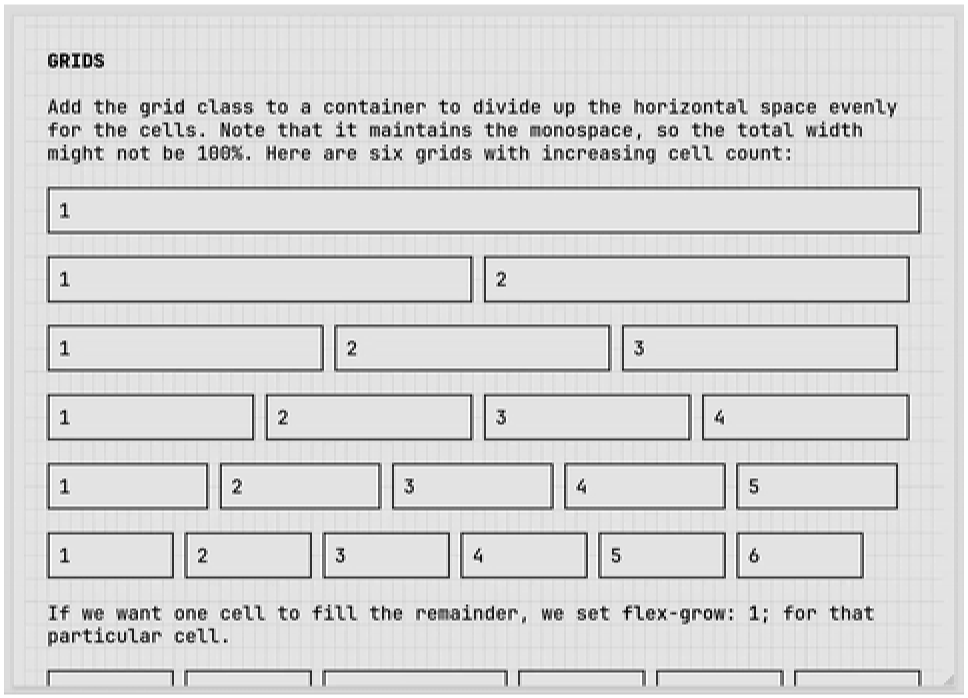
Finally, the first row must have slightly less vertical padding to compensate for the top border. This is hacky, and would be nicer to solve with some kind of negative margin on the table. But then I’d be back in margin interaction land, and I don’t like it there.

Another quirk is that columns need to have set sizes. All but one should use the class, and the remaining should use . Otherwise, cells divide the available width in a way that doesn’t align with the grid.

The Layout Grid

I also included a class for showcasing how a grid layout helper could work. Much like the 12-column systems found in CSS frameworks, but funkier. To use it, you simply slap on a class on a container. It uses a glorious hack to count the children in pure CSS:

Look at it go!


GRIDS

Add the grid class to a container to divide up the horizontal space evenly for the cells. Note that it maintains the monospace, so the total width might not be 100%. Here are six grids with increasing cell count:

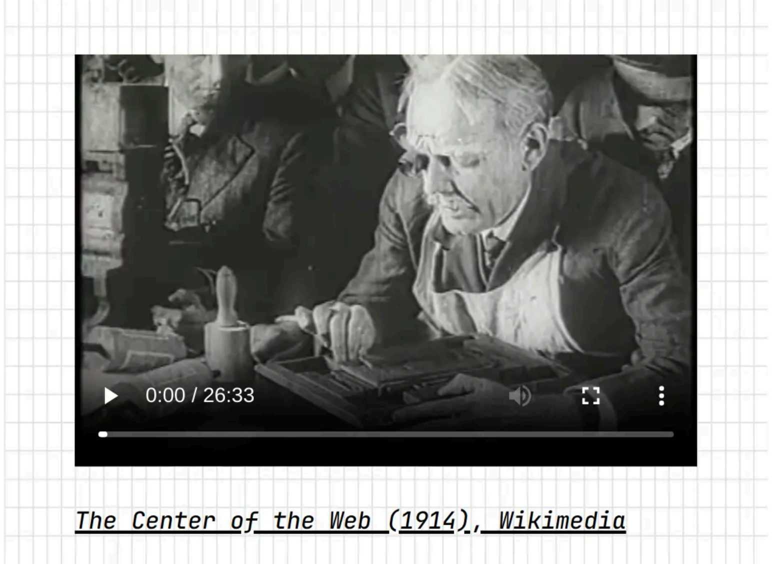
If we want one cell to fill the remainder, we set flex-grow: 1; for that particular cell.

The responsive layout grid in action

Unlike with tables, the layout grid rows don't have to fill the width. Depending on your viewport width, you'll see a ragged right margin. However, by setting on one of the children, that one grows to fill up the remaining width.

The Media Elements

Images and video grow to fill the width. But they have their own proportions, making vertical alignment a problem. How many multiples of the line height should the height of the media be? I couldn't figure out a way to calculate this with CSS alone.



The Center of the Web (1914), Wikimedia

Media element with adjusted bottom padding (the figcaption below lines up with the grid)

One option was a preprocessor step that would calculate and set the ratio of every such element as a CSS variable, and then have CSS calculate a based on the ratio and the width:

However, I eventually settled for small chunk of JavaScript to calculate the difference, and set an appropriate . Ending up in JavaScript was inevitable, I suppose.

Summary

There are many small things I haven't shown in detail, including:

- the debug grid overlay, which you see in the screenshots
- ordered list numbering
- the tree-rendered list
- various form controls and buttons
- the custom element

But I think I've covered the most significant bits. For a full tour, have a look at the source code.

There are still bugs, like alignment not working the same in all browsers, and not working at all in some cases. I'm not sure why yet, and I might try to fix it in at least Firefox and Chromium. Those are the ones I can test with easily.

This has been a fun project, and I’ve learned a bunch of new CSS tricks. Also, the amount of positive feedback has been overwhelming. Of course, there’s been some negative feedback as well, not to be dismissed. I do agree with the concern about legibility. Monospace fonts can be beautiful and very useful, but they’re probably not the best for prose or otherwise long bodies of text.

Some have asked for reusable themes or some form of published artifact. I won’t spend time maintaining such packages, but I encourage those who would. Fork it, tweak it, build new cool things with it!

Finally, I’ll note that I’m happy with how the overall feel of this design turned out, even setting aside the monospace aspect. Maybe it would work with a proportional, or perhaps semi-proportional, font as well?