Quick Note: Using Viewport Units

These little darling units allow for some really nice effects and typographical work. We'll explore what they are and a couple effects we can use them on.

vh and vw

The two basic units for viewport measurement are vh and vw which track like so:

- vw: 1/100th (1%) viewport width
- vh: 1/100th (1%) viewport height

The units will remain constant. 1/100th of the viewport width will remain 1/100th regardless of how wide or narrow the viewport becomes. to make an element span half the screen horizontally we could do something like this:

```
hero {
  width: 50vw;
}
```

There may also be elements like a masthead that we want to place in the top 20% of the screen we could combined with `position: fixed` to do something like this:

```
.masthead {
  height: 20vh;
  position: fixed;
}
```

vmin and vmax

These two units are a little more complicated, they will take the value from whatever side matches the minimum or maximum value.

- `vmin`: 1/100th (1%) of the smallest side
- `vmax`: 1/100th (1%) of the largest side

I've always had a hard time when it comes to choosing how/when to use `vmin` and `vmax`. The default value for these will also depend on the device's orientation and how much has the user resized the browser window:

`vmin` uses the ratio of the smallest side. If the height of the browser window is less than its width, 1`vmin` will be equivalent to 1`vh`. Otherwise 1`vmin` is 1`vw`.

`vmax` is the opposite: 1`vmax` equals 1`vw` if the viewport is wider than it is tall; If not 1`vmax` is equal to 1`vh`.

This means we can use these values inside device orientation media queries (`(@media screen and (orientation : portrait)` or `@media screen and (orientation : landscape)`) since these units are already designed to take the value from either the height or the width.

Some examples

Here are some examples of what you can do with these units.

Trully responsive content

A couple years ago I came across a Pen from Brian Haferkamp called 2-Up Magazine Layout and I was amazed at the level of complexity he crammed into the Pen and the attention to detail that he paid to little details such as responsive sizing of the content and how the 2-column layout would automatically resize itself in narrower viewports.

I took the idea, having a fixed image next to scrolling text. I think I've finally figure out what the technique is and how to implement it. This is a stripped down version of my pen.

The first thing we do is set the wrapper container to be a flex container using `display: flex`. We don't need to set the children items as `flex-item`. This is automatically done when we set the parent element's display.

background-size: cover tells the browser to cover the full dimension of the container, in this case 100vh and 50vw.

QWe can play with the image positioning using background-position to place different parts of the image in the viewpoert we're covering.

The flex property is the shorthand for flex-grow, flex-shrink and flex-basis combined. With these values we're telling `.poster-image` elements to take 50% of the parent container size before other elements are distributed in it.

For the `.text` element(s) we give them padding and a left margin. We also make sure that they are at the top of the stacking order.

```
.article-wrapper {
  display: flex;
}

.poster-image {
  position: fixed;
  background-image: url("path/to/image");
  background-size: cover;
  // background-position: center;
  height: 100vh;
  width: 50vw;
  flex: 0 0 50%;
  z-index: 1;
}

.text {
  padding: 100px 60px 60px;
  margin-left: 50%;
  z-index: 10;
}
```

The HTML is a series of div containers that hold:

- The article wrapper element (`.article-wrapper`)
- The poster image (`.poster-image`)
- The text of the article (`.text`)
- And the article content itself (`article`)

```
<div class="article-wrapper">
  <div class="poster-image"></div>
  <div class="text">
    <h1>Article Title</h1>
    <article>
      <p></p>
    </article>
  </div>
</div>
```

Some considerations

There are a couple things I haven't quite figure out to make this work reliably for all browsers.

Scrollbars

Browsers have inconsistent behaviors regarding scrollbars across operating systems. A very basic way to address the issue is to script the behavior of scrollbars: If the height of an element's content, including content not visible on the screen due to overflow (in the read-only property `element.scrollHeight` of the document element) is taller than the inner height of the document element in pixels, including padding but not the horizontal scrollbar height, border, or margin (in the read-only `clientHeight`property of the document element) the we force scrollbars, otherwise we hide them.

```
const element = document.documentElement;

if(element.scrollHeight > element.clientHeight) {
  // Overflow detected; force scroll bar
  element.style.overflow = 'scrollbar';
} else {
  'scrollbar'// No overflow detected; prevent scroll bar
  element.style.overflow = 'hidden';
}
```

Yes, this is a simplistic solution. If you want more details and a much more

thorough solution to the problem, check Crossbrowser JavaScript Scrollbar Detection by Tyler Cipriani

Legibility on smaller screens

If we set the font to a viewport unit we need to be very careful that the text is readable in smaller screens. The best and most reliable way to do this is to create media queries for the smaller form factors and set the font to an absolute value or em or rem to make sure the text is still readable.

Links and Resources

- Using vw and vh Measurements In Modern Site Design
- MinMaxing: Understanding vMin and vMax in CSS
- Viewport units: vw, vh, vmin, vmax