The algorithm for a perfectly balanced photo gallery

Remember the days in college when you learned all about the big Oh!'s and re-implemented all these sort-algorithms for the hundredth time? If you are a web developer like me, chances are you never had to touch a single one of these algorithms ever again.

I mean, hands down, 99% of the hard work in web development is working around browser quirks and messing with other people's code.

And then, all the sudden, that algorithm knowledge actually becomes very useful.

The idea

Ever since we started working on Chromatic we knew we wanted the photos to be as big as possible. No tiny thumbnails, no square-cropping, no wasting of precious screen real estate.

Galleries typically have between 10 and 100 photos with various aspect ratios (that's width divided by height) and we want them to be equally distributed over the rows, taking up all the space available.

Here's the kicker: While it looks like all the photos are the same height, each row is scaled slightly as a whole to fit in the horizontal space.

The problem

But how do we know how to distribute them? As soon as you leave the well-trodden paths of how-do-I-get-library-x-to-do-y it gets much harder to elicit an answer from StackOverflow.

Then we started thinking of our problem in an algorithmic sense:

> *Might this be a well-known standard problem?*

> *Or can it be turned into one?*

If so, chances are high someone already came up with an algorithm to solve it.

Turns out it actually *was*: The partition problem. And even better, there was a working Python implementation which we ported to Coffeescript.

The solution

Once we got that down, the rest was fairly easy.

To find the number of rows (k) needed, we first scale the photos to half the window's height (we found the photos look best when scaled to roughly half the window's height), sum up their widths, then divide by the window's width. That's probably not a whole number, so we round it.

The photos' aspect ratios (multiplied by 100 and rounded) serve as a set (S) of weights. We now use our newly gained algorithmic superpowers to find the perfect distribution of set S over k.

The result tells us the number of photos in each row. All we have left to do is fill the row with the appropriate range of photos and scale it to fit in the horizontal space available.

Here's the relevant code from the Backbone gallery view. The actual linear partition algorithm can be found here.

```coffeescript
viewport_width = $(window).width()
ideal_height = parseInt($(window).height() / 2)
summed_width = photos.reduce ((sum, p) -> sum += p.get('aspe
rows = Math.round(summed_width / viewport_width)

if rows < 1
  # (2a) Fallback to just standard size
  photos.each (photo) -> photo.view.resize parseInt(ideal_he
else
  # (2b) Distribute photos over rows using the aspect ratio
  weights = photos.map (p) -> parseInt(p.get('aspect_ratio')
  partition = linear_partition(weights, rows)

  # (3) Iterate through partition
  index = 0
  row_buffer = new Backbone.Collection
  _.each partition, (row) ->
    row_buffer.reset()
    _.each row, -> row_buffer.add(photos.at(index++))
    summed_ratios = row_buffer.reduce ((sum, p) -> sum += p
    row_buffer.each (photo) -> photo.view.resize parseInt(vi
```

chromatic_gallery_view.coffee hosted with ❤ by **GitHub** **view raw**

It’s taken a considerable effort, but hey, look at the outcome!

If you liked this article, you should follow me on Twitter.