SMASHING MAGAZINE

Responsive Design

C

Imprint

Version 1: June 2012

ISBN: 978-3-943075-33-5

Cover Design: Ricardo Gimenes

PR & Press: Stephan Poppe

eBook Strategy: Thomas Burkert

Technical Editing: Talita Telma Stöckle, Andrew Rogerson

Idea & Concept: Smashing Media GmbH

ABOUT SMASHING MAGAZINE

Smashing Magazine is an online magazine dedicated to Web designers and developers worldwide. Its rigorous quality control and thorough editorial work has gathered a devoted community exceeding half a million subscribers, followers and fans. Each and every published article is carefully prepared, edited, reviewed and curated according to the high quality standards set in Smashing Magazine's own publishing policy. Smashing Magazine publishes articles on a daily basis with topics ranging from business, visual design, typography, front-end as well as back-end development, all the way to usability and user experience design. The magazine is — and always has been — a professional and independent online publication neither controlled nor influenced by any third parties, delivering content in the best interest of its readers. These guidelines are continually revised and updated to assure that the quality of the published content is never compromised.

ABOUT SMASHING MEDIA GMBH

Smashing Media GmbH is one of the world's leading online publishing companies in the field of Web design. Founded in 2009 by Sven Lennartz and Vitaly Friedman, the company's headquarters is situated in southern Germany, in the sunny city of Freiburg im Breisgau. Smashing Media's lead publication, Smashing Magazine, has gained worldwide attention since its emergence back in 2006, and is supported by the vast, global Smashing community and readership. Smashing Magazine had proven to be a trustworthy online source containing high quality articles on progressive design and coding techniques as well as recent developments in the Web design industry.

About this eBook

This eBook, "Responsive Design", gives an overview about responsive Web design, showing many situations and techniques in which this approach can be applied to. Smartphones, tablets, laptops and desktop computers may share the same design, which is adapted according to screen size, platform and orientation of each device. Flexible grids and layouts, images, text and an intelligent use of CSS media queries are included.

Table of Contents

Responsive Web Design: What It Is And How To Use It

Kayla Knight

Almost every new client these days wants a mobile version of their website. It's practically essential after all: one design for the BlackBerry, another for the iPhone, the iPad, netbook, Kindle — and all screen resolutions must be compatible, too. In the next five years, we'll likely need to design for a number of additional inventions. When will the madness stop? It won't, of course.

In the field of Web design and development, we're quickly getting to the point of being unable to keep up with the endless new resolutions and devices. For many websites, creating a website version for each resolution and new device would be impossible, or at least impractical. Should we just suffer the consequences of losing visitors from one device, for the benefit of gaining visitors from another? Or is there another option?

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. In other words, the website should have the technology to automatically *respond* to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market.

The Concept Of Responsive Web Design

Ethan Marcotte wrote an introductory article about the approach, "Responsive Web Design," for A List Apart. It stems from the notion of responsive architectural design, whereby a room or space automatically adjusts to the number and flow of people within it:

> *"Recently, an emergent discipline called "responsive architecture" has begun asking how physical spaces can respond to the presence of people passing through them. Through a combination of embedded robotics and tensile materials, architects are experimenting with art installations and wall structures that bend, flex, and expand as crowds approach them. Motion sensors can be paired with climate control systems to adjust a room's temperature and ambient lighting as it fills with people. Companies have already produced "smart glass technology" that can automatically become opaque when a room's occupants reach a certain density threshold, giving them an additional layer of privacy."*

Transplant this discipline onto Web design, and we have a similar yet whole new idea. Why should we create a custom Web design for each group of users; after all, architects don't design a building for each group size and type that passes through it? Like responsive architecture, Web design should automatically adjust. It shouldn't require countless custom-made solutions for each new category of users.

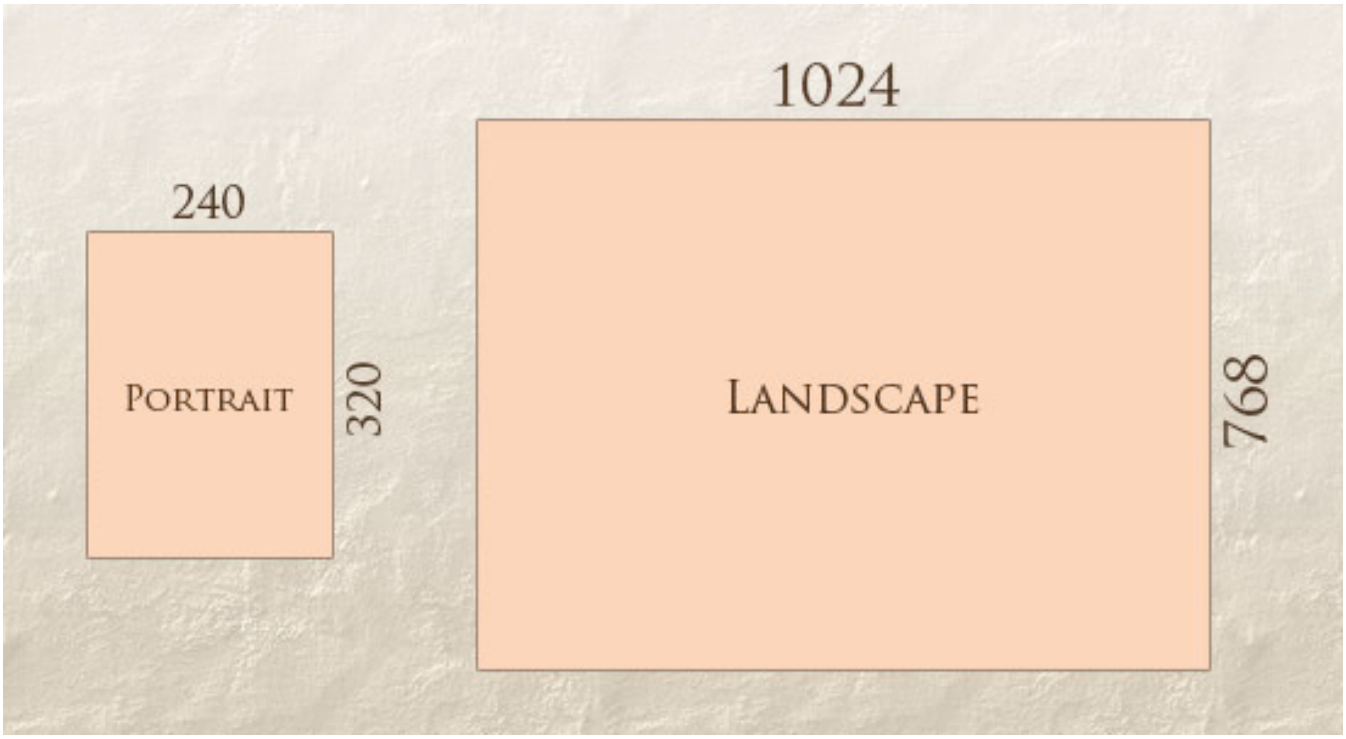
Obviously, we can't use motion sensors and robotics to accomplish this the way a building would. Responsive Web design requires a more abstract way of thinking. However, some ideas are already being practiced: fluid layouts,

media queries and scripts that can reformat Web pages and mark-up effortlessly (or *automatically*).

But responsive Web design is not only about adjustable screen resolutions and automatically resizable images, but rather about a whole new way of thinking about design. Let's talk about all of these features, plus additional ideas in the making.

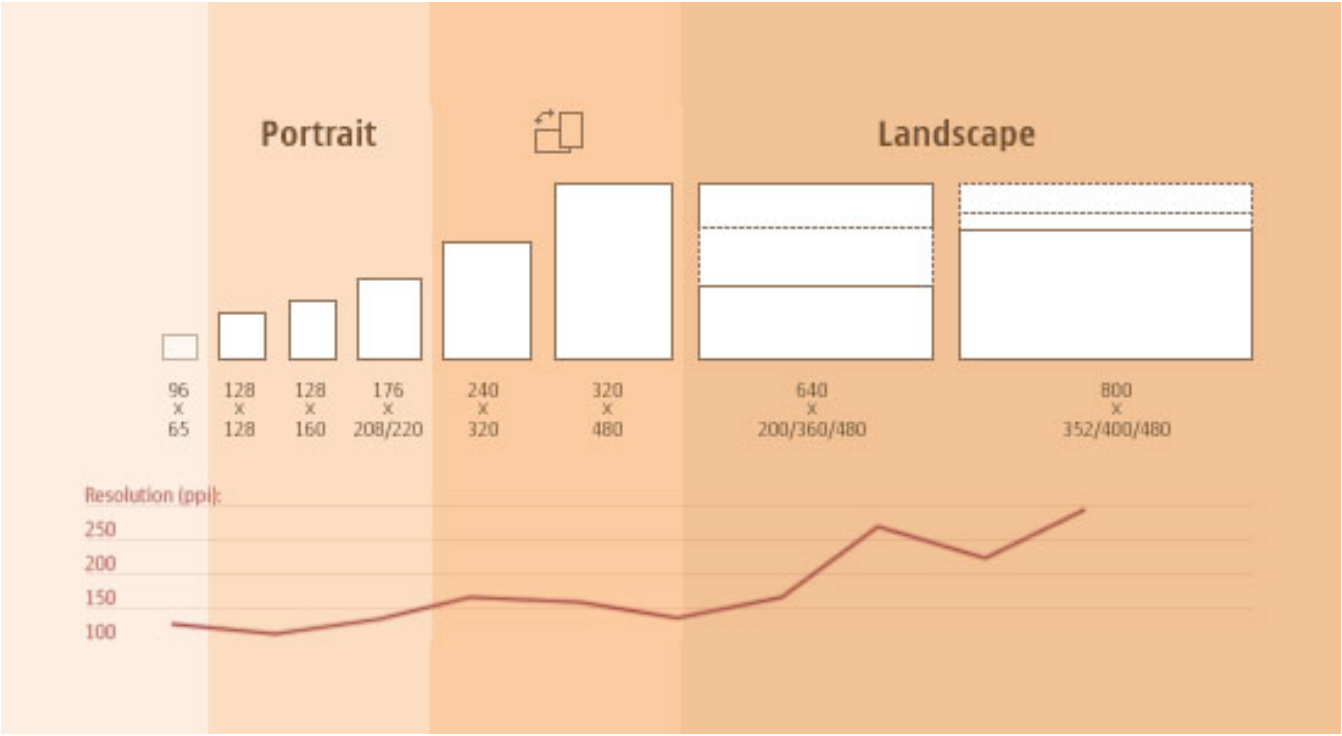
Adjusting Screen Resolution

With more devices come varying screen resolutions, definitions and orientations. New devices with new screen sizes are being developed every day, and each of these devices may be able to handle variations in size, functionality and even color. Some are in landscape, others in portrait, still others even completely square. As we know from the rising popularity of the iPhone, iPad and advanced smartphones, many new devices are able to switch from portrait to landscape at the user's whim. How is one to design for these situations?



In addition to designing for both landscape and portrait (and enabling those orientations to possibly switch in an instant upon page load), we must consider the hundreds of different screen sizes. Yes, it is possible to group them into major categories, design for each of them, and make each design as flexible as necessary. But that can be overwhelming, and who knows what the usage figures will be in five years? Besides, many users do not maximize their browsers, which itself leaves far too much room for variety among screen sizes.

Morten Hjerde and a few of his colleagues identified statistics on about 400 devices sold between 2005 and 2008. Below are some of the most common:



Since then even more devices have come out. It's obvious that we can't keep creating custom solutions for each one. So, how do we deal with the situation?

PART OF THE SOLUTION: FLEXIBLE EVERYTHING

A few years ago, when flexible layouts were almost a "luxury" for websites, the only things that were flexible in a design were the layout columns (structural elements) and the text. Images could easily break layouts, and even flexible structural elements broke a layout's form when pushed enough. Flexible designs weren't really that flexible; they could give or take a few hundred pixels, but they often couldn't adjust from a large computer screen to a netbook.

Now we can make things more flexible. Images can be automatically adjusted, and we have workarounds so that layouts never break (although they may become squished and illegible in the process). While it's not a complete fix, the solution gives us far more options. It's perfect for devices that switch from portrait orientation to landscape in an instant or for when users switch from a large computer screen to an iPad.

In Ethan Marcotte's article, he created a sample Web design that features this better flexible layout:

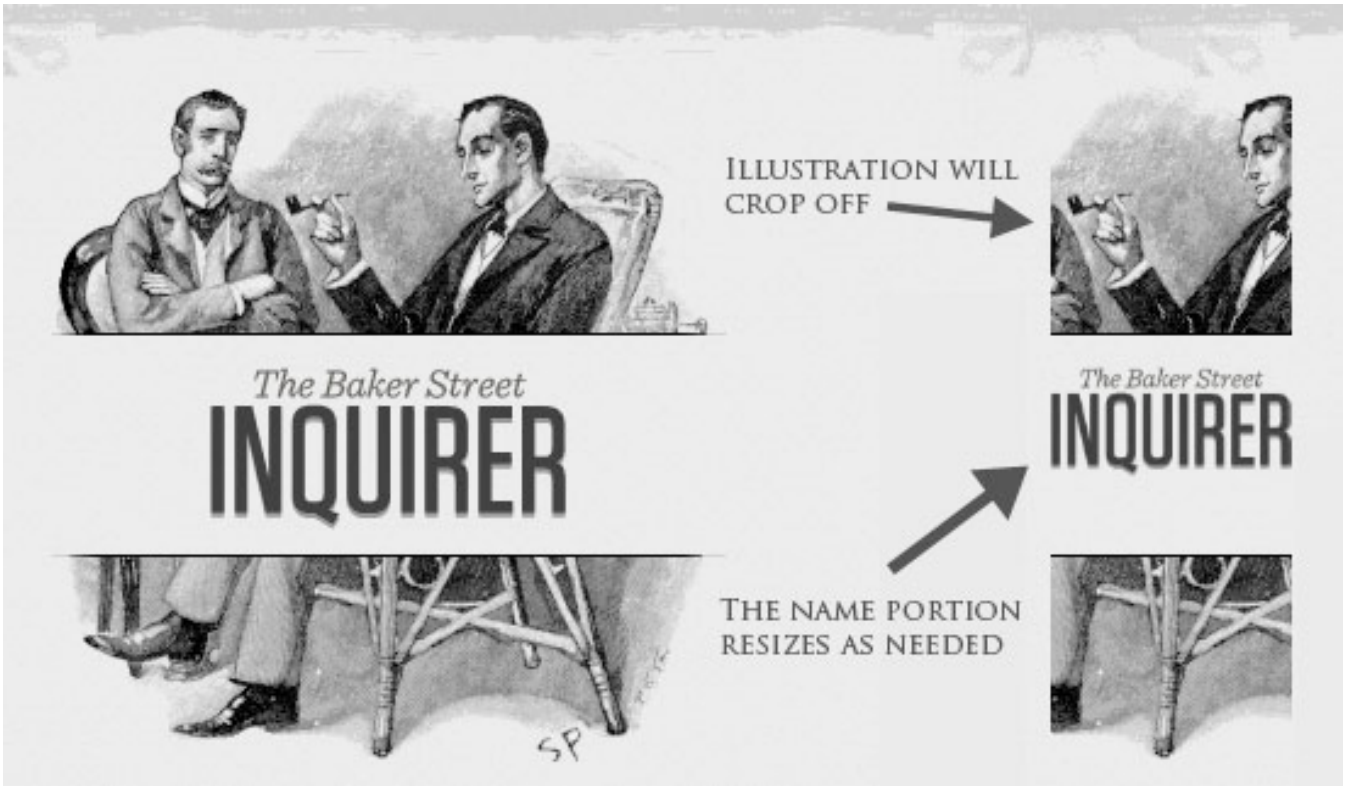
The entire design is a lovely mix of fluid grids, fluid images and smart mark-up where needed. Creating fluid grids is fairly common practice, and there are a number of techniques for creating fluid images:

- Hiding and Revealing Portions of Images
- Creating Sliding Composite Images

- Foreground Images That Scale With the Layout

For more information on creating fluid websites, be sure to look at the book “Flexible Web Design: Creating Liquid and Elastic Layouts with CSS” by Zoe Mickley Gillenwater, and download the sample chapter “Creating Flexible Images.” In addition, Zoe provides the following extensive list of tutorials, resources, inspiration and best practices on creating flexible grids and layouts: “Essential Resources for Creating Liquid and Elastic Layouts.”

While from a technical perspective this is all easily possible, it’s not just about plugging these features in and being done. Look at the logo in this design, for example:



If resized too small, the image would appear to be of low quality, but keeping the name of the website visible and not cropping it off was important. So, the image is divided into two: one (of the illustration) set as a background, to be cropped and to maintain its size, and the other (of the name) resized proportionally.

```
<h1 id="logo"><a href="#"><img src="site/logo.png" alt="The
Baker Street Inquirer" /></a></h1>
```

Above, the `h1` element holds the illustration as a background, and the image is aligned according to the container's background (the heading).

This is just one example of the kind of thinking that makes responsive Web design truly effective. But even with smart fixes like this, a layout can become too narrow or short to look right. In the logo example above (although it works), the ideal situation would be to not crop half of the illustration or to keep the logo from being so small that it becomes illegible and "floats" up.

Flexible Images

One major problem that needs to be solved with responsive Web design is working with images. There are a number of techniques to resize images proportionately, and many are easily done. The most popular option, noted in Ethan Marcotte's article on fluid images but first experimented with by Richard Rutter, is to use CSS's **`max-width`** for an easy fix.

As long as no other width-based image styles override this rule, every image will load in its original size, unless the viewing area becomes narrower than the image's original width. The maximum width of the image is set to 100% of the screen or browser width, so when that 100% becomes narrower, so does the image. Essentially, as Jason Grigsby noted, "The idea behind fluid images is that you deliver images at the maximum size they will be used at. You don't declare the height and width in your code, but instead let the browser resize the images as needed while using CSS to guide their relative size". It's a great and simple technique to resize images beautifully.

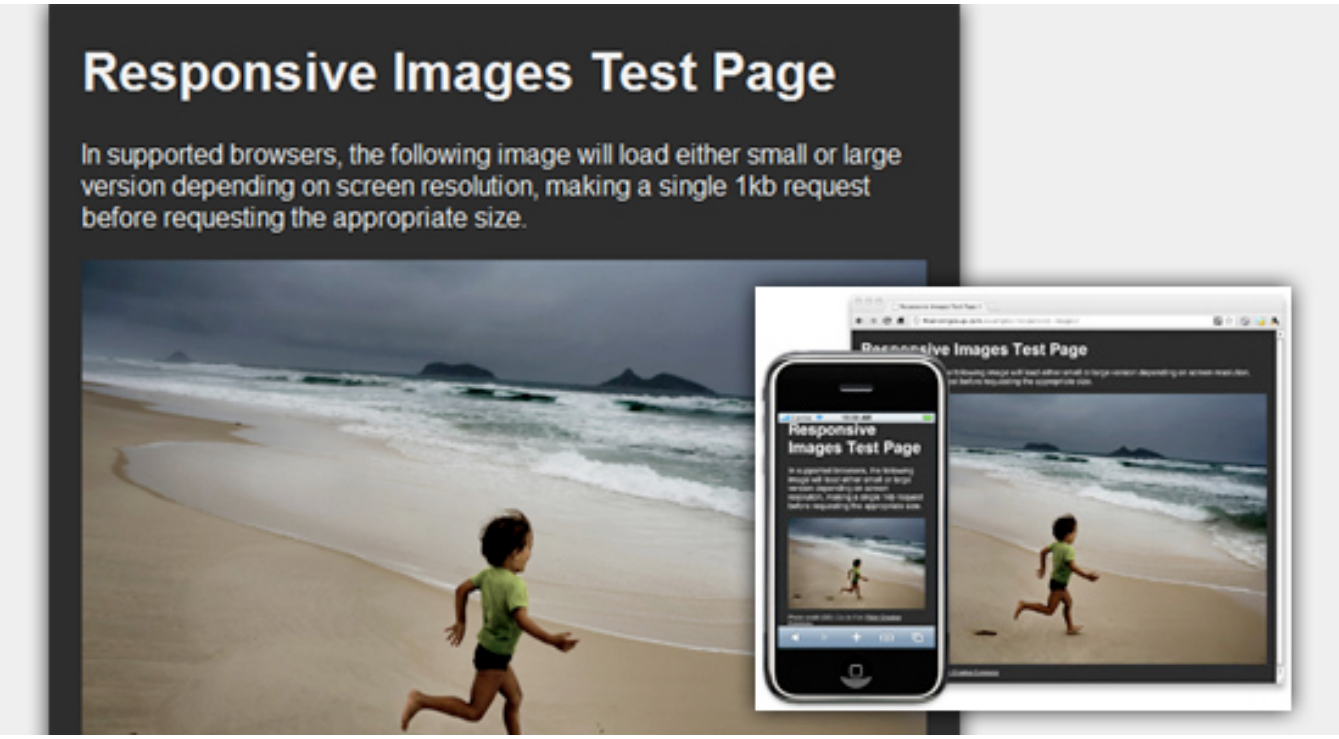
Note that **`max-width`** is not supported in IE, but a good use of **`width: 100%`** would solve the problem neatly in an IE-specific style sheet. One more issue is that when an image is resized too small in some older browsers in Windows, the rendering isn't as clear as it ought to be. There is a JavaScript to fix this issue, though, found in Ethan Marcotte's article.

While the above is a great quick fix and good start to responsive images, image resolution and download times should be the primary considerations. While resizing an image for mobile devices can be very simple, if the original image size is meant for large devices, it could significantly slow download times and take up space unnecessarily.

FILAMENT GROUP'S RESPONSIVE IMAGES

This technique, presented by the Filament Group, takes this issue into consideration and not only resizes images proportionately, but shrinks image resolution on smaller devices, so very large images don't waste space unnecessarily on small screens.

```
<img src="smallRes.jpg" data-fullsrc="largeRes.jpg">
```



This technique requires a few files, all of which are available on Github. First, a JavaScript file (*rwd-images.js*), the *.htaccess* file and an image file (*rwd.gif*). Then, we can use just a bit of HTML to reference both the larger and smaller resolution images: first, the small image, with an *.r* prefix to clarify that it should be responsive, and then a reference to the bigger image using **`data-fullsrc`**.

The **`data-fullsrc`** is a custom HTML5 attribute, defined in the files linked to above. For any screen that is wider than 480 pixels, the larger-resolution

image (*largeRes.jpg*) will load; smaller screens wouldn't need to load the bigger image, and so the smaller image (*smallRes.jpg*) will load.

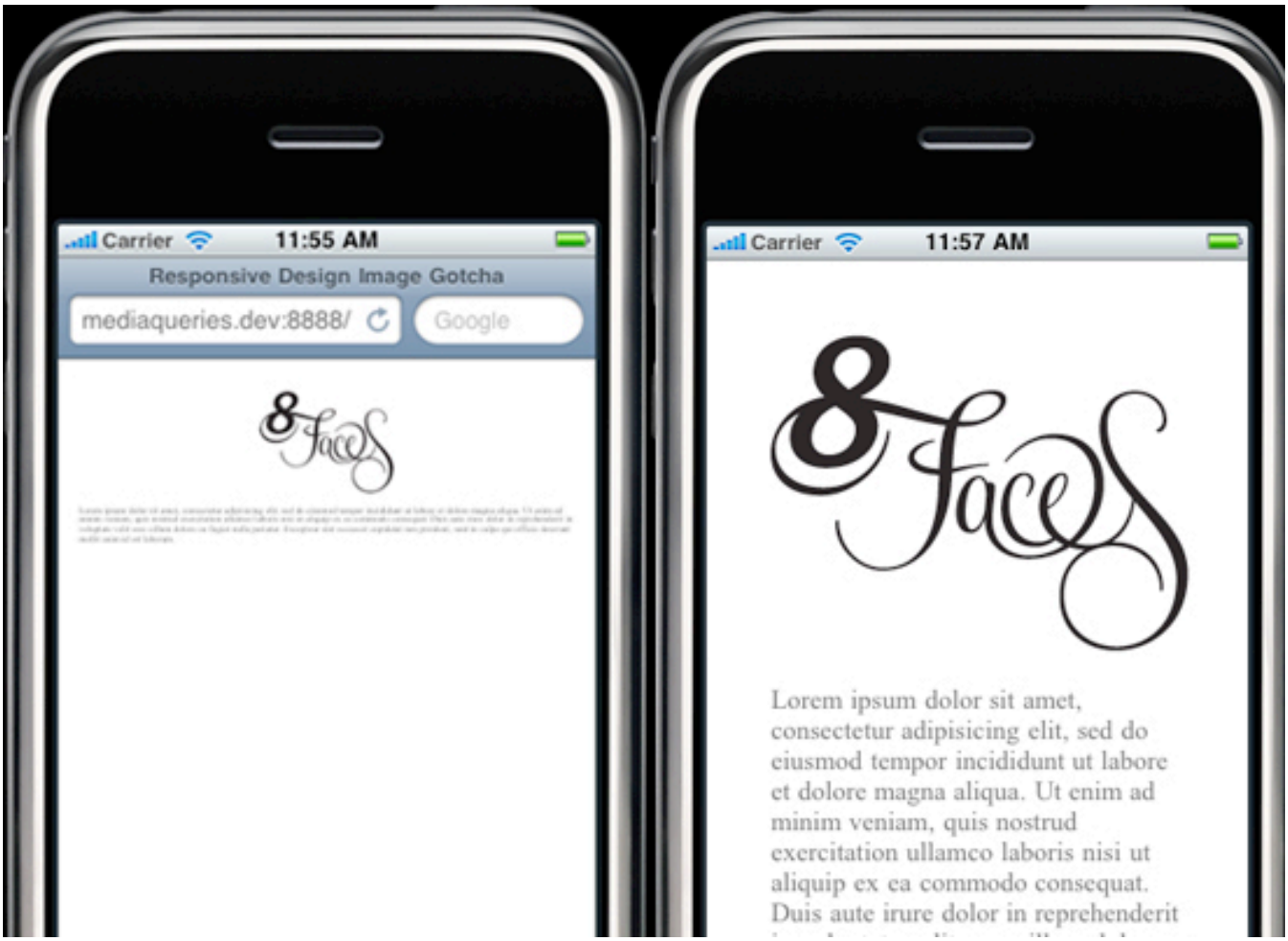
The JavaScript file inserts a base element that allows the page to separate responsive images from others and redirects them as necessary. When the page loads, all files are rewritten to their original forms, and only the large or small images are loaded as necessary. With other techniques, all higher-resolution images would have had to be downloaded, even if the larger versions would never be used. Particularly for websites with a lot of images, this technique can be a great saver of bandwidth and loading time.

This technique is fully supported in modern browsers, such as IE8+, Safari, Chrome and Opera, as well as mobile devices that use these same browsers (iPad, iPhone, etc.). Older browsers and Firefox degrade nicely and still resize as one would expect of a responsive image, except that both resolutions are downloaded together, so the end benefit of saving space with this technique is void.

STOP IPHONE SIMULATOR IMAGE RESIZING

One nice thing about the iPhone and iPod Touch is that Web designs automatically rescale to fit the tiny screen. A full-sized design, unless specified otherwise, would just shrink proportionally for the tiny browser, with no need for scrolling or a mobile version. Then, the user could easily zoom in and out as necessary.

There was, however, one issue this simulator created. When responsive Web design took off, many noticed that images were still changing proportionally with the page even if they were specifically made for (or could otherwise fit) the tiny screen. This in turn scaled down text and other elements.



(Image: Think Vitamin | Website referenced: 8 Faces)

Because this works only with Apple's simulator, we can use an Apple-specific meta tag to fix the problem, placing it *below* the website's **<head>** section. Thanks to Think Vitamin's article on image resizing, we have the meta tag below:

Setting the **initial-scale** to **1** overrides the default to resize images proportionally, while leaving them as is if their width is the same as the device's width (in either portrait or landscape mode). Apple's documentation has a lot more information on the viewport meta tag.

Custom Layout Structure

For extreme size changes, we may want to change the layout altogether, either through a separate style sheet or, more efficiently, through a CSS media query. This does not have to be troublesome; most of the styles can remain the same, while specific style sheets can inherit these styles and move elements around with floats, widths, heights and so on.

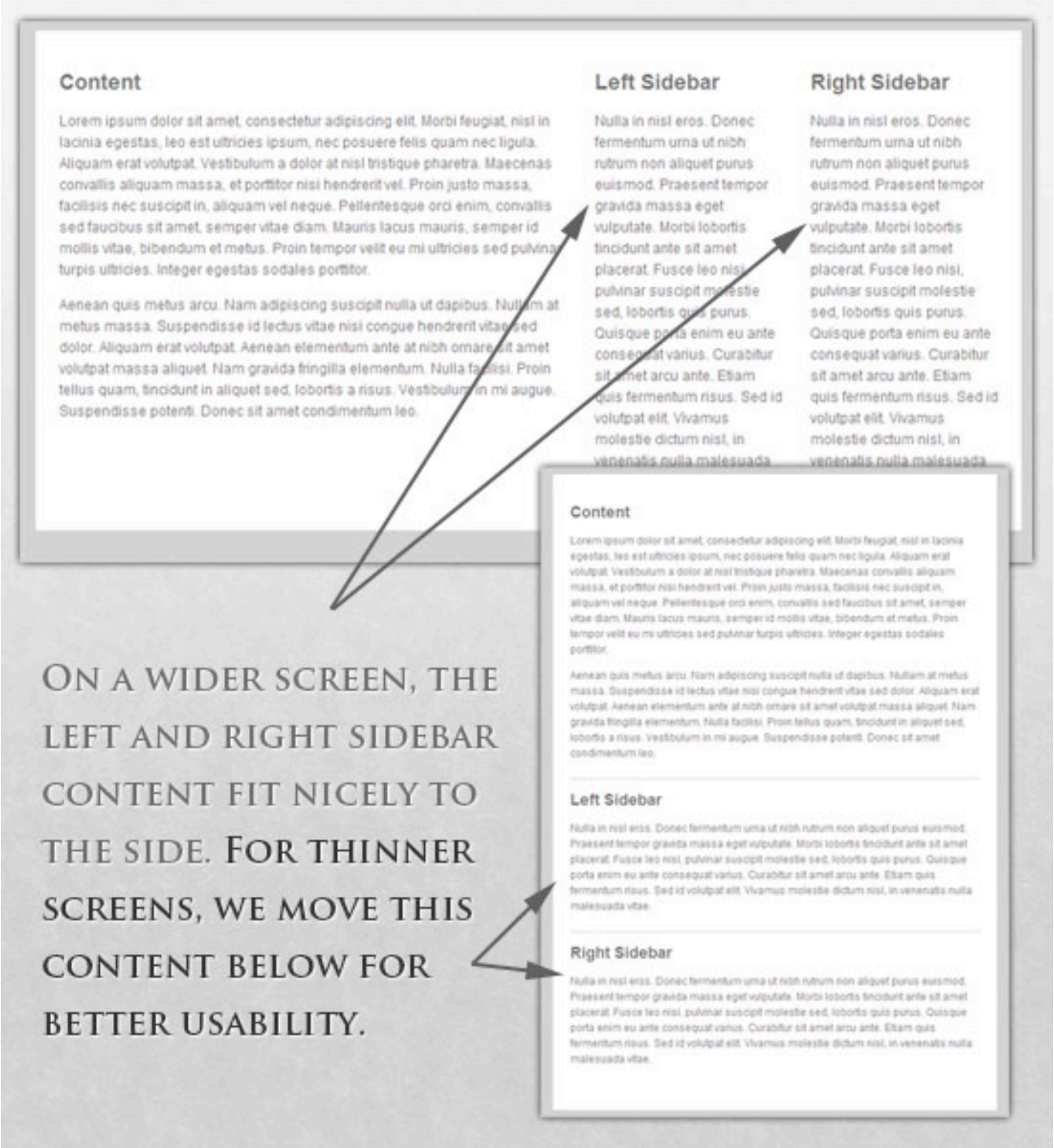
For example, we could have one main style sheet (which would also be the default) that would define all of the main structural elements, such as **`#wrapper, #content, #sidebar, #nav`**, along with colors, backgrounds and typography. Default flexible widths and floats could also be defined.

If a style sheet made the layout too narrow, short, wide or tall, we could then detect that and switch to a new style sheet. This new child style sheet would adopt everything from the default style sheet and then just redefine the layout's structure.

Here is the *style.css* (default) content:

```
/* Default styles that will carry to the child style sheet */
html,body{
   background...
   font...
   color...
}
h1,h2,h3{}
p, blockquote, pre, code, ol, ul{}
/* Structural elements */
#wrapper{
  width: 80%;
  margin: 0 auto;
  background: #fff;
  padding: 20px;
```

```
}
#content{
  width: 54%;
  float: left;
  margin-right: 3%;
}
#sidebar-left{
  width: 20%;
  float: left;
  margin-right: 3%;
}
#sidebar-right{
  width: 20%;
  float: left;
}
Here is the mobile.css (child) content:
#wrapper{
  width: 90%;
}
#content{
  width: 100%;
}
#sidebar-left{
  width: 100%;
  clear: both;
  /* Additional styling for our new layout */
  border-top: 1px solid #ccc;
  margin-top: 20px;
}
#sidebar-right{
  width: 100%;
  clear: both;
  /* Additional styling for our new layout */
  border-top: 1px solid #ccc;
  margin-top: 20px;
}
```


Content
Left Sidebar
Right Sidebar
On a wider screen, the left and right sidebar content fit nicely to the side. For thinner screens, we move this content below for better usability.

MEDIA QUERIES

CSS3 supports all of the same media types as CSS 2.1, such as **`screen`**, **`print`** and **`handheld`**, but has added dozens of new media features, including **`max-width`**, **`device-width`**, **`orientation`** and **`color`**. New devices made after the release of CSS3 (such as the iPad and Android devices) will definitely support media features. So, calling a media query using CSS3 features to target these devices would work just fine, and it will be ignored if accessed by an older computer browser that does not support CSS3.

In Ethan Marcotte's article, we see an example of a media query in action:

```
<link rel="stylesheet" type="text/css"
  media="screen and (max-device-width: 480px)"
  href="shetland.css" />
```

This media query is fairly self-explanatory: if the browser displays this page on a screen (rather than print, etc.), and if the width of the screen (not necessarily the viewport) is 480 pixels or less, then load *shetland.css*.

New CSS3 features also include **`orientation`** (portrait vs. landscape), **`device-width`**, **`min-device-width`** and more. Look at "The Orientation Media Query" for more information on setting and restricting widths based on these media query features.

One can create multiple style sheets, as well as basic layout alterations defined to fit ranges of widths — even for landscape vs. portrait orientations. Be sure to look at the section of Ethan Marcotte's article entitled "Meet the media query" for more examples and a more thorough explanation.

Multiple media queries can also be dropped right into a single style sheet, which is the most efficient option when used:

```
/* Smartphones (portrait and landscape) ---------- */
```

```
@media only screen
and (min-device-width : 320px)
and (max-device-width : 480px) {
/* Styles */
}
/* Smartphones (landscape) ----------- */
@media only screen
and (min-width : 321px) {
/* Styles */
}
/* Smartphones (portrait) ----------- */
@media only screen
and (max-width : 320px) {
/* Styles */
}
```

The code above is from a free template for multiple media queries between popular devices by Andy Clark. See the differences between this approach and including different style sheet files in the mark-up as shown in the post "Hardboiled CSS3 Media Queries."

CSS3 MEDIA QUERIES

Above are a few examples of how media queries, both from CSS 2.1 and CSS3 could work. Let's now look at some specific how-to's for using CSS3 media queries to create responsive Web designs. Many of these uses are relevant today, and all will definitely be usable in the near future.

The min-width and max-width properties do exactly what they suggest. The **`min-width`** property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored. The **`max-width`** property does just the opposite. Anything above the maximum browser or screen width specified would not apply to the respective media query.

Note in the examples below that we're using the syntax for media queries that could be used all in one style sheet. As mentioned above, the most efficient way to use media queries is to place them all in one CSS style sheet, with the rest of the styles for the website. This way, multiple requests don't have to be made for multiple style sheets.

```
@media screen and (min-width: 600px) {
     .hereIsMyClass {
          width: 30%;
          float: right;
     }
}
```

The class specified in the media query above (**`hereIsMyClass`**) will work only if the browser or screen width is above 600 pixels. In other words, this media query will run only if the minimum width is 600 pixels (therefore, 600 pixels or wider).

```
@media screen and (max-width: 600px) {
     .aClassforSmallScreens {
          clear: both;
        font-size: 1.3em;
     }
}
```

Now, with the use of **`max-width`**, this media query will apply only to browser or screen widths with a maximum width of 600 pixels or narrower.

While the above **`min-width`** and **`max-width`** can apply to either screen size or browser width, sometimes we'd like a media query that is relevant to device width specifically. This means that even if a browser or other viewing area is minimized to something smaller, the media query would still apply to the size of the actual device. The min-device-width and max-device-width media query properties are great for targeting certain devices with set

dimensions, without applying the same styles to other screen sizes in a browser that mimics the device's size.

```
@media screen and (max-device-width: 480px) {
     .classForiPhoneDisplay {
          font-size: 1.2em;
     }
}
@media screen and (min-device-width: 768px) {
     .minimumiPadWidth {
          clear: both;
        margin-bottom: 2px solid #ccc;
     }
}
```

There are also other tricks with media queries to target specific devices. Thomas Maier has written two short snippets and explanations for targeting the iPhone and iPad only:

- CSS for iPhone 4 (Retina display)
- How To: CSS for the iPad

For the iPad specifically, there is also a media query property called orientation. The value can be either landscape (horizontal orientation) or portrait (vertical orientation).

```
@media screen and (orientation: landscape) {
     .iPadLandscape {
          width: 30%;
        float: right;
     }
}
@media screen and (orientation: portrait) {
     .iPadPortrait {
          clear: both;
     }
```

```
}
```

Unfortunately, this property works only on the iPad. When determining the orientation for the iPhone and other devices, the use of **`max-device-width`** and **`min-device-width`** should do the trick.

There are also many media queries that make sense when combined. For example, the **`min-width`** and **`max-width`** media queries are combined all the time to set a style specific to a certain range.

```
@media screen and (min-width: 800px) and (max-width: 1200px) {
     .classForaMediumScreen {
          background: #cc0000;
          width: 30%;
          float: right;
     }
}
```

The above code in this media query applies only to screen and browser widths between 800 and 1200 pixels. A good use of this technique is to show certain content or entire sidebars in a layout depending on how much horizontal space is available.

Some designers would also prefer to link to a separate style sheet for certain media queries, which is perfectly fine if the organizational benefits outweigh the efficiency lost. For devices that do not switch orientation or for screens whose browser width cannot be changed manually, using a separate style sheet should be fine.

You might want, for example, to place media queries all in one style sheet (as above) for devices like the iPad. Because such a device can switch from portrait to landscape in an instant, if these two media queries were placed in separate style sheets, the website would have to call each style sheet file every time the user switched orientations. Placing a media query for both

the horizontal and vertical orientations of the iPad in the same style sheet file would be far more efficient.

Another example is a flexible design meant for a standard computer screen with a resizable browser. If the browser can be manually resized, placing all variable media queries in one style sheet would be best.

Nevertheless, organization can be key, and a designer may wish to define media queries in a standard HTML link tag:

```
<link rel="stylesheet" media="screen and (max-width: 600px)"
href="small.css" />
<link rel="stylesheet" media="screen and (min-width: 600px)"
href="large.css" />
<link rel="stylesheet" media="print" href="print.css" />
```

JAVASCRIPT

Another method that can be used is JavaScript, especially as a back-up to devices that don't support all of the CSS3 media query options. Fortunately, there is already a pre-made JavaScript library that makes older browsers (IE 5+, Firefox 1+, Safari 2) support CSS3 media queries. If you're already using these queries, just grab a copy of the library, and include it in the mark-up: *css3-mediaqueries.js*.

In addition, below is a sample jQuery snippet that detects browser width and changes the style sheet accordingly — if one prefers a more hands-on approach:

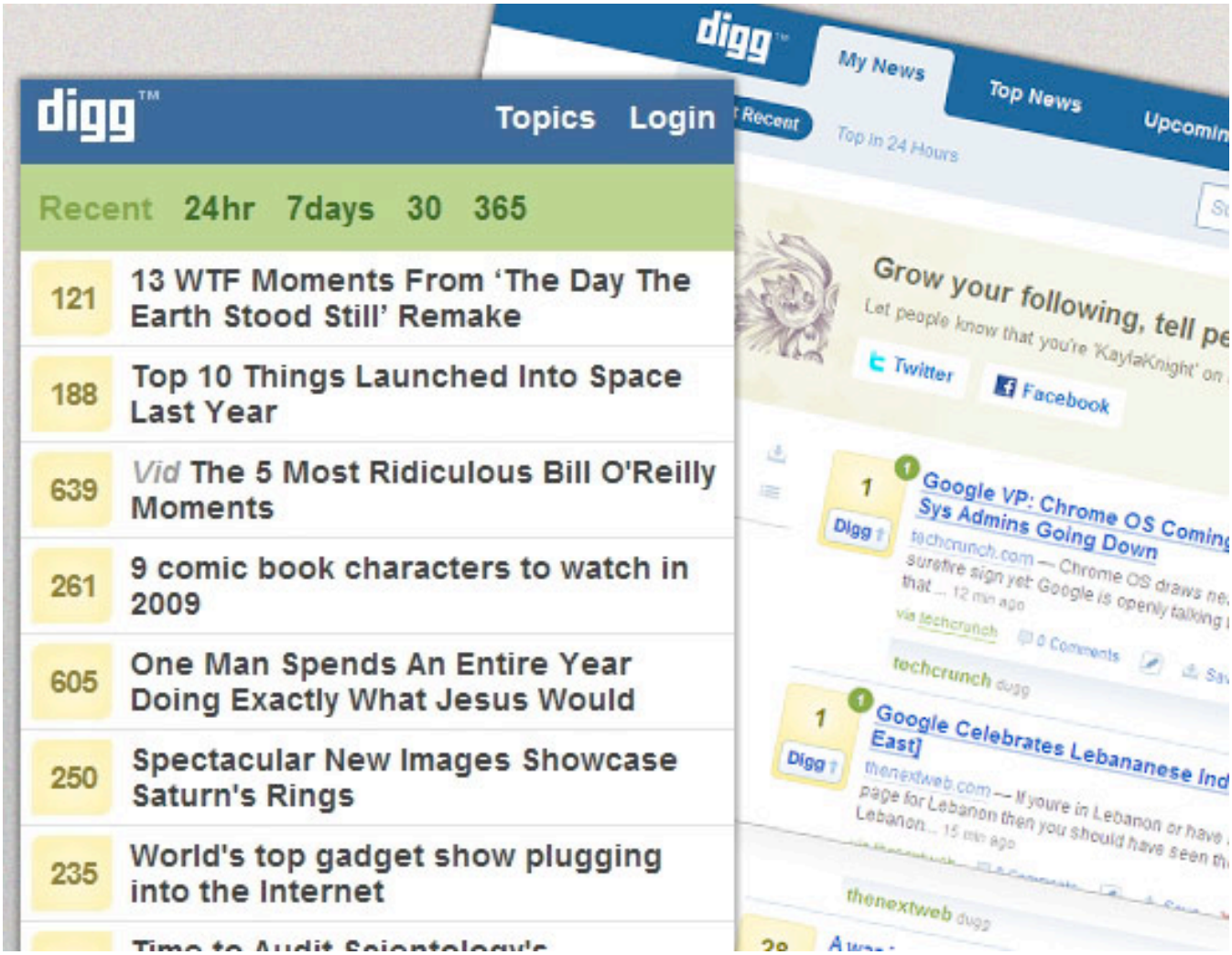
```
<script type="text/javascript" src="http://
ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></
script>
<script type="text/javascript">
  $(document).ready(function(){
      $(window).bind("resize", resizeWindow);
```

```
      function resizeWindow(e){
          var newWindowWidth = $(window).width();
          // If width width is below 600px, switch to the
mobile stylesheet
          if(newWindowWidth < 600){                $
("link[rel=stylesheet]").attr({href : "mobile.css"});
          }          // Else if width is above 600px, switch to
the large stylesheet           else if(newWindowWidth > 600){
              $("link[rel=stylesheet]").attr({href :
"style.css"});
          }
      }
  });
</script>
```

There are many solutions for pairing up JavaScript with CSS media queries. Remember that media queries are not an absolute answer, but rather are fantastic options for responsive Web design when it comes to pure CSS-based solutions. With the addition of JavaScript, we can accomodate far more variations. For detailed information on using JavaScript to mimic or work with media queries, look at “Combining Media Queries and JavaScript.”

Showing or Hiding Content

It is possible to shrink things proportionally and rearrange elements as necessary to make everything fit (reasonably well) as a screen gets smaller. It’s great that that’s possible, but making every piece of content from a large screen available on a smaller screen or mobile device isn’t always the best answer. We have best practices for mobile environments: simpler navigation, more focused content, lists or rows instead of multiple columns.

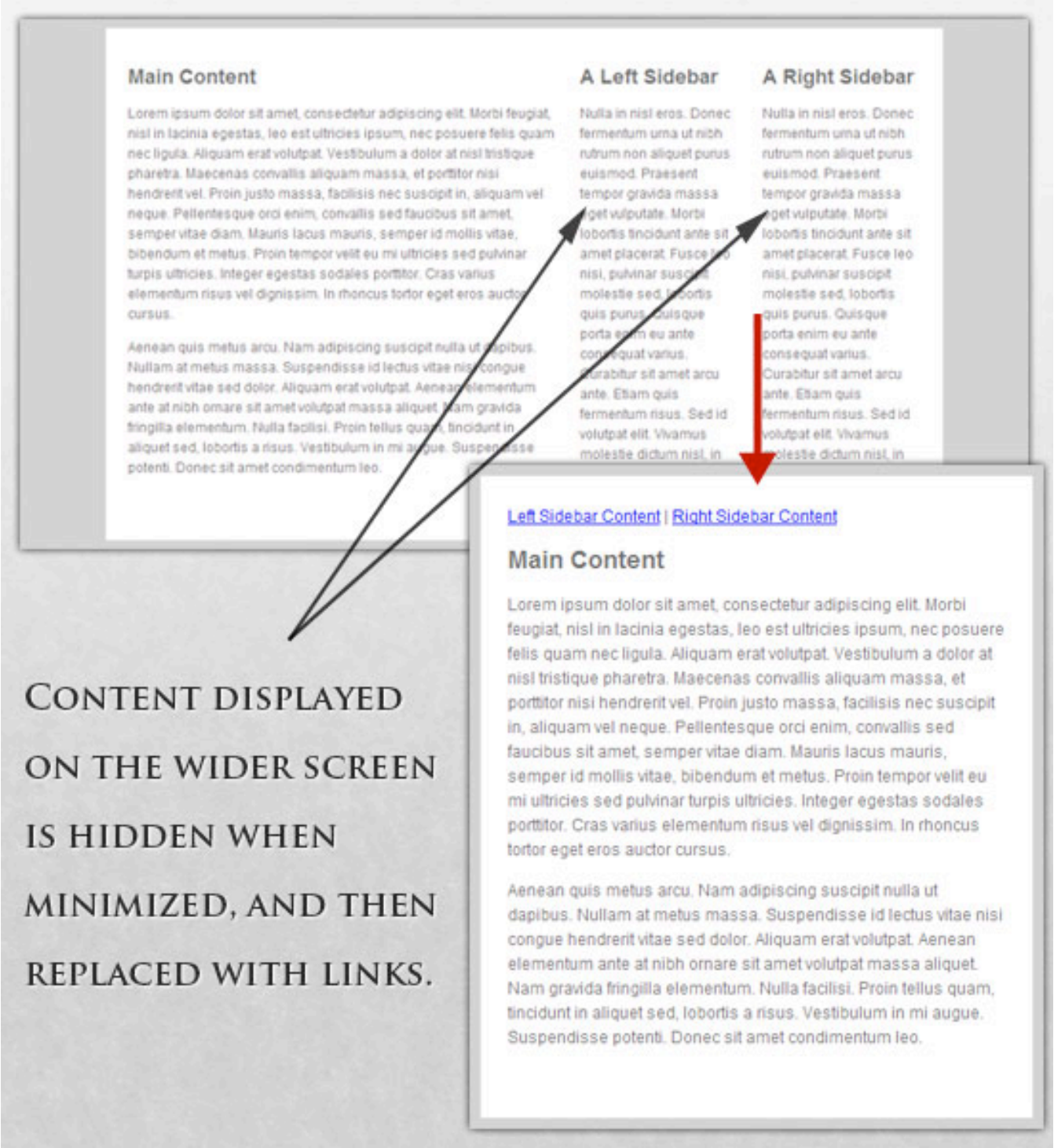


Responsive Web design shouldn't be just about how to create a flexible layout on a wide range of platforms and screen sizes. It should also be about the user being able to pick and choose content. Fortunately, CSS has been allowing us to show and hide content with ease for years!

```
display: none;
```

Either declare **`display: none`** for the HTML block element that needs to be hidden in a specific style sheet or detect the browser width and do it through JavaScript. In addition to hiding content on smaller screens, we can also hide content in our default style sheet (for bigger screens) that should be available only in mobile versions or on smaller devices. For example, as we hide major pieces of content, we could replace them with navigation to that content, or with a different navigation structure altogether.

Note that we haven't used **`visibility: hidden`** here; this just hides the content (although it is still there), whereas the `display` property gets rid of it altogether. For smaller devices, there is no need to keep the mark-up on the page — it just takes up resources and might even cause unnecessary scrolling or break the layout.



Here is our mark-up:

```
<p class="sidebar-nav"><a href="#">Left Sidebar Content</a> |
<a href="#">Right Sidebar Content</a></p>
```

```
<div id="content">
  <h2>Main Content</h2>
</div>
<div id="sidebar-left">
  <h2>A Left Sidebar</h2>
</div>
<div id="sidebar-right">
  <h2>A Right Sidebar</h2>
</div>
```

In our default style sheet below, we have hidden the links to the sidebar content. Because our screen is large enough, we can display this content on page load.

Here is the **style.css** (default) content:

```
#content{
  width: 54%;
  float: left;
  margin-right: 3%;
}

#sidebar-left{
  width: 20%;
  float: left;
  margin-right: 3%;
}

#sidebar-right{
  width: 20%;
  float: left;
}
.sidebar-nav{display: none;}
```

Now, we hide the two sidebars (below) and show the links to these pieces of content. As an alternative, the links could call to JavaScript to just cancel out the **`display: none`** when clicked, and the sidebars could be realigned in the CSS to float below the content (or in another reasonable way).

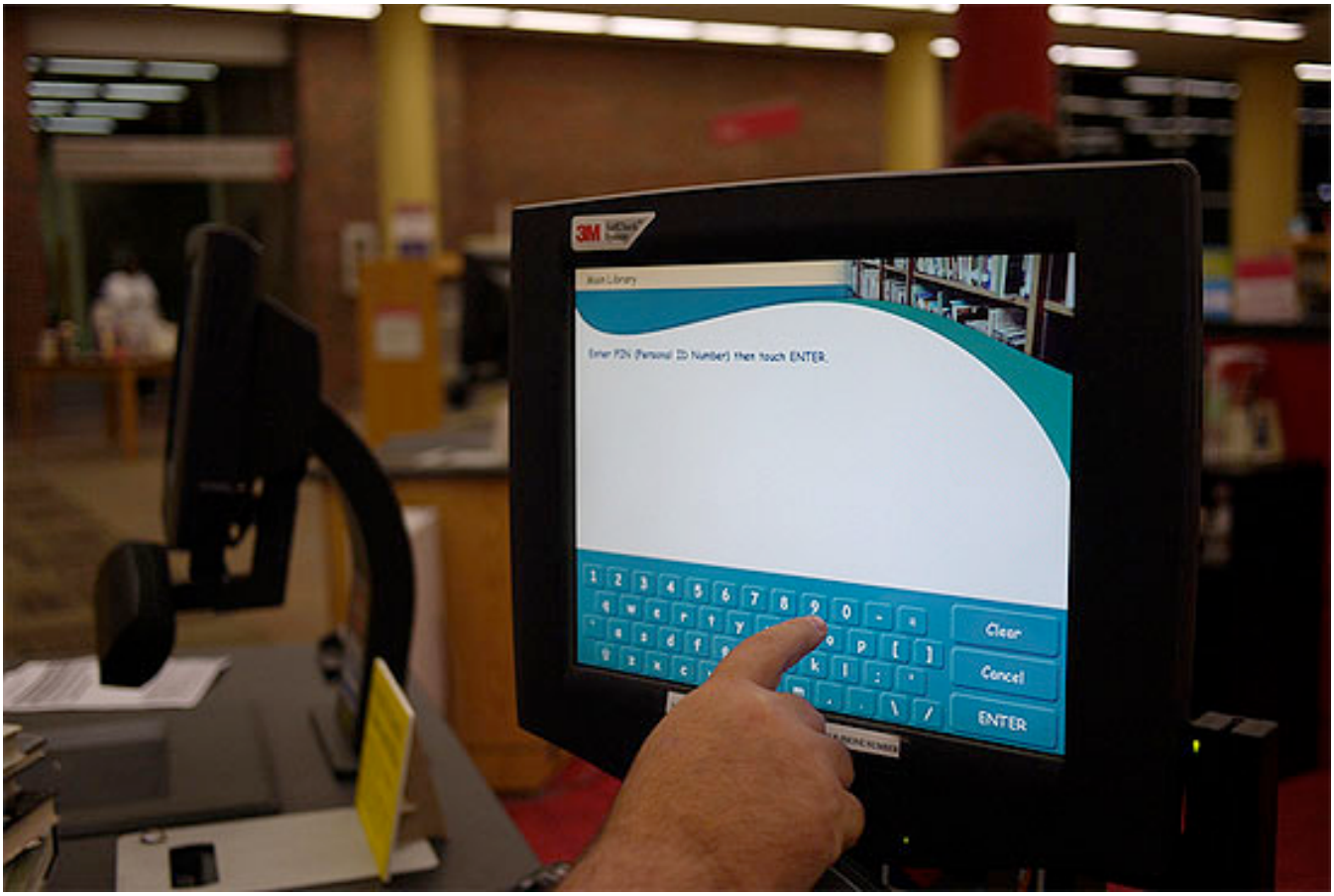
Here is the **mobile.css** (simpler) content:

```
#content{
  width: 100%;
}
#sidebar-left{
  display: none;
}
#sidebar-right{
  display: none;
}
.sidebar-nav{display: inline;}
```

With the ability to easily show and hide content, rearrange layout elements and automatically resize images, form elements and more, a design can be transformed to fit a huge variety of screen sizes and device types. As the screen gets smaller, rearrange elements to fit mobile guidelines; for example, use a script or alternate style sheet to increase white space or to replace image navigation sources on mobile devices for better usability (icons would be more beneficial on smaller screens).

TOUCHSCREENS VS. CURSORS

Touchscreens are becoming increasingly popular. Assuming that smaller devices are more likely to be given touchscreen functionality is easy, but don't be so quick. Right now touchscreens are mainly on smaller devices, but many laptops and desktops on the market also have touchscreen capability. For example, the HP Touchsmart tm2t is a basic touchscreen laptop with traditional keyboard and mouse that can transform into a tablet.



Touchscreens obviously come with different design guidelines than purely cursor-based interaction, and the two have different capabilities as well. Fortunately, making a design work for both doesn't take a lot of effort. Touchscreens have no capability to display CSS hovers because there is no cursor; once the user touches the screen, they click. So, don't rely on CSS hovers for link definition; they should be considered an additional feature only for cursor-based devices.

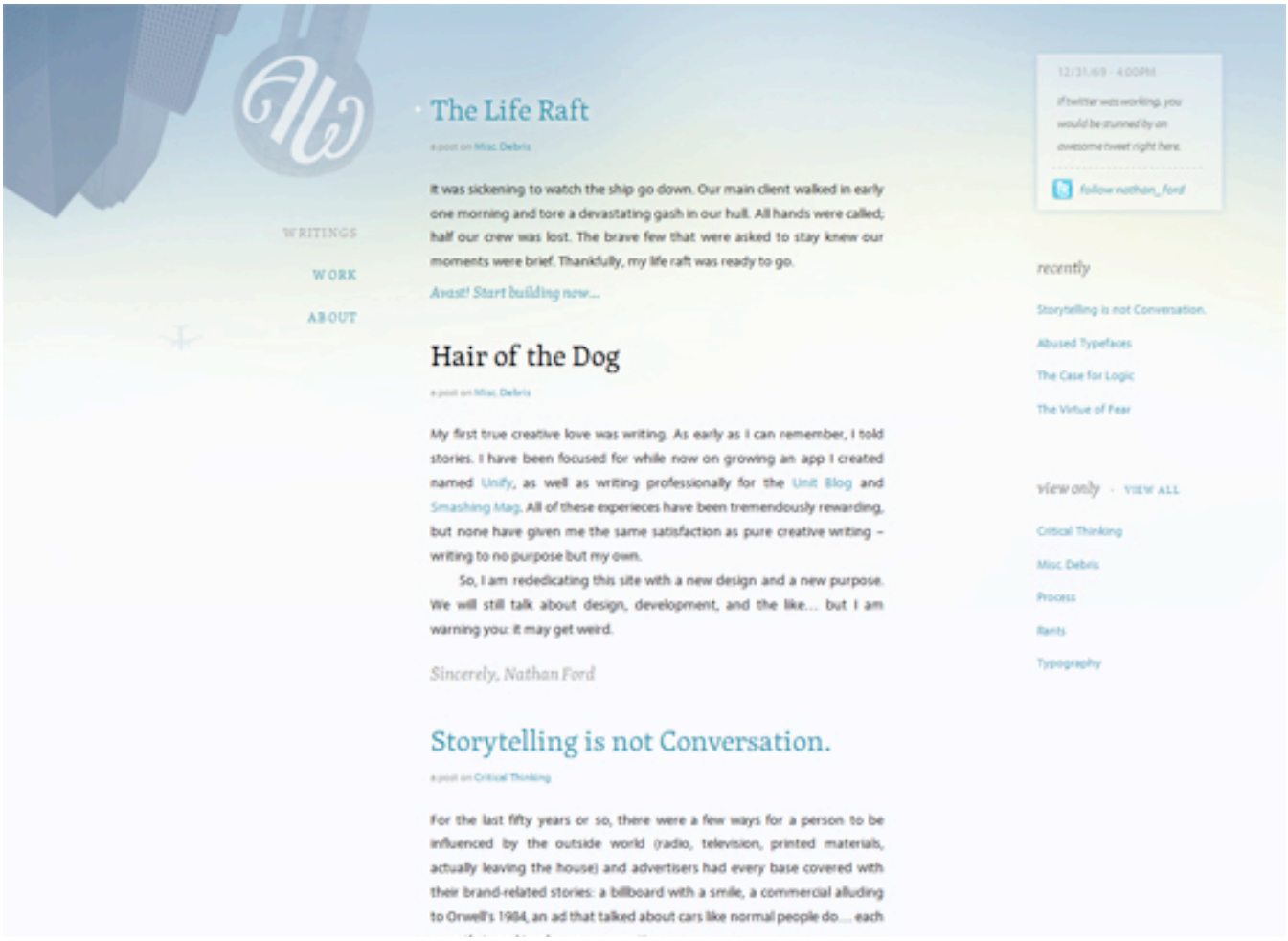
Look at the article "Designing for Touchscreen" for more ideas. Many of the design suggestions in it are best for touchscreens, but they would not necessarily impair cursor-based usability either. For example, sub-navigation on the right side of the page would be more user-friendly for touchscreen users, because most people are right-handed; they would therefore not bump or brush the navigation accidentally when holding the device in their left hand. This would make no difference to cursor users, so we might as well follow the touchscreen design guideline in this instance. Many more guidelines of this kind can be drawn from touchscreen-based usability.

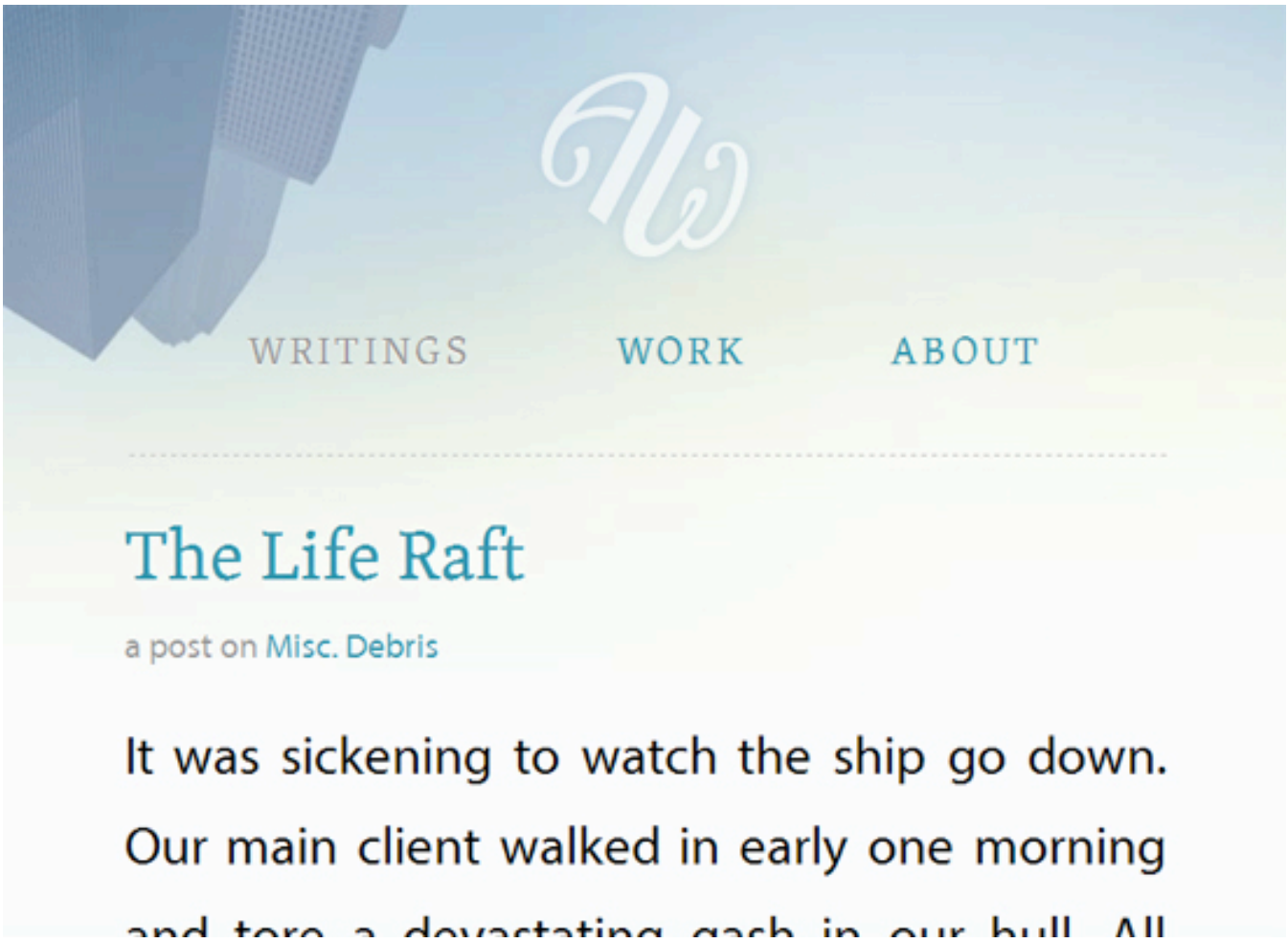
A Showcase Of Responsive Web Design

Below we have a few examples of responsive Web design in practice today. For many of these websites, there is more variation in structure and style than is shown in the pairs of screenshots provided. Many have several solutions for a variety of browsers, and some even adjust elements dynamically in size without the need for specific browser dimensions. Visit each of these, and adjust your browser size or change devices to see them in action.

Art Equals Work

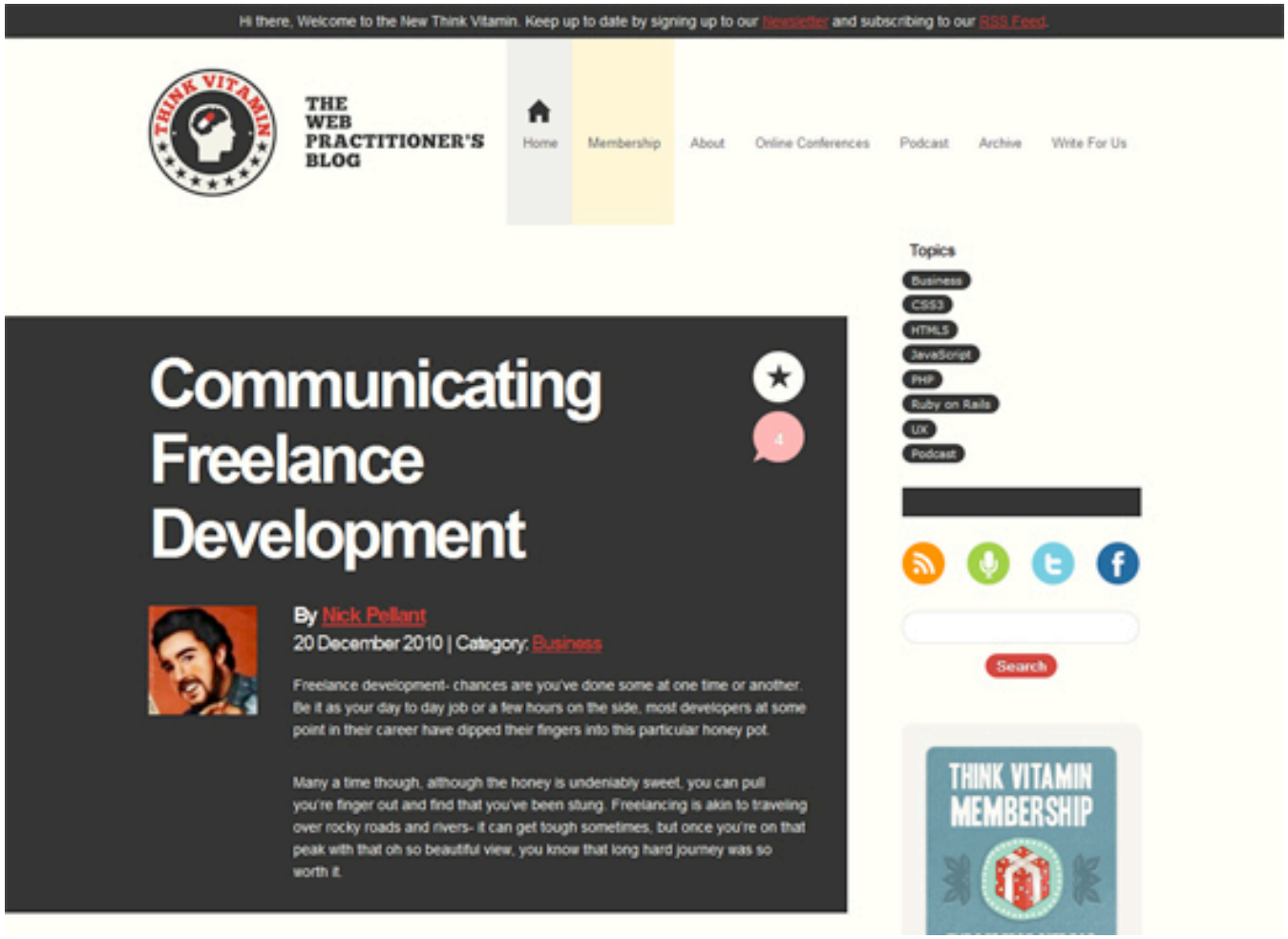
Art Equals Work is a simple yet great example of responsive Web design. The first screenshot below is the view from a standard computer screen dimension. The website is flexible with browser widths by traditional standars, but once the browser gets too narrow or is otherwise switched to a device with a smaller screen, then the layout switches to a more readable and user-friendly format. The sidebar disappears, navigation goes to the top, and text is enlarged for easy and simple vertical reading.

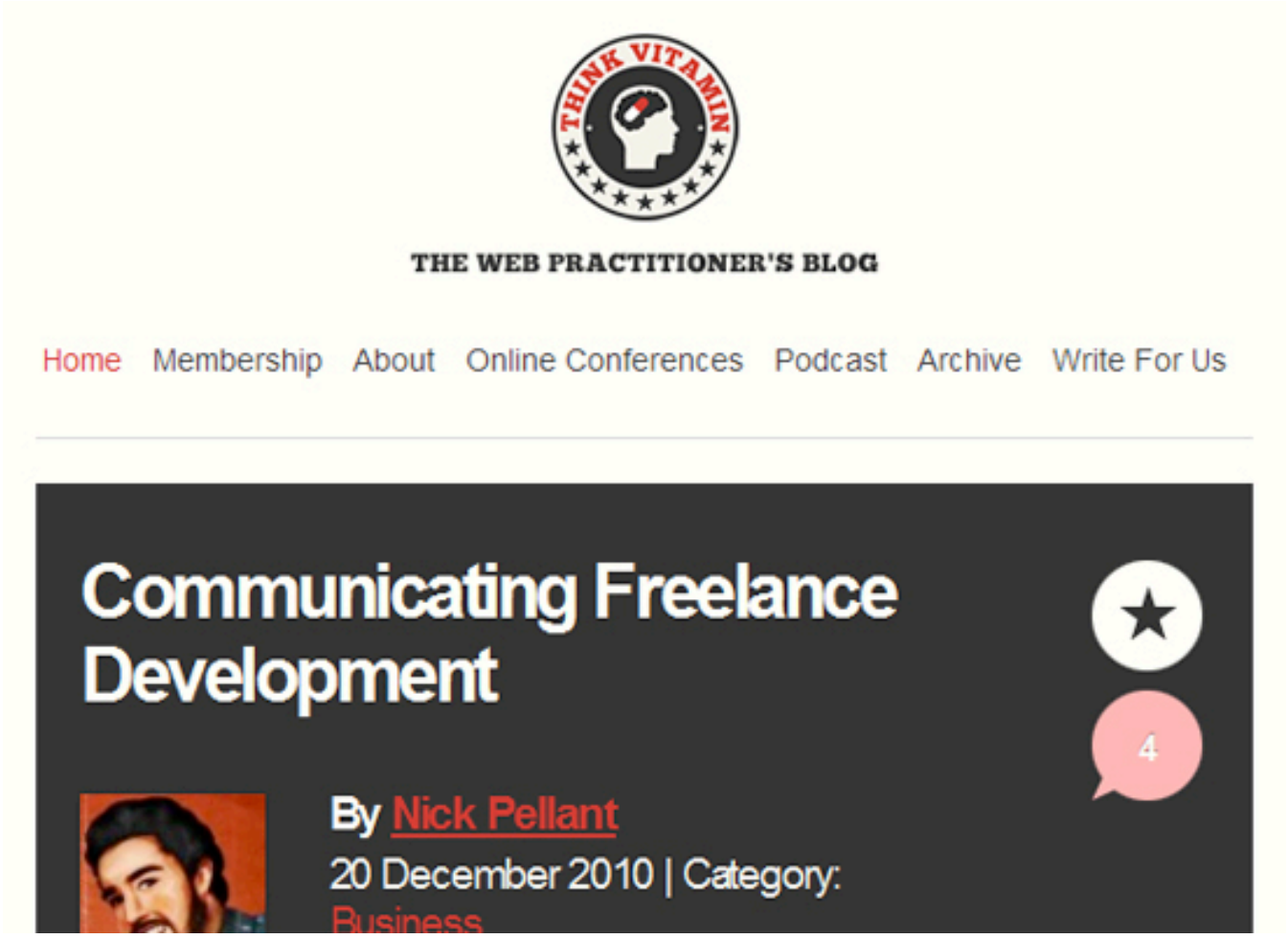



WRITINGS
WORK
ABOUT
The Life Raft
a post on Misc. Debris
It was sickening to watch the ship go down.
Our main client walked in early one morning

Think Vitamin

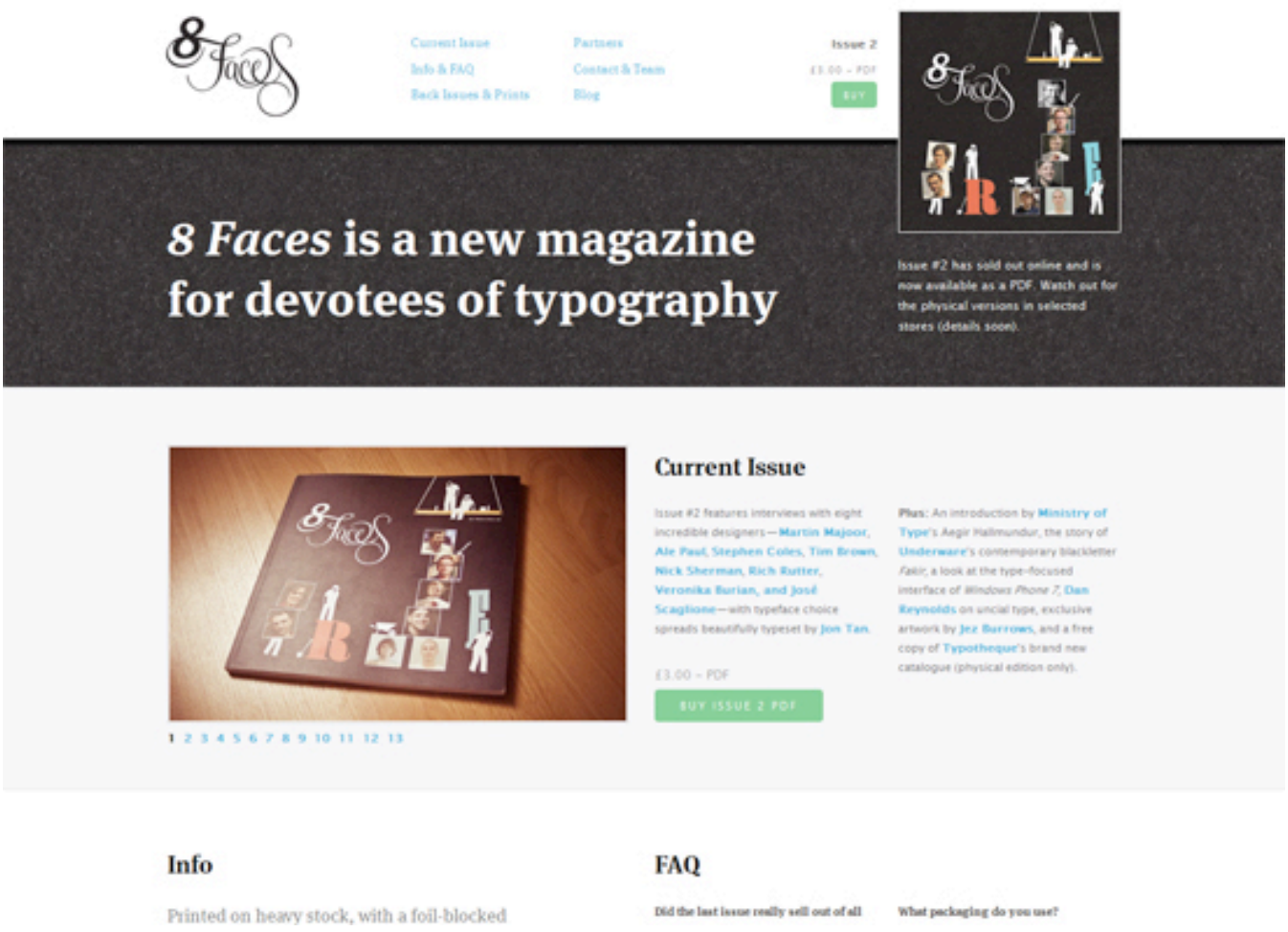
With Think Vitamin, we see a similar approach. When on a smaller screen or browser, the sidebar and top bar are removed, the navigation simplifies and moves directly above the content, as does the logo. The logo keeps its general look yet is modified for a more vertical orientation, with the tagline below the main icon. The white space around the content on larger screens is also more spacious and interesting, whereas it is simplified for practical purposes on smaller screens.

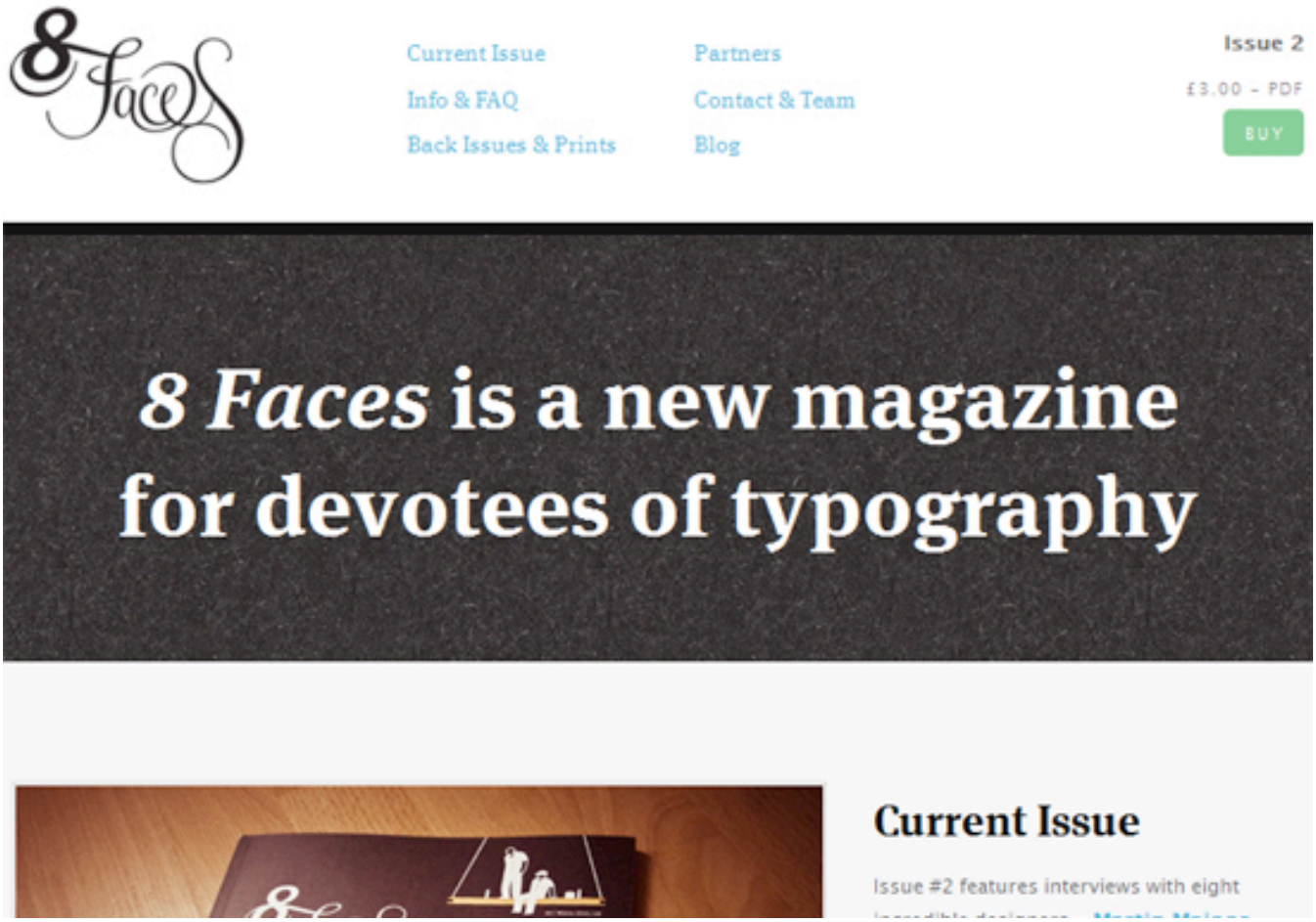



THINK VITAMIN
THE WEB PRACTITIONER'S BLOG
Home Membership About Online Conferences Podcast Archive Write For Us
Communicating Freelance Development
4
By Nick Pellant
20 December 2010 | Category:
Business

8 Faces

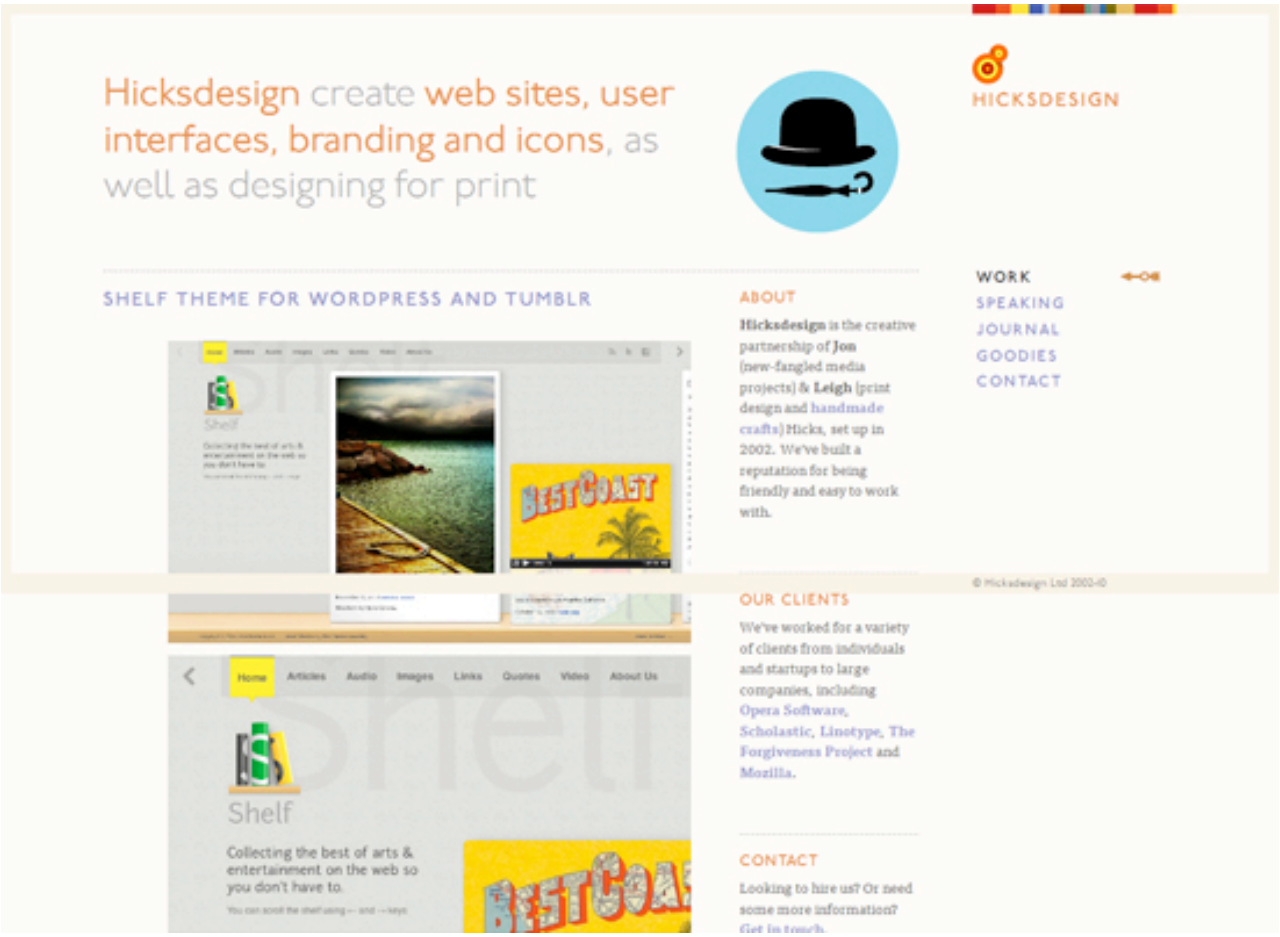
8 Faces' website design is flexible, right down to a standard netbook or tablet device, and expands in content quantity and layout width when viewed on wider screens or expanded browsers. When viewed on narrower screens, the featured issue on the right is cut out, and the content below is shortened and rearranged in layout, leaving only the essential information.

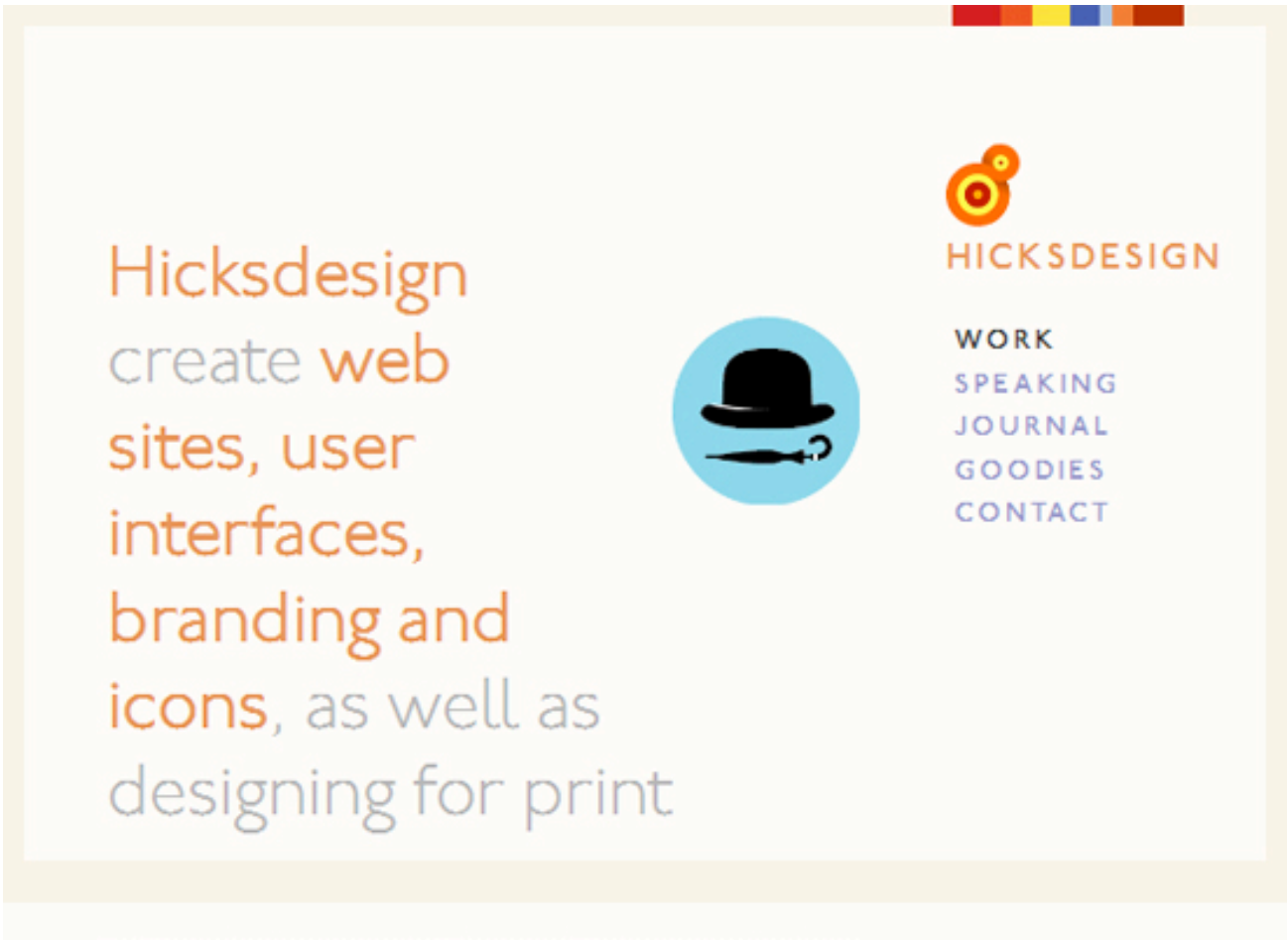



8 Faces
Current Issue
Info & FAQ
Back Issues & Prints
Partners
Contact & Team
Blog
Issue 2
£3.00 – PDF
BUY
8 Faces is a new magazine for devotees of typography
Current Issue
Issue #2 features interviews with eight

Hicksdesign

The Hicksdesign website has three columns when viewed on a conventional computer screen with a maximized browser. When minimized in width, the design takes on a new layout: the third column to the right is rearranged above the second, and the logo moves next to the introductory text. Thus, no content needs to be removed for the smaller size. For even narrower screens and browser widths, the side content is removed completely and a simplified version is moved up top. Finally, the font size changes with the screen and browser width; as the browser gets narrower, the font size throughout gets smaller and remains proportional.




Hicksdesign create web sites, user interfaces, branding and icons, as well as designing for print
HICKSDESIGN
WORK
SPEAKING
JOURNAL
GOODIES
CONTACT

Information Architects

Here is a great example of a flexible image. The image in this design automatically resizes after certain “break” points, but in between those width changes, only the side margins and excess white space are altered. On smaller screens and minimized browsers, the navigation simplifies and the columns of navigation at the top fall off. At the design’s smallest version, the navigation simplifies to just a drop-down menu, perfect for saving space without sacrificing critical navigation links.

November 18, 2010

News on iPad, the Obvious

Garret Keizer

The website for Garret Keizer is fully flexible in wider browsers and on larger screens: the photo, logo and other images resize proportionally, as do the headings and block areas for text. At a few points, some pieces of text change in font size and get smaller as the screen or browser gets narrower. After a certain break point, the layout transforms into what we see in the second screenshot below, with a simple logo, introductory text and a simple vertical structure for the remaining content.

Simon Collison

With four relatively content-heavy columns, it's easy to see how the content here could easily be squished when viewed on smaller devices. Because of the easy organized columns, though, we can also collapse them quite simply when needed, and we can stack them vertically when the space doesn't allow for a reasonable horizontal span. When the browser is minimized or the user is on a smaller device, the columns first collapse into two and then into one. Likewise, the horizontal lines for break points also change in width, without changing the size or style of each line's title text.


Established Nottingham 2003
THE CELEBRATED NEW MISCELLANY OF
MR. SIMON COLLISON
A.K.A COLLY
Bottled for your pleasure
POTTED AUTOBIOGRAPHY
Hello. I'm a freelance designer, speaker, and author based in Nottingham, England. I've written a few books, and I love doing presentations and workshops, plus occasional interviews. Read More →
Dropping science like it's hot
THE SPLENDID JOURNAL
That was the year that was I'd been hesitant about writing a review of my 2010, partly because it is something I've avoided each year, and also because I had such a brilliant… More →
Catalogued nocturnal matter
EXHAUSTIVE ARCHIVES
819—My year in music
818—Adaptive Systems at HD…
817—Twenty-four Ways
816—Pipeline interview
815—To Belfast and back
Mr. Collison is currently
AVAILABLE FOR HIRE
Opinions & quotes this way Drop me a line if you wish. I'm currently considering all new projects, including design and development, writing, presentations, workshops, and consultation. →
EXTERNAL REFERENCES (VIEW ALL)
Conference and speaking history
LANYRD PROFILE
Mr. Collison is organizing some
NEW ADVENTURES
Images from the field
FLICKR PHOTOGRAPHS
The tweets of @colly
TWITTER HAPPIER
Playing on the gramophone
LAST.FM SCROBBLES
Notable items from other folk
PINBOARD BOOKMARKS
Sneak peaks at my work
DRIBBBLE SHOTS
I've been, therefore I am
GOWALLA PASSPORT

CSS Tricks

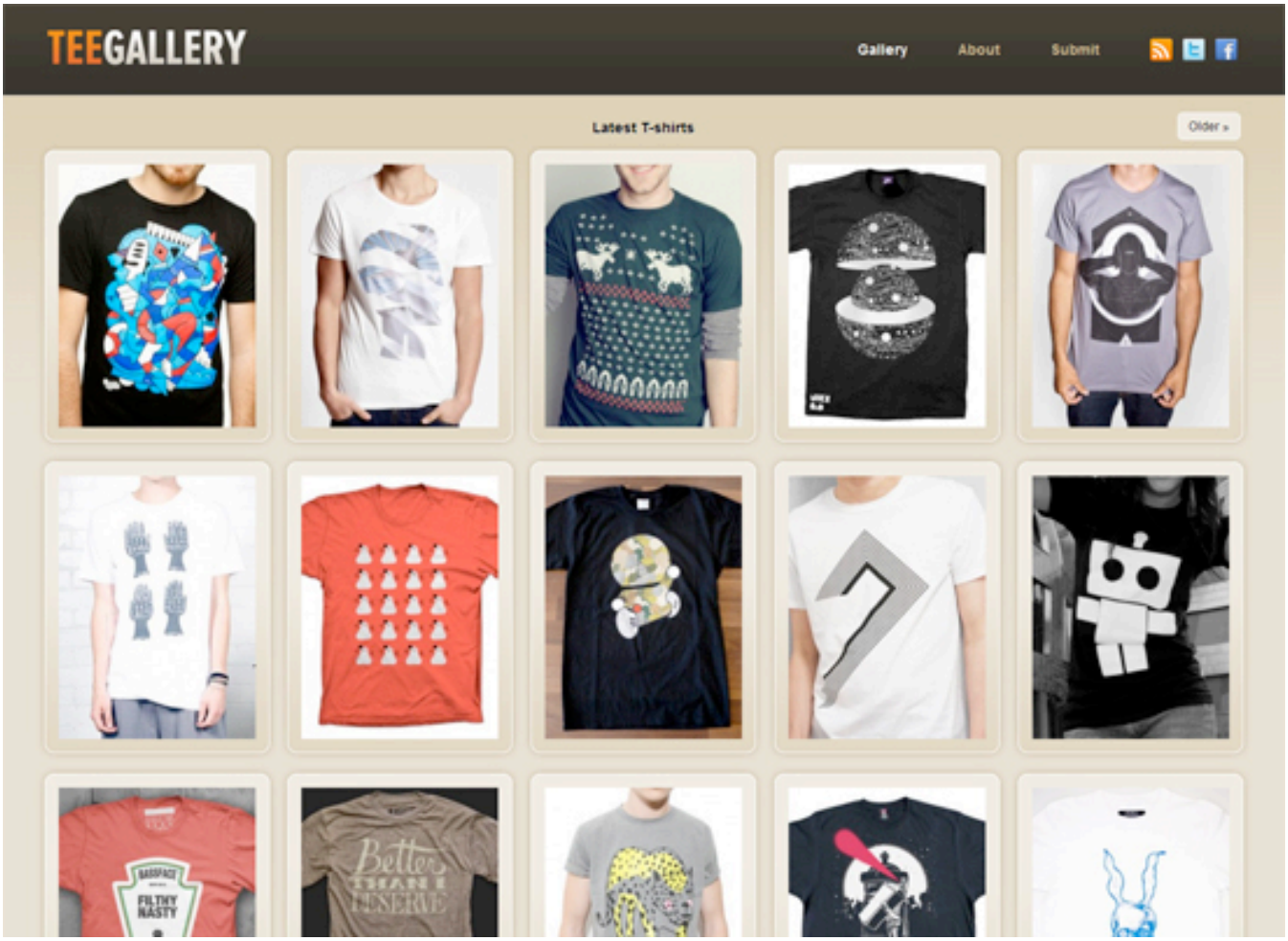
On the CSS Tricks website, like many other collapsible Web designs, the sidebars with excess content are the first to fall off when the screen or browser gets too narrow. On this particular website, the middle column or first sidebar to the left was the first to disappear; and the sidebar with the ads and website extras did the same when the browser got even narrower. Eventually, the design leaves the posts, uses less white space around the navigation and logo and moves the search bar to below the navigation. The remaining layout and design is as flexible as can be because of its simplicity.

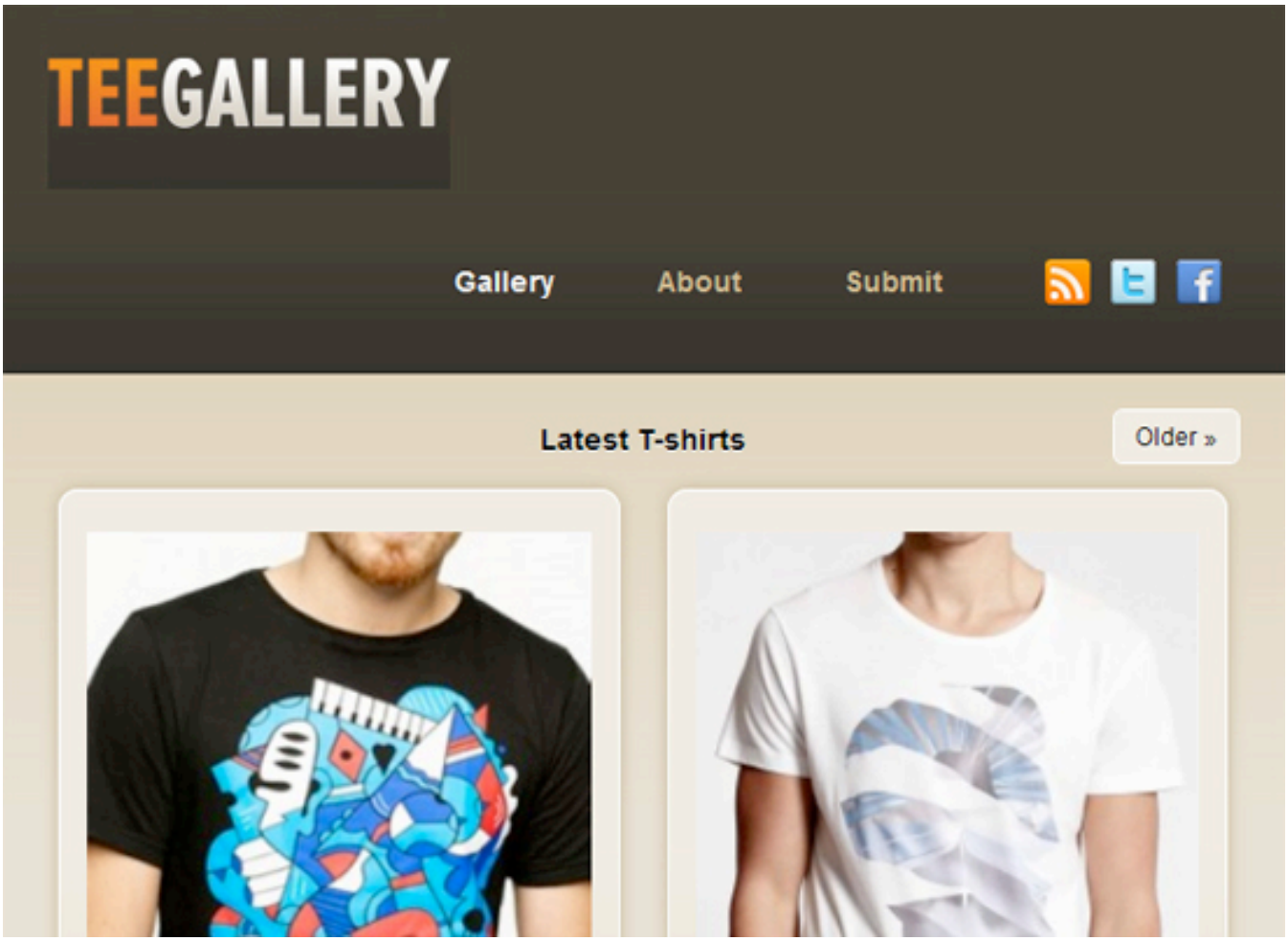

CSS-TRICKS
A Web Design Community
curated by Chris Coyier
DESIGN TO XHTML
PSD to HTML conversion
PSD2HTML.com with over 300 professionals takes the designs to HTML and beyond
HOME
FORUMS
VIDEOS
DOWNLOADS
SNIPPETS
LATEST ARTICLES
Links
1/3/2011
No Comments
Got a couple of slightly-oldie-but-goodies for you:
A jQuery plugin by Manos Malihutsakis which allows you do to create custom scrollbars for content areas that do some neat things like move their position and size, style them differently, and use easing.
Benjamin De Cock's CSS Playground has a number of stripped down and clever CSS ideas like a slideshow, spinner graphic without images, and some neat navigation ideas.
Caleb Ogden uses some CSS3 to make a submit
...
Read on! →
Why use Classes or IDs on the HTML element?
1/2/2011
33 Comments
Reader Nicolas writes in:
HOT LINKS
Striped and Checkerboard Backgrounds Without Images →
Lea Verou with a clever idea on using CSS3 gradients that repeat (by setting background size) to create striped and checkboard backgrounds with no images.
Body Borders without Markup →
Harry Roberts with a nice technique on using pseudo elements to create the body border effect without extraneous markup (same idea with extra markup gets you older browser support). Read the comments for some other good suggestions.
Join over 60,000 other web design enthusiasts and subscribe now!
Clutterpad
Awesome
FlippingBook
Premium WP Plugins
on CodeCanyon
PROFESSIONAL
FRESHBOOKS
PSD to XHTML
XhtmlWeaver
YOU DESIGN, WE CODE


CSS-TRICKS
A Web Design Community
curated by Chris Coyier
HOME
FORUMS
VIDEOS
DOWNLOADS
SNIPPETS
Google™ Custom Search
LATEST ARTICLES
Links
1/3/201
No Comments

Tee Gallery

As one can see, the main navigation here is the simple layout of t-shirt designs, spanning both vertically and horizontally across the screen. As the browser or screen gets smaller, the columns collapse and move below. This happens at each break point when the layout is stressed, but in between the break points, the images just change proportionally in size. This maintains balance in the design, while ensuring that any images (which are essential to the website) don't get so small that they become unusable.




TEEGALLERY
Gallery
About
Submit
Latest T-shirts
Older »

City Crawlers: Berlin

When varied between larger screen sizes and browser widths, this design remains flexible. It also remains flexible after a few layout pieces collapse into a more vertical orientation for small screens and narrow browsers. At first, the introductory image, logo and navigation image links resize proportionally to accommodate variations in screen and browser widths, as do the blocks of content below. The bottom columns of content eventually collapse and rearrange above or below other pieces, until (at the narrowest point) they are all stacked vertically. In the layout for the smallest screen and narrowest browser, the slideshow is left out altogether, the navigation is moved below the logo and other images are also removed.


CITY CRAWLERS BERLIN
THE BOOK PURCHASE NEWS PRESS CONTACT
THE SEARCH TO FIND NEW PATTERNS, BEHAVIORS, EXCEPTIONS AND BEAUTIFUL FACTS THROUGH INFORMATION
NEWS & UPDATES
The Information Architecture of Cities
Cities can be viewed as information architecture systems. The term makes sense not in reference to the the design of buildings, but rather to the components of a complex system that interact with each other. In a

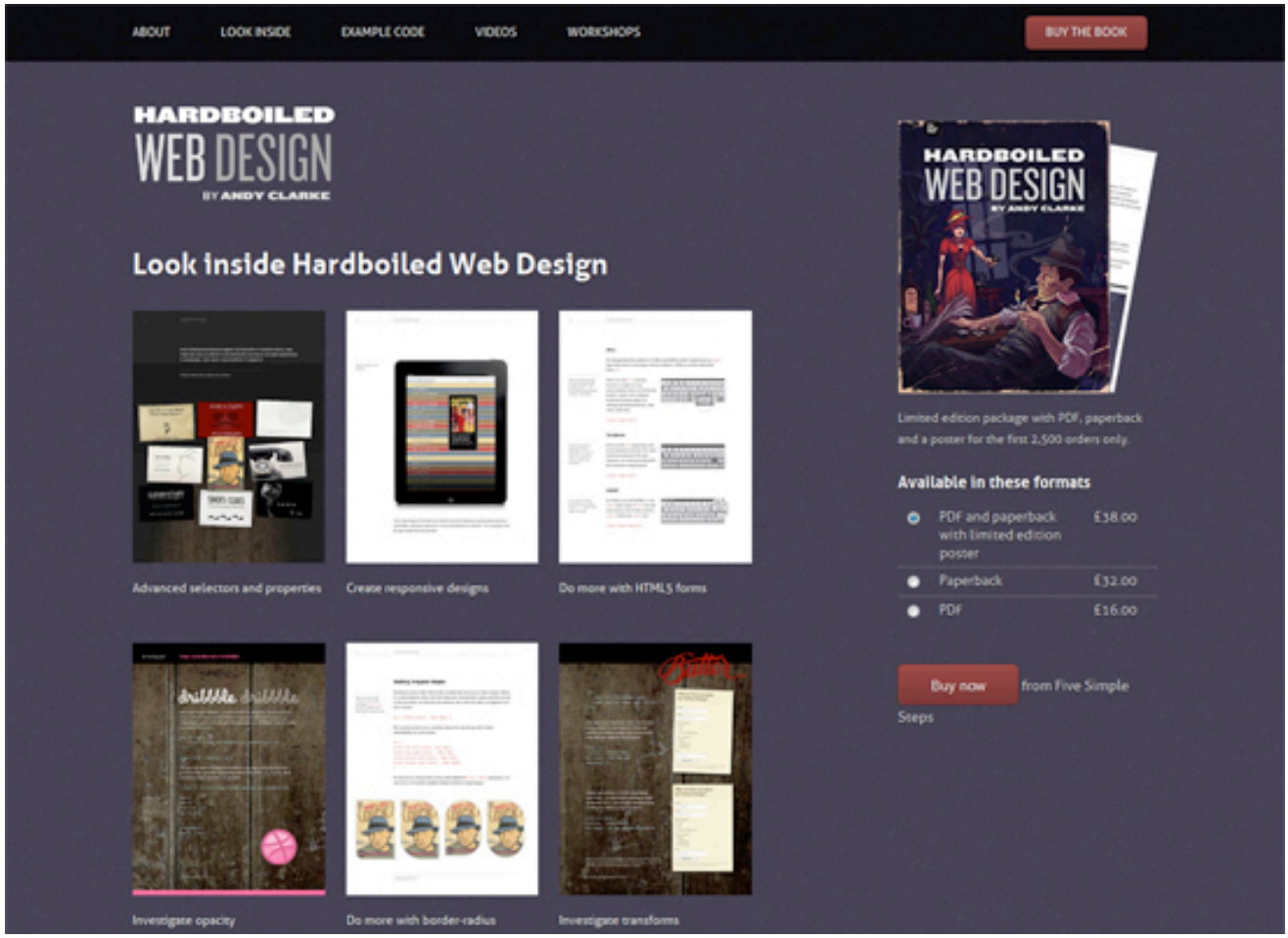
Ten by Twenty

Ten by Twenty is another design that does not resort to changing layout structure at all after certain break points, but rather simplifies responsive Web design by making everything fully flexible and automatically resizing, no matter what the screen or browser width. After a while, the design does stress a bit and could benefit from some rearrangement of content. But overall, the image resizing and flexible content spaces allow for a fairly simple solution that accommodates a wide range of screen sizes.


edmerritt.com
tenbytwenty.com
Ten by twenty
Fonts, Wordpress themes & icons
designed by Ed Merritt
NEVIS
Nevis
FREE
TYPEFACE
Jura
Jura
FREE
TYPEFACE
alborz
Alborz
£2
TYPEFACE

Hardboiled Web Design

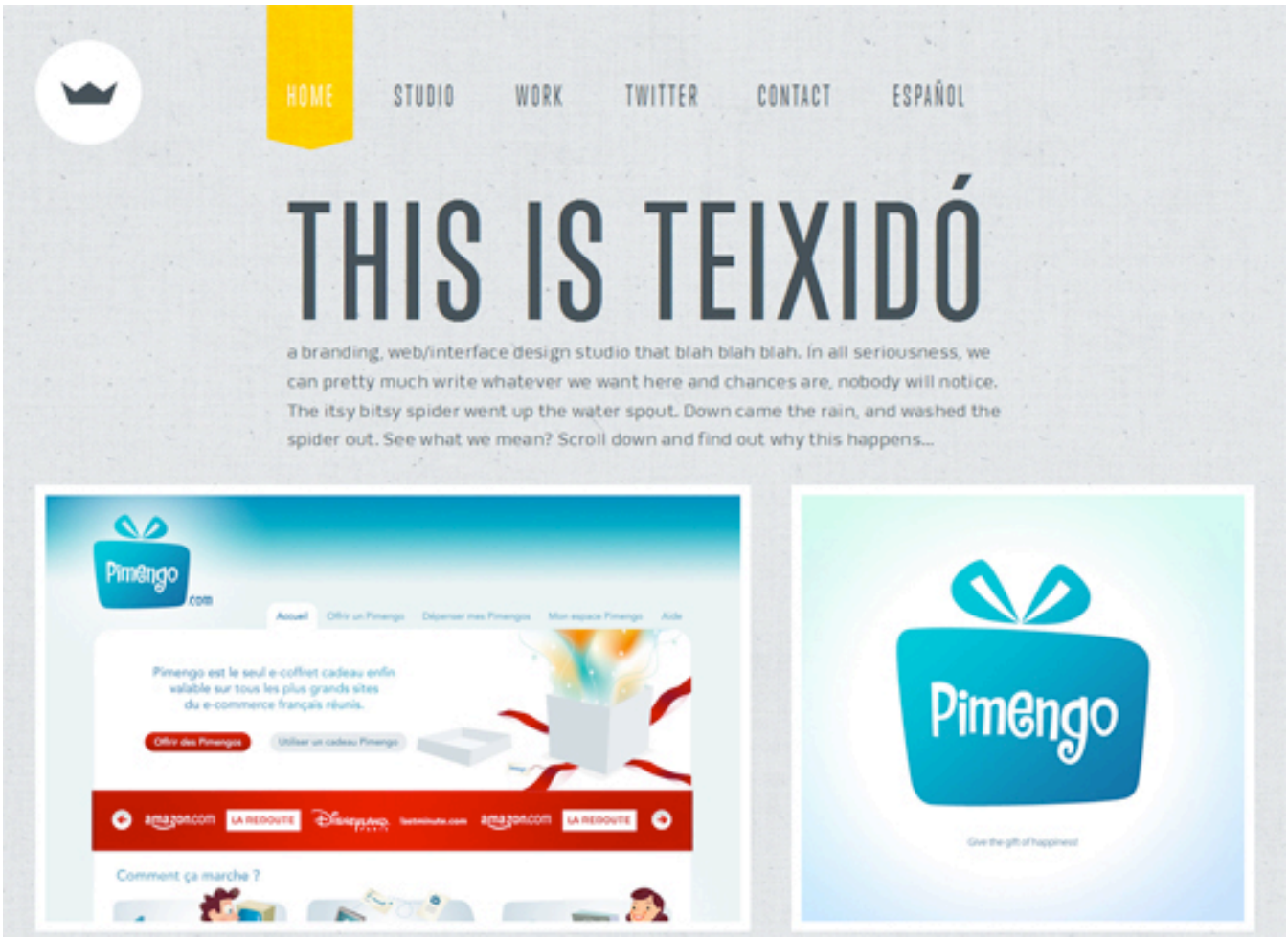
On wide screens and browsers, all of the content on this simply designed website is well organized into columns, sidebar and simple navigation up top. It's a fairly standard and efficient layout. On smaller screens, the sidebar is the first to drop off, and its content is moved below the book previews and essential information. Being limited in space, this design preserves its important hierarchy. Whereas on a wider screen we'd look left to right, on a narrower screen we'd tend to look from top to bottom. Content on the right is moved below content that would appear on the left on a wider screen. Eventually, when the horizontal space is fully limited, the navigation is simplified and stacked vertically, and some repeated or inessential elements are removed.




ABOUT
LOOK INSIDE
EXAMPLE CODE
VIDEOS
BUY THE BOOK
HARDBOILED
WEB DESIGN
BY ANDY CLARKE
Look inside Hardboiled Web Design

Teixido

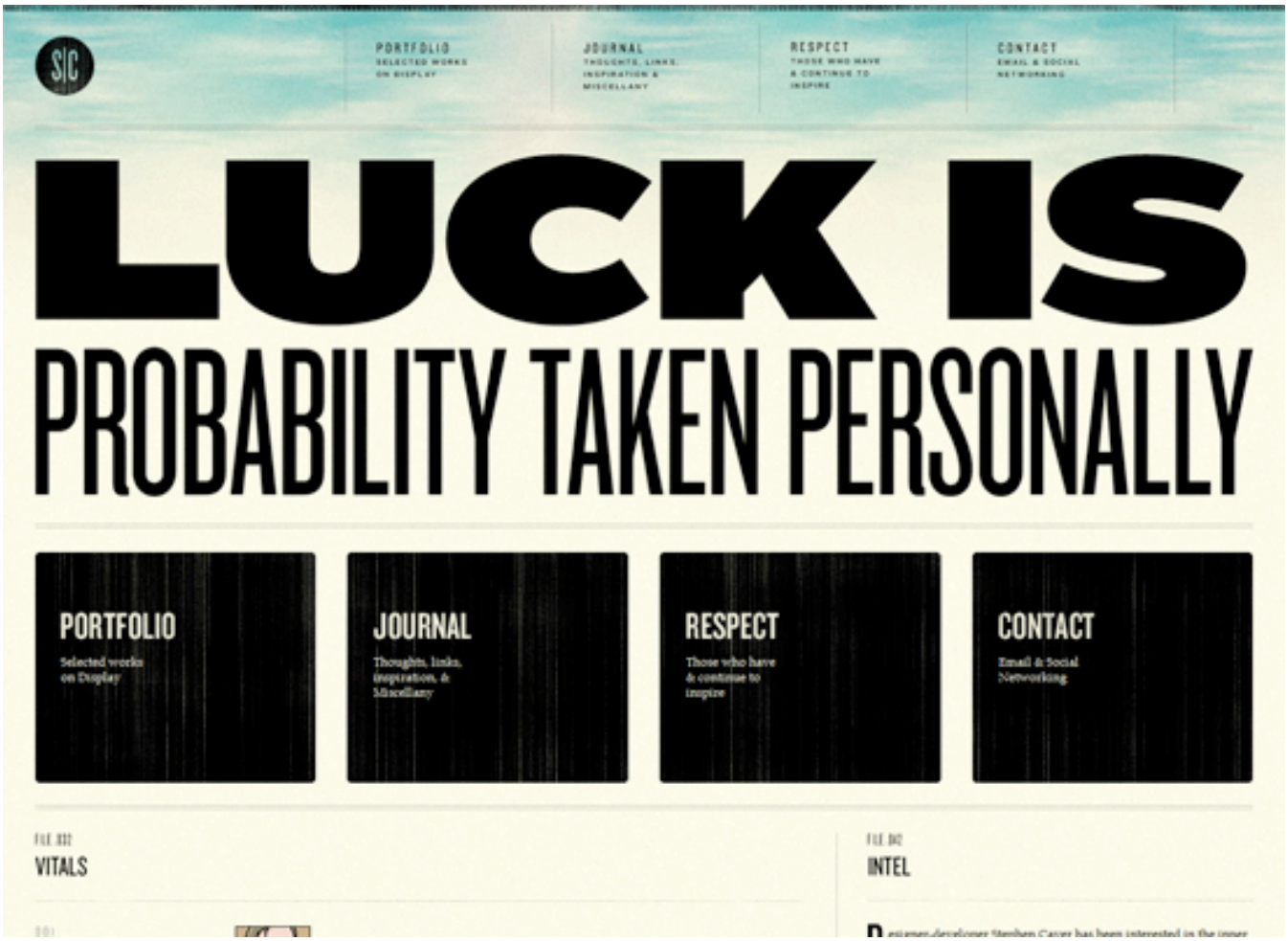
This design features a complex layout that looks inspired by a print style. When viewed on a standard wide computer screen, more portfolio pieces are featured and spanned horizontally across the page. As one moves down the page, more graphics and imagery span the space. On a smaller screen, the portfolio piece is cut down to one, and then eventually left out altogether for very small screens and narrow browsers. The visualizations below collapse into fewer columns and more rows, and again, some drop off entirely for very small screens. This design shows a creative and intelligent way to make a not-so-common layout work responsively.




HOME
STUDIO
WORK
TWITTER
CONTACT
ESPAÑOL
THIS IS TEIXIDÓ
a branding, web/interface design studio that blah blah blah. In all seriousness, we can pretty much write whatever we want here and chances are, nobody will notice. The itsy bitsy spider went up the water spout. Down came the rain, and washed the spider out. See what we mean? Scroll down and find out why this happens...
Pimengo
.com

Stephen Caver

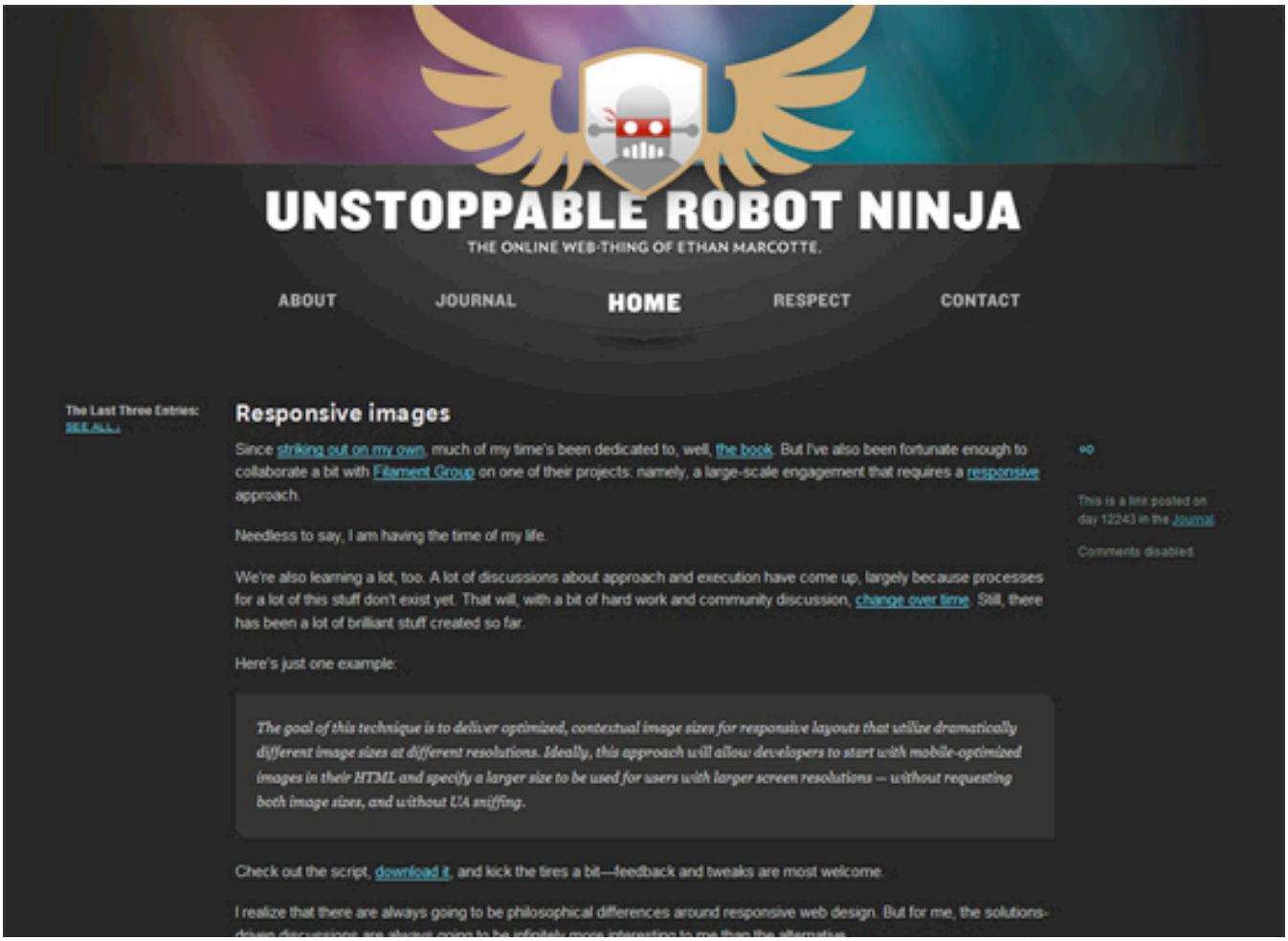
This design has three main stages at which the design and layout collapse into a more user-friendly form, depending on how wide the screen or browser is. The main image (featuring type) is scaled proportionally via a flexible image method. Each “layout structure” is fully flexible until it reaches a breaking point, at which point the layout switches to something more usable with less horizontal space. The bottom four columns eventually collapse into two, the logo moves above the navigation, and the columns of navigation below are moved on top or below each other. At the design’s narrowest stage, the navigation is super-simplified, and some inessential content is cut out altogether.




SIC
PORTFOLIO
SELECTED WORKS ON DISPLAY
JOURNAL
THOUGHTS, LINKS, INSPIRATION & MISCELLANY
RESPECT
THOSE WHO HAVE & CONTINUE TO INSPIRE
CONTACT
EMAIL & SOCIAL NETWORKING
LUCK IS
PROBABILITY TAKEN PERSONALLY
PORTFOLIO
Selected works on Display
JOURNAL
Thoughts, links, inspiration, &

Unstoppable Robot Ninja

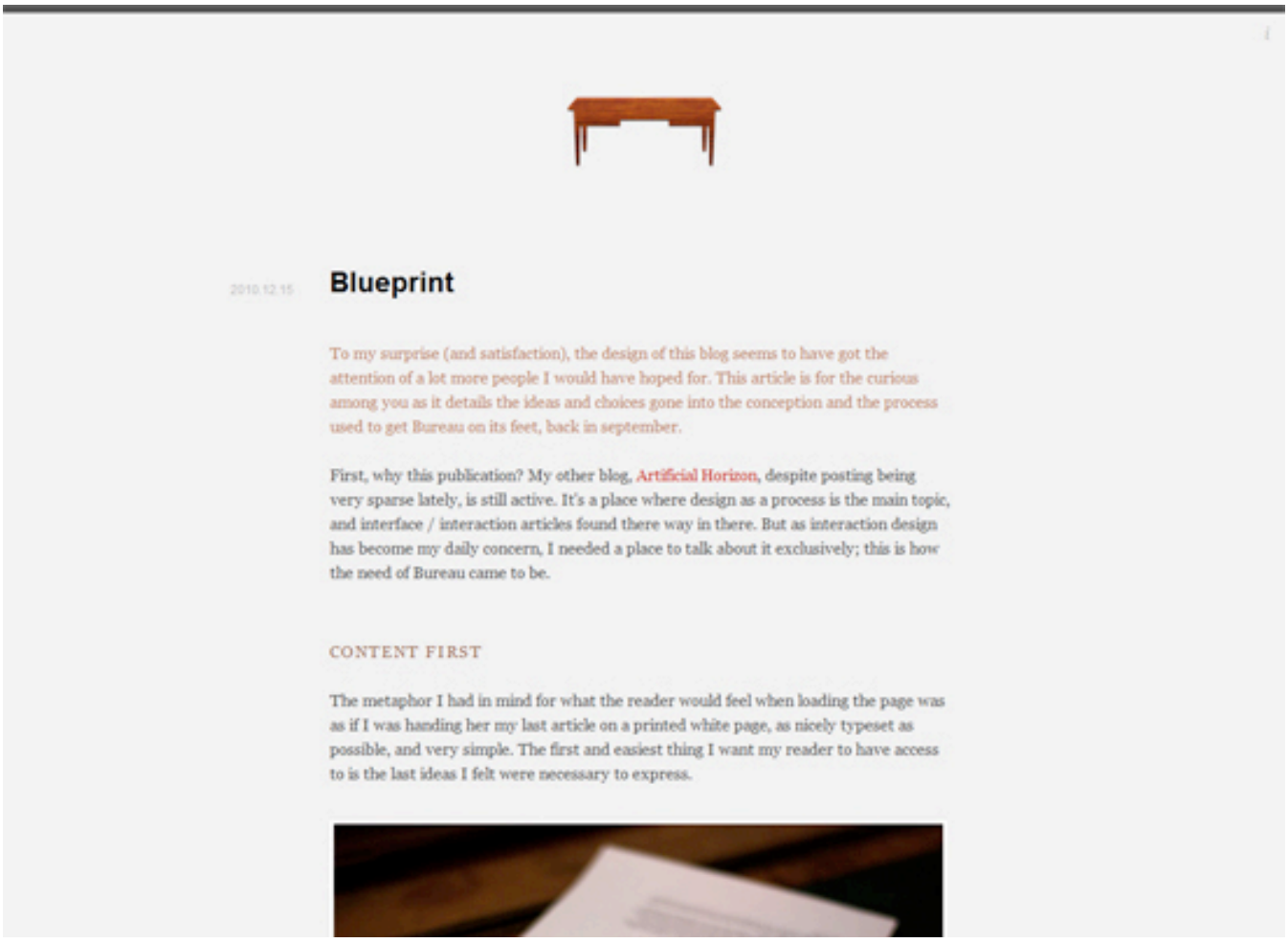
This layout does not change at all; no content is dropped or rearranged; and the text size does not change either. Instead, this design keeps its original form, no matter what the change in horizontal and vertical space. Instead, it automatically resizes the header image and the images for the navigation. The white space, margins and padding are also flexible, giving more room as the design expands and shrinks.




UNSTOPPABLE ROBOT NINJA
THE ONLINE WEB-THING OF ETHAN MARCOTTE.
ABOUT JOURNAL HOME RESPECT CONTACT
The Last Three Entries:
SEE ALL ›
Responsive images
Since striking out on my own, much of my time's been

Bureau

This is perhaps the simplest example of a responsive Web design in this showcase, but also one of the most versatile. The only piece in the layout that changes with the browser width is the blog post's date, which moves above the post's title or to the side, depending on how much horizontal space is available. Beyond this, the only thing that changes is the width of the content area and the margin space on the left and right. Everything is centered, so a sense of balance is maintained whatever the screen or browser width. Because of this design's simplicity, switching between browser and screen widths is quick and easy.



2010.12.15

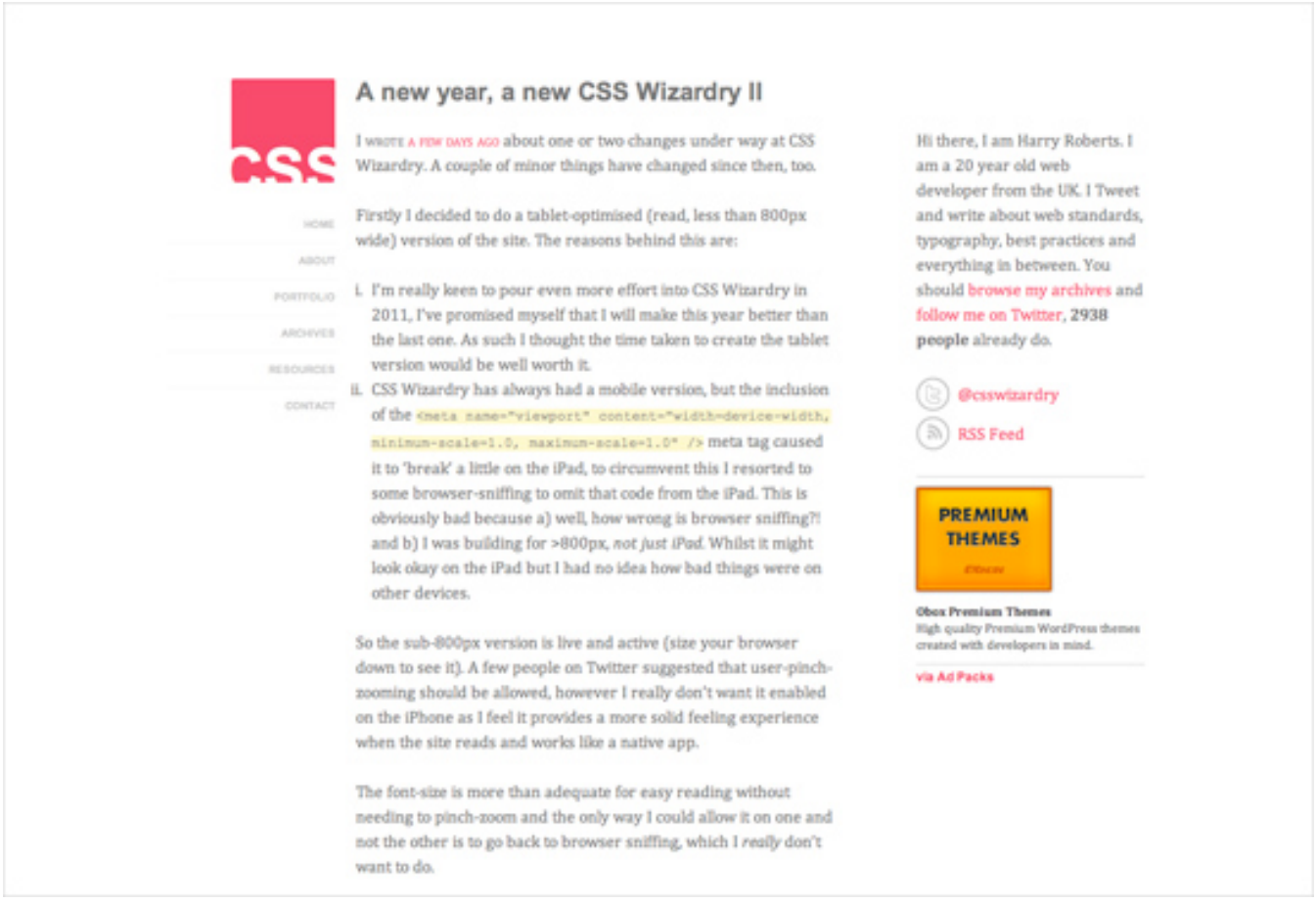
Blueprint

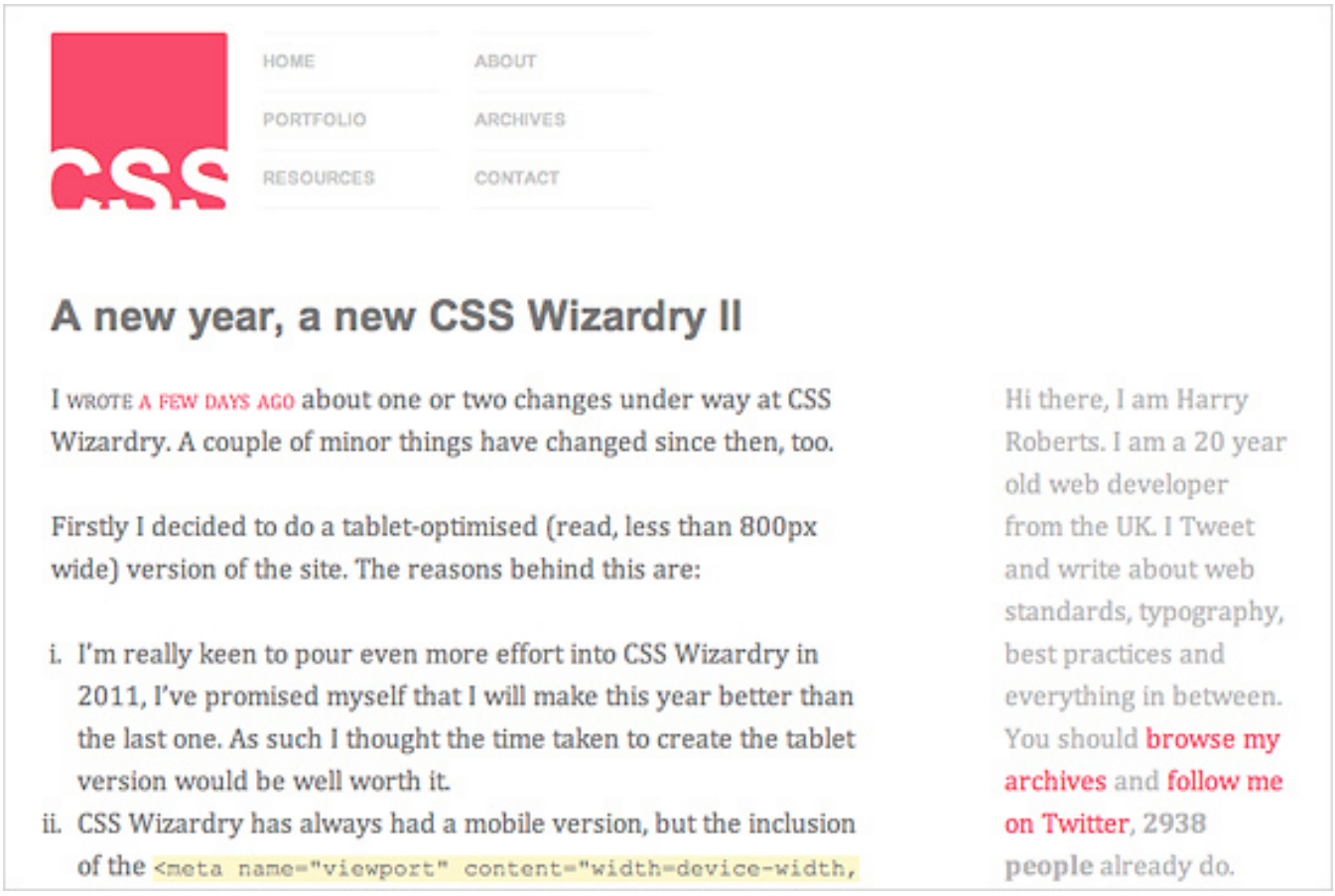
To my surprise (and satisfaction), the design of this blog seems to have got the attention of a lot more people I would have hoped for. This article is for the curious among you as it details the ideas and choices gone into the conception and the process used to get Bureau on its feet, back in september.

First, why this publication? My other blog, Artificial Horizon, despite posting being very sparse lately, is still active. It's a place where design as a process is

CSS Wizardry

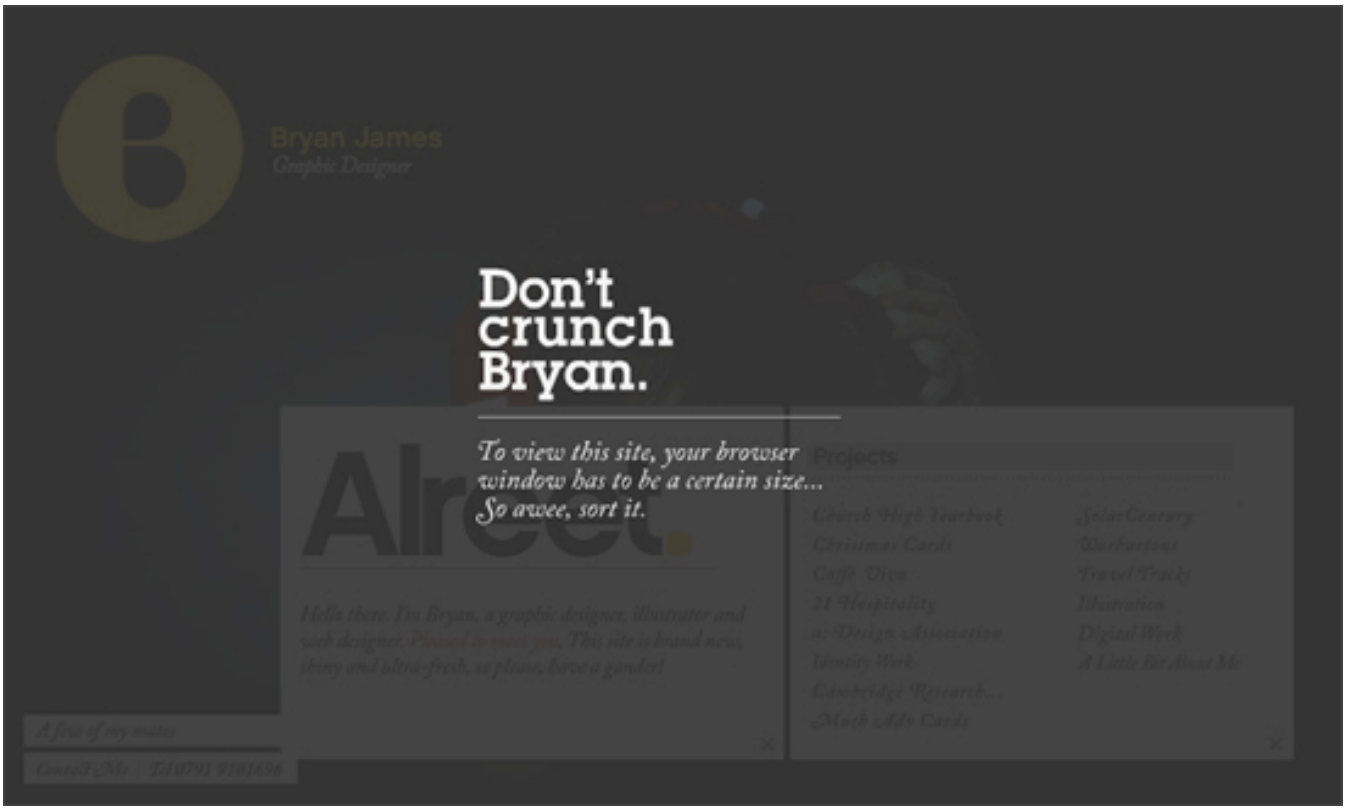
Harry Roberts shows that responsive design can also have quite humble uses. If the user has a large viewport, the website displays three columns with a navigation menu floating on the left. For users with a viewport between 481px and 800px, a narrow version is displayed: the navigation jumps to the top of the site leaving the area for the content column and the sidebar. Finally, the iPhone view displays the sidebar under the content area. Harry also wrote a detailed article about the CSS styles he added to the stylesheet in his article “Media queries, handier than you think“. A nice example of how a couple of simple CSS adjustments can improve the website’s appearance across various devices.




HOME
ABOUT
PORTFOLIO
ARCHIVES
RESOURCES
CONTACT
A new year, a new CSS Wizardry II
I WROTE A FEW DAYS AGO about one or two changes under way at CSS Wizardry. A couple of minor things have changed since then, too.
Firstly I decided to do a tablet-optimised (read, less than 800px wide) version of the site. The reasons behind this are:
i. I'm really keen to pour even more effort into CSS Wizardry in 2011, I've promised myself that I will make this year better than the last one. As such I thought the time taken to create the tablet version would be well worth it.
ii. CSS Wizardry has always had a mobile version, but the inclusion of the <meta name="viewport" content="width=device-width,
Hi there, I am Harry Roberts. I am a 20 year old web developer from the UK. I Tweet and write about web standards, typography, best practices and everything in between. You should browse my archives and follow me on Twitter, 2938 people already do.

Bryan James

This last design by Bryan James shows that responsive Web design need not apply only to static HTML and CSS websites. Done in Flash, this one features a full-sized background image and is flexible up to a certain width and height. As a result of the design style, on screens that are too small, the background image gets mostly hidden and the content can become illegible and squished. Instead of just letting it be, though, a message pops up informing the user that the screen is too small to adequately view the website. It then prompts the user to switch to a bigger screen. One can discuss if the design solution is good or bad in terms of usability, but the example shows that Flash websites can respond to user's viewport, too.


Bryan James
Graphic Designer
Don't crunch Bryan.
To view this site, your browser window has to be a certain size...
So awee, sort it.
Projects

Conclusion

We are indeed entering a new age of Web design and development. Far too many options are available now, and there will be far too many in the future to continue adjusting and creating custom solutions for each screen size, device and advancement in technology. We should rather start a new era today: creating websites that are future-ready right now. Understanding how to make a design responsive to the user doesn't require too much learning, and it can definitely be a lot less stressful and more productive than learning how to design and code properly for every single device available.

Responsive Web design and the techniques discussed above are not the final answer to the ever-changing mobile world. Responsive Web design is a mere concept that when implemented correctly can *improve* the user experience, but not completely solve it for every user, device and platform. We will need to constantly work with new devices, resolutions and technologies to continually improve the user experience as technology evolves in the coming years.

Besides saving us from frustration, responsive Web design is also best for the user. Every custom solution makes for a better user experience. With responsive Web design, we can create custom solutions for a wider range of users, on a wider range of devices. A website can be tailored as well for someone on an old laptop or device as it can for the vast majority of people on the trendiest gadgets around, and likewise as much for the few users who own the most advanced gadgets now and in the years to come. Responsive Web design creates a great custom experience for everyone. As Web designers, we all strive for that every day on every project anyway, right?

Progressive And Responsive Navigation

Jeremy Hixon

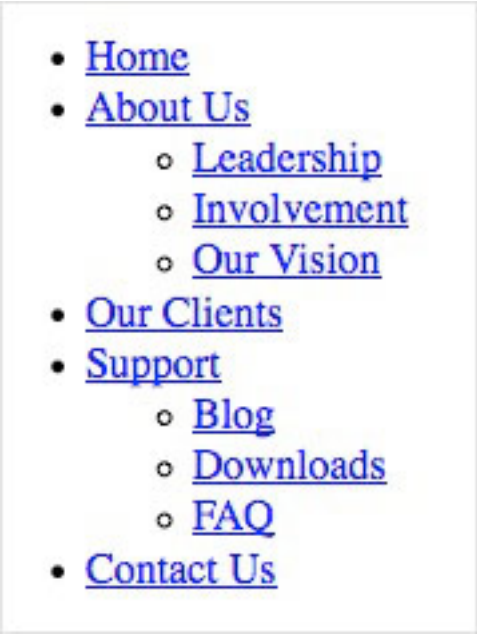
Developing for the Web can be a difficult yet rewarding job. Given the number of browsers across the number of platforms, it can sometimes be a bit overwhelming. But if we start coding with a little forethought and apply the principles of progressive enhancement from the beginning and apply some responsive practices at the end, we can easily accommodate for less-capable browsers and reward those with modern browsers in both desktop and mobile environments.

A Common Structure

Below is the HTML structure of a navigation menu created by WordPress. This unordered list is pretty common for content management systems and hand-coded websites alike. This will be the basis for our work.

Please note: Any ellipses (…) in the snippets below stand in for code that we have already covered. We have used them to shorten the code and highlight the parts that are relevant to that section.

```
<nav class="main-navigation">
   <ul>
      <li><a href="#home">Home</a></li>
      <li>
         <a href="#about">About Us</a>
         <ul class="children">
            <li><a href="#leadership">Leadership</a></li>
            <li><a href="#involvement">Involvement</a></li>
            <li><a href="#vision">Our Vision</a></li>
         </ul>
      </li>
      <li><a href="#clients">Our Clients</a></li>
      <li>
         <a href="#support">Support</a>
         <ul class="children">
            <li><a href="#blog">Blog</a></li>
            <li><a href="#downloads">Downloads</a></li>
            <li><a href="#faq">FAQ</a></li>
         </ul>
      </li>
      <li><a href="#contact">Contact Us</a></li>
   </ul>
</nav>
```



Our navigation, unstyled.

Our Tools

- CSS Reset
- HTML5 elements
- LESS CSS
- jQuery

CSS RESET

Resetting our CSS styles is where we'll start. Browsers have different default styles for the elements we'll be using, so understanding this and getting all of the elements to look the same is important. In this example, since we're using an unordered list, there will be default left padding, top and bottom margins, and a `list-style`. You can either deal with these individually or, if you're project will include more than just this navigation, use a reset to clear all of the styles and start fresh. I prefer Eric Meyer's Reset CSS, but there are a few others to choose from, listed below. Whichever you choose, make sure it accounts for the new HTML5 elements.

- Yahoo! YUI CSS Reset
- HTML5 Doctor CSS Reset
- Normalize.css (HTML5-ready alternative to CSS resets)

HTML5 AND CSS3 ELEMENTS

We'll be wrapping the menu in HTML5's `nav` element, which is one HTML5 feature that we should be using right now. If you need more good reasons to use HTML5 in your daily work, such as accessibility, then read "Top 10 Reasons to Use HTML5 Right Now" over at Codrops.

CSS3 will give our menu the progressive feel we're looking for. We can use nifty effects such as linear gradients, text and box shadows and rounded corners, while providing a reasonable appearance for browsers that are dragging their feet. You could also consider using something like CSS3 Pie in the process. This will give those lagging browsers most of the functionality they lack to display your CSS3 properties.

LESS CSS

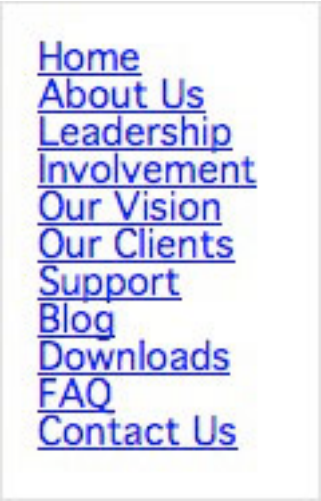
To make our CSS more efficient, we'll use LESS along with a class file to ease the difficulty of dealing with all of those browser prefixes. Other options, such as Sass and Compass, do effectively the same thing and might better fit your particular development environment. If you're interested in learning more about LESS and how it compares to Sass, check out another article of mine, "An Introduction to LESS, and Comparison to Sass."

JQUERY

To make our navigation a little friendlier in small browsers, such as those on mobile devices, we'll use JavaScript. Essentially, we will gather all of the elements in our navigation and reorganize them into a **`select`** form element. Then, when the user selects an option from the list, they will navigate to that page. Interaction with a **`select`** element is one of the easiest and best ways to handle navigation in a small window. The practice is pretty common as well, so the learning curve for users will be gentler.

Getting Started

After applying a reset, we get something like the following. You can see that the margins, padding and list styles have been cleared.



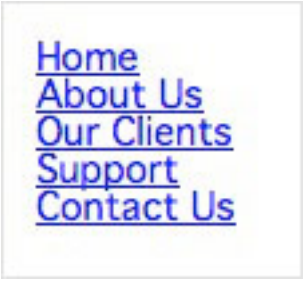
Reset navigation

CHILD-LEVEL MENUS

For now, the child-level menus will only get in the way. The best thing to do is remove them from the equation and add them back in when it's time to style them. To achieve this, we will apply **`position: relative`** to all of the list elements, and move the children off screen until they are needed.

```
.main-navigation {
   li {
      position: relative;
   }
   .children {
      left: -999em;
      position: absolute;
   }
}
```

Applying **`left: -999em; position: absolute;`** will move the children to the left of the screen by a significant margin. This method is preferable to just using **`display: none`** because it is more accessible to screen readers.



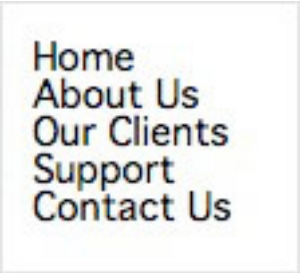
Unstyled without children

COMMON NAVIGATION STYLES

Every navigation menu will probably have links in it. But these links are not like the links we see in the main body of the page, which are blue, underlined and distinguishable from the surrounding text. Rather, links in the navigation will stand alone, and their function will be obvious. That being said, the links in a `nav` element will probably have a few features of their own that distinguish them from typical anchor tags.

```
nav {
    a {
        color: inherit;
        display: block;
        text-decoration: none;
    }
}
```

Thus, a link will inherit the color of the text assigned to the parent element, in this case **`nav`**. It will be set to display as a block-level element, because we want the clickable area to be large and we do not want underlining (because that would just look funny).



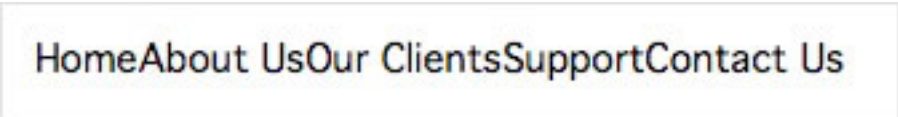
Navigation with more functional links

Please note: `color: inherit` is not supported in IE 6 or 7. If you need to support those browsers, then you will need to explicitly set the color that you want.

LINING UP

Getting the menu in line calls for the use of floats. Initially, we'll float all of the elements in the **`nav`** element to the left. Later, we'll undo this property for the child-level menus, along with a lot of the other styles that we'll set along the way.

```
.main-navigation {
   ul, li, a {
      float: left;
   }
   …
}
```



Inline navigation

Because every element in the **`nav`** element is now floated, the element itself will collapse as though it were empty. There are a few ways to deal with this. One is to also float the **`nav`** element itself, which will expand it to wrap

around its contents. If need be, you can set it to **`width: 100%`** to fill any remaining space to the right. Or you could use Nicolas Gallagher's "micro" clearfix solution, which essentially adds **`clear: both`** just before the closing of the **`nav`** element.

```
/* For modern browsers */
.cf:before,
.cf:after {
    content:"";
    display:table;
}
.cf:after {
    clear:both;
}
/* For IE 6/7 (trigger hasLayout) */
.cf {
    zoom:1;
}
```

Because we're using LESS for our CSS, applying the clearfix to our **`main-navigation`** class without modifying the markup is very easy.

```
.main-navigation {
   .cf;
   …
}
```

We'll see more of this, as well as a description of how this works, in the section titled "Rounded Corners and Gradients" below.

Styling

All righty. By now, you're probably as tired of looking at an unstyled menu as I am. To start, we'll build what looks like a block wall, and then chisel a nice menu out of it. We won't serve the block wall to antiquated browsers, but it's a good start anyway.

BACKGROUND COLOR AND BORDERS

```
.main-navigation {
   font-size: 0.8em;

   ul, li, a {
      …
   }
   ul {
      background: #eee;
      border: 1px solid #ddd;
   }
   li {
      …
      border-right: 1px solid #ddd;
   }
   li:last-child {
      border-right: none;
   }
   a {
      height: 35px;
      line-height: 35px;
      margin: 3px;
      padding: 0 15px;
   }
   .children {
      …
   }
```

```
}
```

In the code above, the text was just too big, so we shrunk it with **`font-size: 0.8em`**. This property is set on the **`main-navigation`** class, so it applies throughout the navigation. The top-level unordered list has a **`border: 1px solid #ddd`** property to break it out from the page. Each list item element is given a **`border-right: 1px solid #ddd;`** to separate it from each other. The **`li:last-child`** selector targets the last list item element in the unordered list, removing the right border because no item follows it.

The links within the navigation are given a background color and some left and right padding to add distinction and increase their clickable area. We're fixing the **`height`** and **`line-height`**, instead of using top and bottom padding, so that we can predict more accurately where the child-level menus will be positioned relative to their shared parent list item.

Navigation resembling a block wall

ROUNDED CORNERS AND GRADIENTS

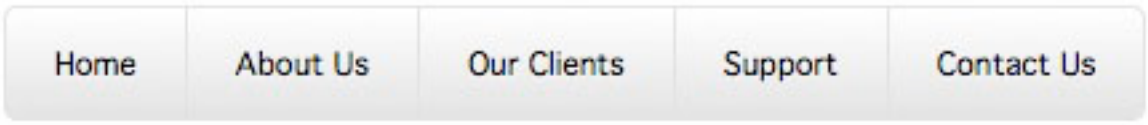
```
.main-navigation {
   …
   text-shadow: 0 1px 1px #fff;
   ul {
      border: 1px solid #ddd;
      .border-radius();
      .linear-gradient();
   }
   …
```

```
}
.border-radius (@radius: 5px) {
   border-radius: @radius;
}
.linear-gradient (@start: #fff, @end: #ddd, @percent: 100%) {
   background: @start; /* Old */
   background: -moz-linear-gradient(top,  @start 0%, @end
@percent); /* FF3.6+ */
   background: -webkit-gradient(linear, left top, left bottom,
color-stop(0%,@start), color-stop(@percent,@end)); /* Chrome,
Safari 4+ */
   background: -webkit-linear-gradient(top,  @start 0%,@end
@percent); /* Chrome 10+, Safari 5.1+ */
   background: -o-linear-gradient(top,  @start 0%,@end
@percent); /* Opera 11.10+ */
   background: -ms-linear-gradient(top,  @start 0%,@end
@percent); /* IE 10+ */
   background: linear-gradient(top,  @start 0%,@end
@percent); /* W3C */
}
```

Above, we have created two new classes, **`border-radius`** and **`linear-gradient`.**

The **`border-radius`** class is actually what LESS developers refer to as a parametric mixin. Essentially, it's like a class, but you can pass variables to it in case the default value isn't exactly what you want. In this case, if 5 pixels isn't what you want, you could reference the mixin as **`.border-radius(10px)`**, and then it would use **`10px`** instead of the original **`5px`.** With the **`border-radius`** property, you could also pass it something like **`.border-radius(5px 0 0 5px)`**, and it would apply the 5-pixel rounding to only the top-left and bottom-left corners. For more information and possibilities on **`border-radius`**, see "Border-Radius: Create Rounded Corners With CSS" at CSS3.info.

Another parametric mixin is **`linear-gradient`**. But with LESS, you can add classes to other selectors and it will apply the same styles—thus negating the need to modify the markup just to add another class (and, by extension, its styles) to an element. Both of the classes I've created cover the possibilities of browser syntax. Currently, Webkit has two different syntaxes, because for some reason the browser makers decided to ignore the specification when first implementing it and made up their own syntax. With Chrome 10 and Safari 5.1, they went back to the specification, joining the other browsers, and made things a little easier for us. However, if you still care about the previous versions, you'll need to add their crazy syntax as well. We've also added a white **`text-shadow`** to the text in the navigation to give it a slightly beveled look.



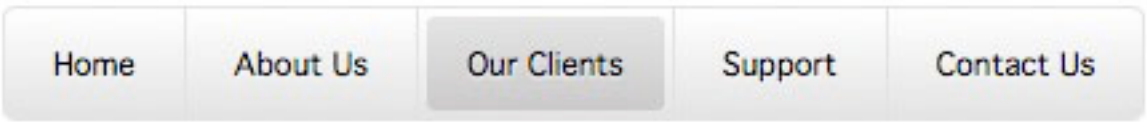
With the two classes applied, you can see the slight gradient and the rounded corners.

Some browsers do not support CSS3 gradients. Yes, I'm looking at you, Internet Explorer 6, 7, 8 and 9. If you want to use something other than the filter syntax for gradients, you'll have to wait for version 10. In the meantime, either you could use the filter syntax for IE (see the "For Internet Explorer" section of "Cross-Browser CSS Gradient") and put them in an IE-specific style sheet, or you could use an image gradient. You could also just leave them without the gradient, but that's not the point here.

PARENT-LEVEL HOVER STATES

```
.main-navigation {
   …
   li:hover {
      a {
         .linear-gradient(#dfdfdf, #c0bebe, 100%);
      }
      .children {
         …
         a {
            background: none;
         }
      }
   }
   …
}
```

The code above will trigger the hover state for anchor elements when the user hovers over their parent list item, rather than hovering over the anchors themselves. This way is preferable so that the anchor element maintains its hover state when the user is mousing over the child-level menu as well. Doing it this way does, however, create the need to reset the background color of anchor elements within the child-level menus. That's the part you see within the **`children`** selector.



Hovering over the parent-level links

Displaying the Children

Bringing the children back onto the screen is easy enough. But before we get carried away, we need to clear out a few styles that are applied to all unordered lists, list items and anchors.

```
.main-navigation {
   …
   .children {
      background: #fff;
      left: -999em;
      position: absolute;
      li, a {
         float: none;
      }
      li {
         border-right: none;
      }
   }
}
```

The code above changes the background of the child-level menu to white, instead of the light gradient that we used in the parent-level menu. The next couple of lines remove the left float from the list items and anchors. We've also gotten rid of the right border that separates the list items in the parent-level menu.

THE HOVERING BOX

```
.main-navigation {
   …
   .children {
      background: #fff;
      .box-shadow();
      left: -999em;
```

```
        margin-left: -65px;
        position: absolute;
        top: 30px;
        width: 130px;
        …
    }
}
…
.box-shadow (@x: 0, @y: 5px, @blur: 5px, @spread: -5px,
@color: #000) {
    -moz-box-shadow: @x @y @blur @spread @color;
    -webkit-box-shadow: @x @y @blur @spread @color;
    box-shadow: @x @y @blur @spread @color;
}
…
```

We've added another parametric mixin to the equation. This one produces the box shadow, with all of its parameters as variables, and with the browser prefixes. We've borrowed the styles from **`.children`** to make the box appear to hover over the parent menu. To center the child underneath the parent element, we've set the left position to 50%, and set the left margin to the negative value of half the width of the child. In this case, the child level menu is set to 130 pixels wide, so we've set the left margin to -65 pixels.

Navigation with the child reset to hover style

CHILD-LEVEL HOVERS

```
.main-navigation {
   …
   .children {
      …
      a {
         .border-radius(3px);
         height: 30px;
         line-height: 30px;
         margin: 3px;
      }
      a:hover {
         background: #dff2ff;
      }
   }
}
```

We're using the parametric mixin that we created for the **`border-radius`** for the links in the children as well. Adding a 3-pixel margin and 3-pixel border radius to all of the anchor elements within the child menu will accent the 5-pixel border radius of the menu well. We've also adjusted the height and line height a little, because they just seemed too high. Finally, we gave the list items a nice soft-blue background color on hover.

Navigation with child menus and their hover state

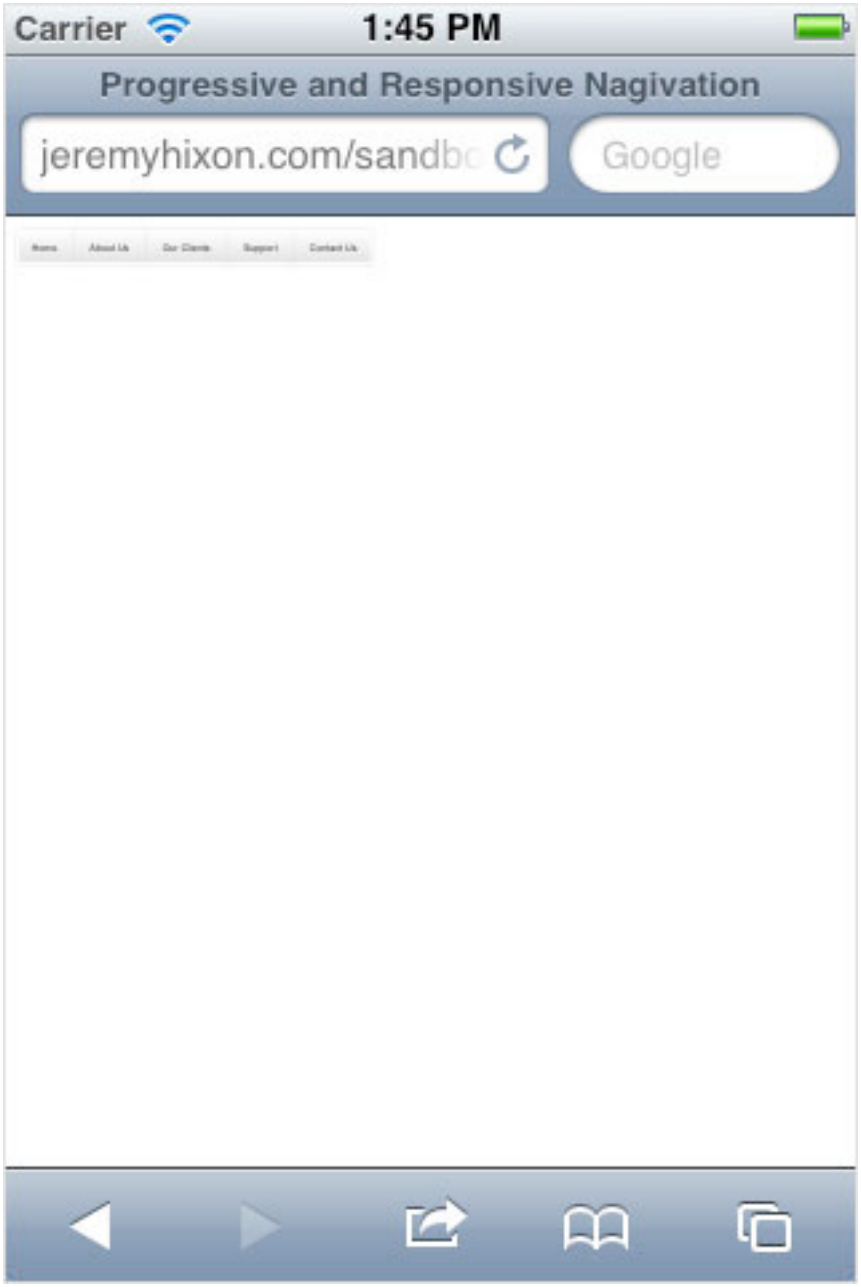
Responding to Mobile Browsers and Size Constraints

A lot of screen sizes and browsers are out there. The iPhone has had two resolutions. Up to the 3GS model, it was 480 × 320; since the iPhone 4, it has been 960 × 640. Android browsers run from 480 × 320 to 854 × 480. Android also has a lot of browsers to choose from. There are the usual Firefox and Opera, as well as a ton of browsers by small start-ups. You can get Opera for the iPhone, but since you can't make it the default browser, you're pretty much stuck with Safari. Given this variety, we'll have to make some adjustments if we want our navigation to be useful on all devices and in all browsers.

FITTING THE CONTENT

Accomplishing this part is easy, but doing it will probably require adjusting our styles. But that's why we're here, isn't it?

Currently, when we open the navigation demo in iOS, it looks like this:

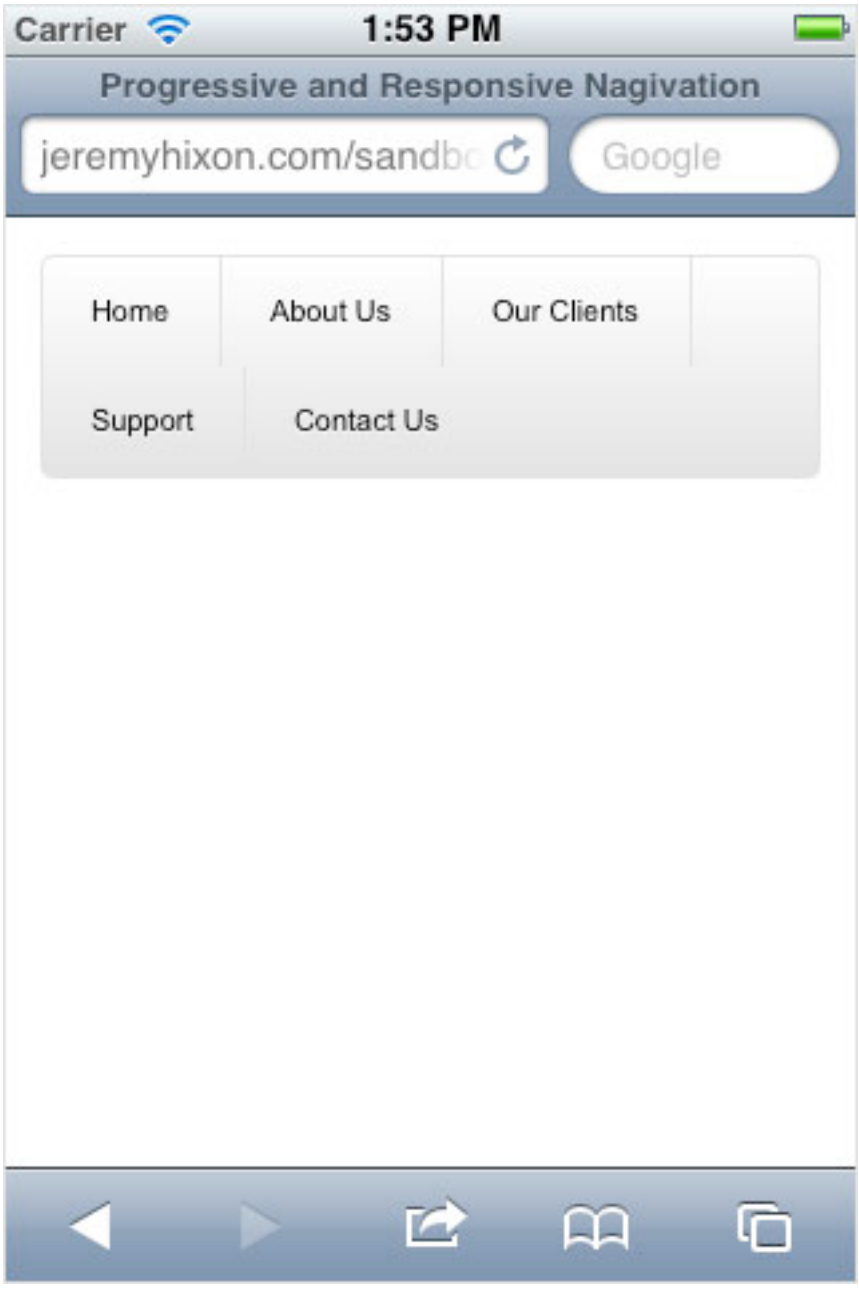


Original navigation in iOS

This might not look too bad on a giant screen, and it might even be usable on the iPad, but you would struggle to use it on a phone. Zooming in might make it easier, but that's not ideal. Optimizing for the device is preferable, and forcing the browser to use the available space is simple.

```
<meta name="viewport" content="width=device-width">
```

This alone makes a huge difference in the way the browser renders the page. And while the menu is not the prettiest it's ever been, it is a lot more functional.



Navigation on iOS with the viewport adjusted

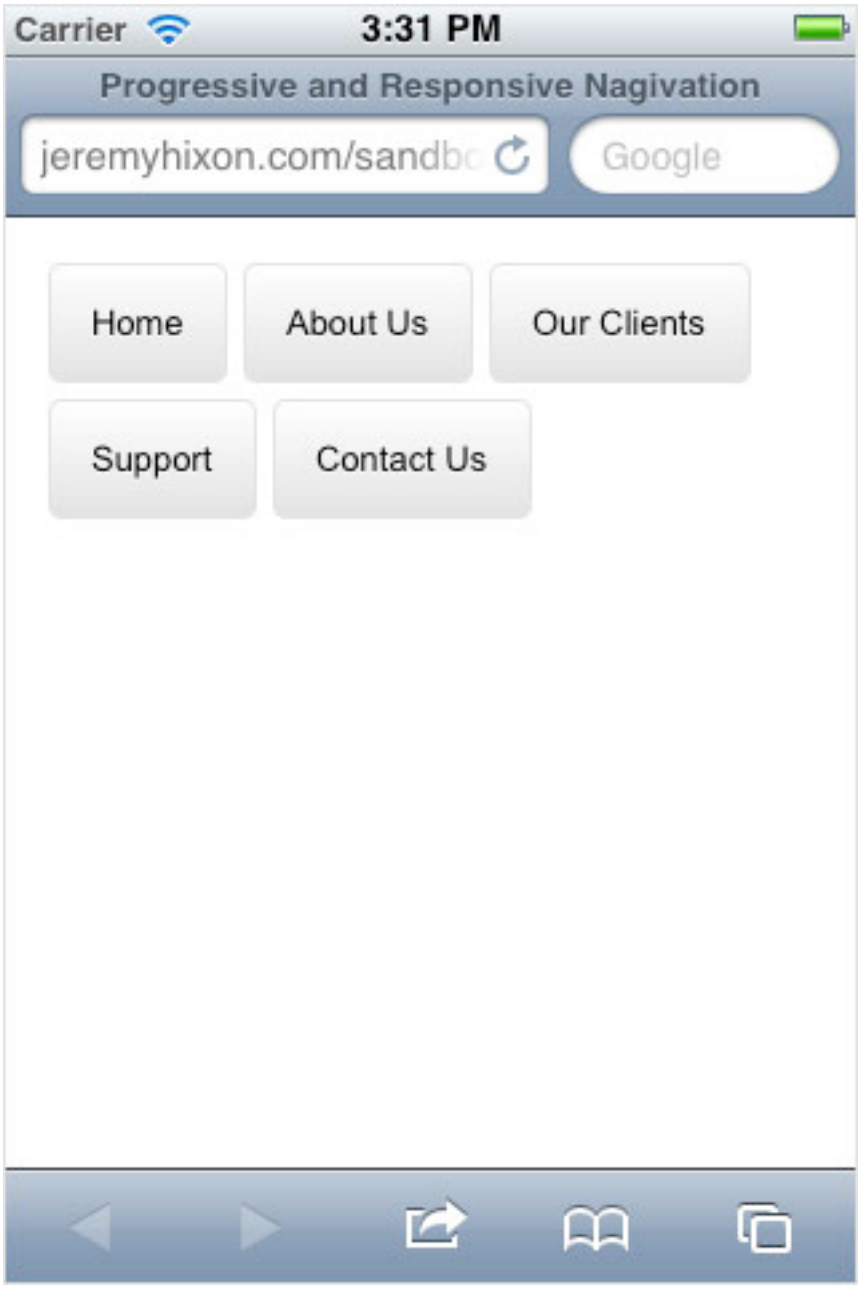
MEDIA QUERIES

We can use media queries to adjust the styles based on the media in the browser. In this case, we'll use the width of the page to change the look and feel of the navigation to make it more suitable to the available space. In this case, we'll make the menu items more button-like.

```
@media only screen and (max-width: 640px) {
   .main-navigation {
      ul {
         border: none;
         background: none;
         .border-radius(0);
      }
      li {
         border-right: none;
      }
      a {
         border: 1px solid #ddd;
         .border-radius();
         font-size: 1.2em;
         height: auto;
         .linear-gradient();
         line-height: 1em;
         padding: 15px;
      }
   }
}
```

In the code above, we've used a media query to target situations in which the user is only looking at a screen and in which the width of the window is a maximum of 640 pixels. In this scenario, we've removed the border, background and border radius from the unordered list, and applied those styles to the anchors themselves. We've also increased the font size of the

anchors, cleared the height and line height, and adjusted the padding of the links to increase the clickable area.



Navigation adjusted for mobile

As you can see, the links look much friendlier in a mobile browser. They are, however, only half functional, because touch devices don't have a hover state. This means that if you have child-level menus, as we do here, you'll

have to figure out a way to display them as well. You could replace the hover state with a touch event of some kind, or expand the children out onto the page. That would greatly increase the size of the navigation, though. The following method might be best.

REPLACING THE MENU IN MOBILE BROWSERS WITH JAVASCRIPT

```
$(function() {
   /* Get the window's width, and check whether it is narrower
than 480 pixels */
   var windowWidth = $(window).width();
   if (windowWidth <= 480) {
      /* Clone our navigation */
      var mainNavigation = $('nav.main-navigation').clone();
      /* Replace unordered list with a "select" element to be
populated with options, and create a variable to select our
new empty option menu */
      $('nav.main-navigation').html('<select class="menu"></
select>');
      var selectMenu = $('select.menu');
      /* Navigate our nav clone for information needed to
populate options */
      $
(mainNavigation).children('ul').children('li').each(function()
{
         /* Get top-level link and text */
         var href = $(this).children('a').attr('href');
         var text = $(this).children('a').text();
         /* Append this option to our "select" */
         $(selectMenu).append('<option value="'+href+'">'+text
+'</option>');
         /* Check for "children" and navigate for more options
if they exist */
         if ($(this).children('ul').length > 0) {
            $
(this).children('ul').children('li').each(function() {
```

```
                /* Get child-level link and text */
                var href2 = $(this).children('a').attr('href');
                var text2 = $(this).children('a').text();
                /* Append this option to our "select" */
                $(selectMenu).append('<option
value="'+href2+'">--- '+text2+'</option>');
            });
        }
    });
  }
  /* When our select menu is changed, change the window
location to match the value of the selected option. */
  $(selectMenu).change(function() {
    location = this.options[this.selectedIndex].value;
  });
});
```

To summarize, first we're checking whether the window is less than or equal to 480 pixels. To ensure an accurate reading on mobile devices, you can use a meta tag to scale the viewport accordingly:

```
<meta name="viewport" content="width=device-width">
```

We populate the first variable, **`windowWidth`**, with the value of the window's width as defined by the given device. We can use this value to then check whether the width is narrower than a particular value. We've chosen 480 pixels here because, while we might want to use media queries to adjust the menu below 640 pixels, at a certain point the viewport would be just too small to justify the menu taking up all that space.

We then use jQuery to create a clone of our menu that we can later crawl to create our options. After we've done that, it's safe to replace the unordered list with the `select` element that we'll be using and then select it with jQuery.

In the largest part of the code, we're crawling through the clone of our navigation. The selector used, **`$(mainNavigation).children('ul').children('li')`**, ensures that we go through only the uppermost list elements first. This is key to creating the nested appearance of the select menu. With it, we select the "direct" child-level unordered list elements and then their "direct" child-level list elements, and then parse through them.

Inside each of these "direct" descendants, we get the value of the `href` attribute and the text of the link, and we store them in variables to be inserted in their respective options. This is implemented by appending **`<option value="'+href+'">'+text+'&kt;/option>`** to our new select list.

While we're in the top-level list item elements, we can check whether any child-level menus need to be parsed. The statement **`if ($(this).children('ul').length > 0)`** checks whether the count of the selector is greater than 0. If it is, that means child-level items need to be added. We can use that same selector, with a slight addition, to go through these elements and add them to our select list, **`$(this).children('ul').children('li').each().`**

The same parsing method applies to these elements, although they use different variables to store the values of the anchor tags, so as not to create conflicts. We have also prefixed text to the menu labels at this level, **`---`** , to differentiate them from the other items.

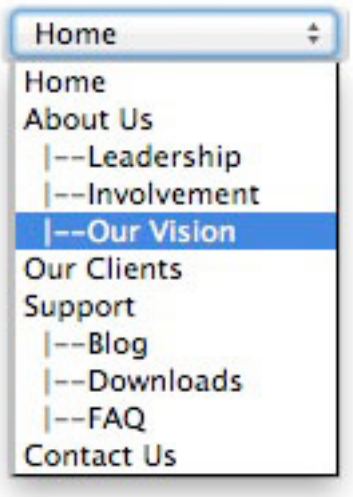
Parsing through the menu in this method (nested) will create the parent-child relationship you would expect.

After the menu is created, a little more JavaScript will enable the select list to serve as navigation.

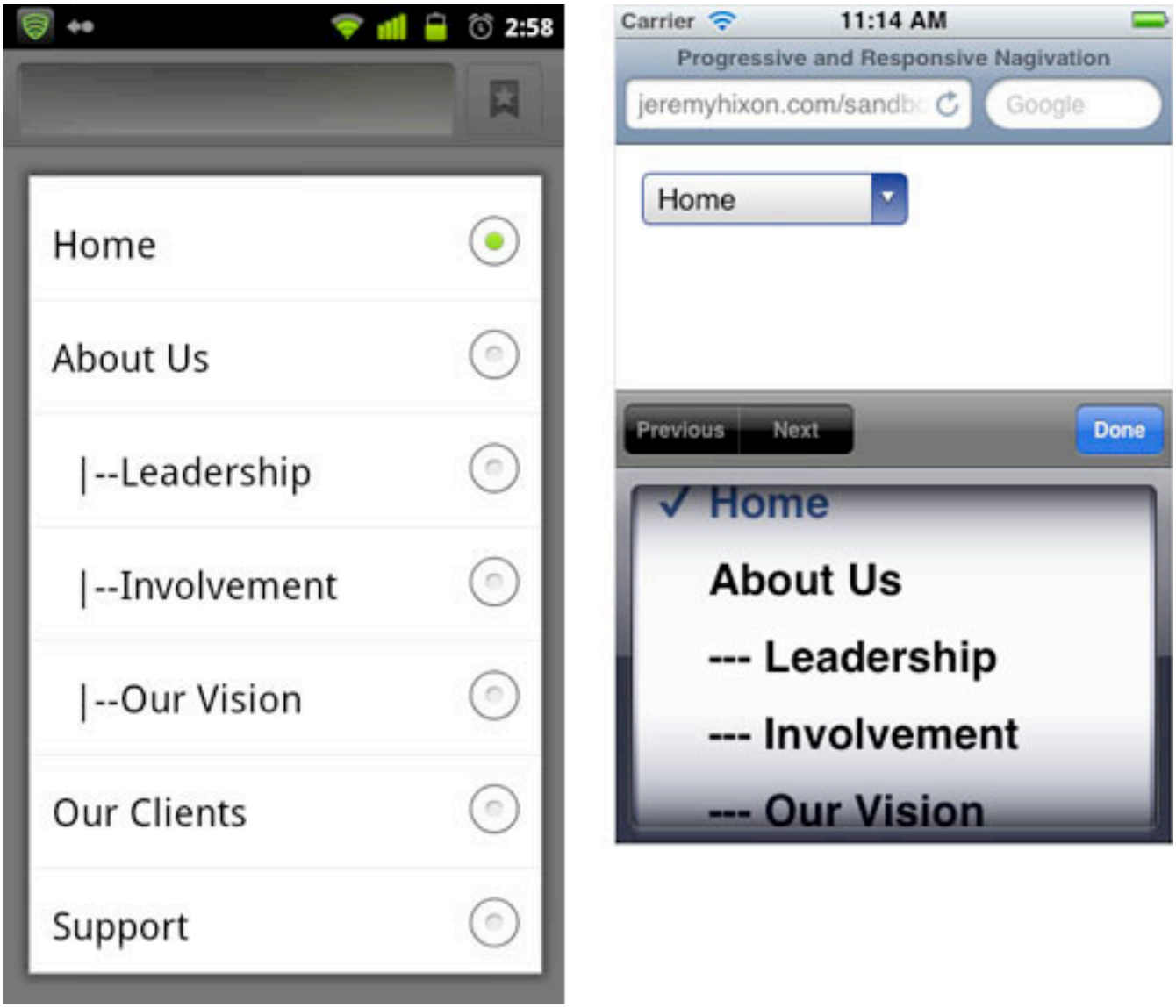
```
$(selectMenu).change(function() {
   location = this.options[this.selectedIndex].value;
});
```

When the select menu is changed, a new option is selected, and the window location is changed to reflect the value of the option. That value comes from the **`href`** of the original anchor element.

The result is like so:



The select menu in a desktop browser



The select menu in Android and iPhone browsers

Given the increased clickable area of the native controls, the select menu is obviously much more user-friendly on mobile.

Techniques For Gracefully Degrading Media Queries

Lewis Nyman

Media queries are the third pillar in Ethan Marcotte's implementation of responsive design. Without media queries, fluid layouts would struggle to adapt to the array of screen sizes on the hundreds of devices out there. Fluid layouts can appear cramped and unreadable on small mobile devices and too large and chunky on big widescreen displays. Media queries enable us to adapt typography to the size and resolution of the user's device, making it a powerful tool for crafting the perfect reading experience.

CSS3 media queries, which include the browser width variable, are supported by most modern Web browsers. Mobile and desktop browsers that lack support will present a subpar experience to the user unless we step up and take action. I'll outline some of techniques that developers can follow to address this problem.

IT DEPENDS

> *If you're looking for the more honest, truthful answer to pretty much any question on web design and usability, here it is: It depends.*
>
> *– Jeremy Keith*

There is no one-size-fits-all fix. Each project has its own focus, requirements and audience. This article will hopefully help you make the best decision for your project by outlining the advantages and disadvantages of each solution.

Mobile First

Your chosen implementation of media queries will have a big effect on how you tackle this. Mobile-first responsive design is the process of building a mobile layout first, and then progressively modifying the layout as more screen space becomes available. This ensures that only the minimum required files are loaded, and it keeps the mobile solution lightweight. Mobile first has the advantage of providing a nice fallback for mobile devices that don't support media queries, such as older BlackBerrys, Opera Mini and Windows Mobile devices. These devices simply see the most basic layout, with no extra work required of the developer. Ideal!

Technique 1: Do Nothing

Sometimes the lazy approach is the best approach. It keeps your code light and maintainable and reduces any needless processing on the client side. Some old browsers run Javascript like a dog, and old mobile phones struggle to run intensive Javascript. The proprietary non-Webkit browser in most BlackBerrys can take up to eight seconds just to parse the jQuery library. If your project has a long tail of users with low-powered mobile devices, then maybe a mobile-first approach is enough for you.

The elephant in the room is Internet Explorer for the desktop. With a mobile-first solution, large screens will display the content in one column, resulting in a painful reading experience for the user, way past the established maximum for comfort: 50 to 75 characters. It might be worth setting a **`max-width`** property in your main container and then upping that **`max-width`** in a media query.

```
#container {
 _width: 460px; /* Take that, IE6! */
 max-width: 460px;
```

```
}
@media only screen and (width) { /* A quick and simple test
for CSS3 media query support. */
#container {
  max-width: 1200px; /* Add the real maximum width here. */
 }
}
```

DO NOTHING IF…

- Your core audience uses modern smartphones,
- You are required to provide an acceptable experience to a long tail of feature-phone users,
- The desktop is not a big part of your Web strategy.

Example: jQuery Mobile (“Any device that doesn’t support media queries will receive the basic C-grade experience”).

Technique 2: Conditional IE Style Sheets

Surprisingly, in researching this article, I found this to be the most popular technique in use on responsive websites. Instead of polyfilling support for media queries, you simply ~~you simply load an additional style sheet only for Internet Explorer~~ load the same stylesheet that you’re serving up to browsers that do understand media queries for Internet Explorer. For mobile-first approaches, this usually entails loading a basic style sheet that sets up a multi-column layout for large screens. Jeremy Keith documents this approach in detail on his blog. He also adds a condition that doesn’t load the style sheet for mobile versions of IE. Crafty.

It's a simple and effective technique for supporting Internet Explorer on the desktop, and it supports the mobile-first approach because it loads a light and appropriate linear layout for feature phones.

~~On the other hand, this technique could potentially degrade maintainability, requiring you to maintain a style sheet of duplicate content.~~ It also adds another HTTP request for IE users, which should be avoided if possible.

~~I'm surprised that Jeremy Keith advocates this technique. The man who proclaimed on stage that user agent sniffing is "~~the spawn of Satan~~" is using a solution aimed squarely at one browser.~~ Bear in mind that this does not work with browsers that do not support CSS3 media queries. But it can be perfectly acceptable in situations where support for other legacy browsers is not required.

USE CONDITIONAL IE COMMENTS IF…

- You are using a mobile-first workflow;
- Your media queries are simple enough to include in a single style sheet;
- Desktop Internet Explorer requires a multi-column layout, at the expense of speed;
- You do not have to support a long tail of legacy desktop browsers.

Example: Huffduffer, a mobile-first approach with an additional column for screen widths over 480 pixels.

Bonus example: Designing With Data by Five Simple Steps. I love these guys.

Technique 3: Circumvent Media Query Conditions

The Opera Developer Blog published an article in 2007 detailing the safe usage of media queries. It helped pave the way for CSS3 media query usage by presenting research on the correct way to write them, a way that prevents browsers from applying the containing CSS when they do not understand a media query.

… Browsers like IE.

But what if, with a mobile-first approach, that's exactly what we do want? What if we were to write our media queries so that the containing CSS gets applied by IE unconditionally. We could then have our full desktop layout without any additional style sheets to load or maintain.

```
@media screen, all and (min-width: 300px) {
  div {
      background: blue;
  }
}
```

As the blog post states:

> *Now it is no longer the case that IE does not apply the contents of the query. It now doesn't understand the second part (**`all and`**), so it ignores that and happily applies the contents of the query…*

CIRCUMVENT MEDIA QUERY CONDITIONS IF…

- You are only required to support modern smartphones,
- You are building mobile first and require a desktop layout in IE,
- Loading time and maintainability must be kept to a minimum.

Technique 4: Respond.js

Scott Jehl's lightweight polyfill Respond.js offers a leg up for browsers that do not support CSS3 media queries. It can be compressed down to as little as 1 KB, and it parses CSS files fast, without needing any additional libraries.

JavaScript reliance aside, Respond.js appears to be a solid solution for full support of media queries. However, the small file size and speed come at a cost. Respond.js was never meant to be a full-featured solution. Its purpose is to provide the bare minimum for responsive layouts to work.

It supports only the **`min-width`** and **`max-width`** queries, so it's not the right solution if you are looking at using **`device-width, device-height, orientation, aspect-ratio, color, monochrome`** or **`resolution.`** Some good use cases here are not supported, one being the detection of high-resolution devices such as the iPhone 4 and non-color devices such as the Kindle.

Respond.js does not support em-based queries, which makes impossible any decent support for font-size user preferences (even more important on a small screen than on a desktop). Products like Readability and Reeder validate this desire among users to control and refine the reading experience. Em-based media queries will become only more important as we head towards a content-first approach to Web design, so they are worth considering.

There are a lot of small bumps and caveats with Respond.js. I recommend browsing the read-me text and the issue queue before settling on it for you project.

USE RESPOND.JS IF…

- Desktop support is a ~~primary~~ high concern,
- You are querying only the width and height of the browser,
- You don't want to query the width by ems,
- You have no problem with non-JavaScript users seeing an unoptimized page.

Example: Aaron Weyenberg, a desktop-centric website with a basic layout.

Technique 5: CSS3-MediaQueries-js

CSS3-MediaQueries-js picks up where Respond.js leaves off. It supports the full range of media queries introduced in CSS3. The "everything and the kitchen sink" approach is great for a developer's peace of mind. Simply drop it in, and tick the "IE support" box.

But there are significant downsides to consider: this script is not fast; it parses CSS much slower than Respond.js; and it weighs in at a hefty 15 KB.

PRO TIP 1

Let's be responsible and load this file only if the browser doesn't actually support CSS3 media queries. Otherwise, you're wasting good time and data. You can use Yepnope to load the 15 KB file if it detects that media queries are not available.

Here's a modification of a Yepnope function that I wrote for Modernizr's media query test. Yepnope now comes bundled with Modernizr.

```
yepnope({
test : Modernizr.mq('@media only screen and (width)'),
yep  : '',
nope : 'css3-mediaqueries.js',
});
```

If you don't require support for non-IE devices, then replace the Yepnope function with a much lighter conditional comment.

```
<!--[if (lt IE 9) & (!IEMobile)]>
 <script src="css3-mediaqueries.js"></script>
 <![endif]-->
```

PRO TIP 2

If you are building for mobile first, then adding a `min-device-width` condition to the Yepnope query is definitely worthwhile. This will prevent the hefty 15 KB file from loading on small screens that will never use it. Win!

USE CSS3-MEDIAQUERIES-JS IF…

- You are using advanced media queries, beyond simple pixel-width conditions;
- You are happy to take that 15 KB loading hit;
- Your audience doesn't include a long tail of feature-phone users.

Example: Hicksdesign uses complex media queries beyond simple width and height.

In Conclusion

Responsive design is still a new way of thinking. Media Queries are a great tool to enhance the experience of browsing a web site on multiple devices and it's a great idea to consider devices that do no support them. We would be dreaming if we expected an easy solution from day one but at least we have a range of options in front of us that allow us to find the best solution for the problem at hand.

It's important to bear in mind that context is key, a well informed decision will always yield better results instead of quickly choosing the most popular solution.

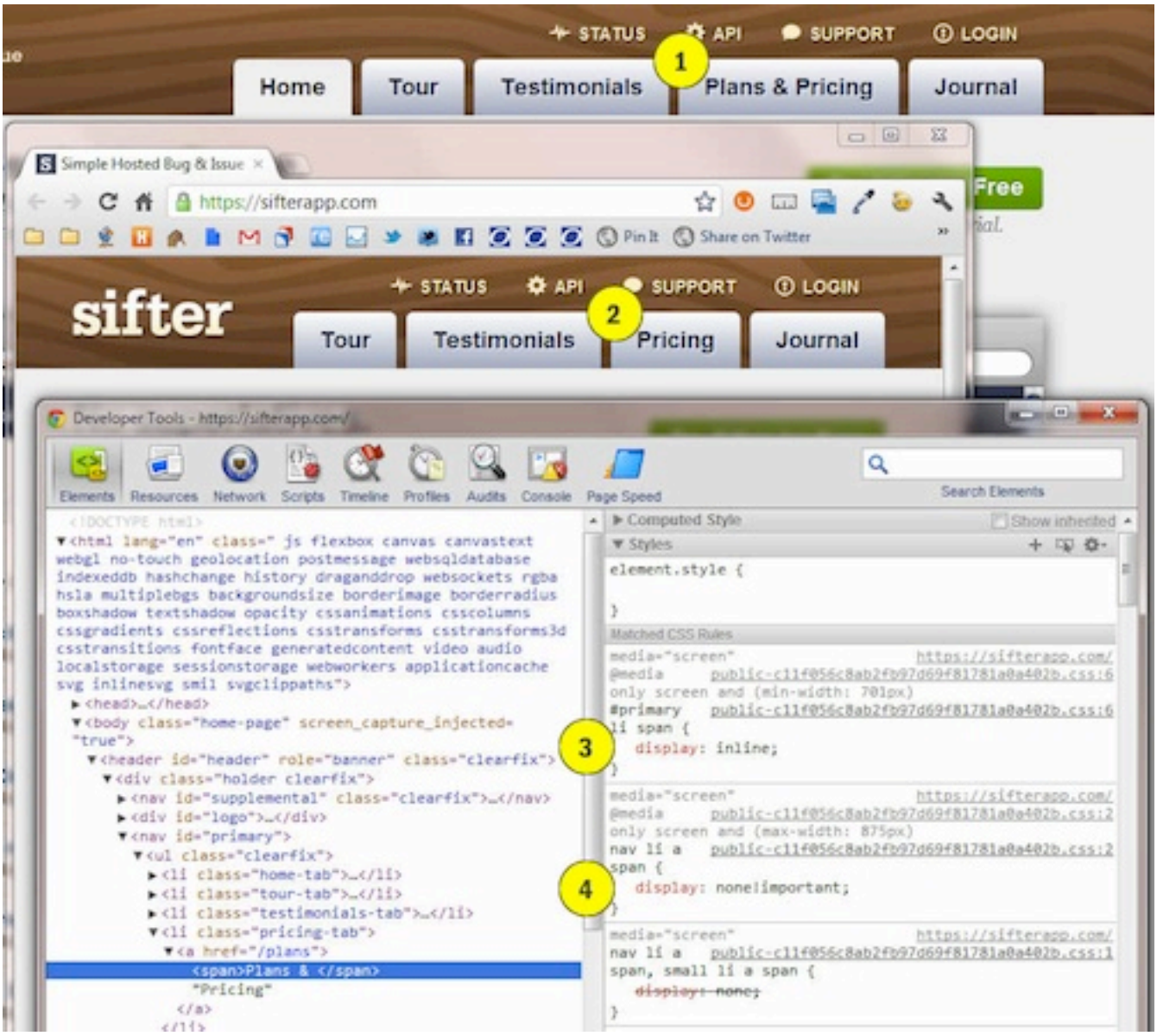
Is There Ever A Justification For Responsive Text?

James Young

Depending on who you follow and what you read, you may have noticed the concept of “responsive text” being discussed in design circles recently. It’s not what you might imagine — resizing and altering the typography to make it easier to read on a range of devices — but rather delivering varying amounts of content to devices based on screen size.

One example of this is an experiment by designer Frankie Roberto. Another is the navigation menu on the website for Sifter App. Roberto and Sifter are using media queries to actually hide and display text based on screen size (i.e. not rewriting or delivering different content based on context — as one would do with mobile-focused copy, for example).

Having looked at how this technique works, I wouldn’t endorse it yet, because its practical value is not clear. Also, describing this as “responsive” could legitimize what is possibly a less than optimal coding technique. Below are screenshots of how it works on the Sifter website:



Altering the tabbed content in the navigation menu at Sifter. Large view.

How Is This Accomplished?

In this example (and in Roberto's demo), you'll notice a couple of things. The screenshots show two versions of the Sifter website at different screen sizes to demonstrate what is happening at two breakpoints.

When you view the website on a large device, the second-last menu tab will show the full label of "Pricing & Plans." On smaller devices (anything up to

the size of a tablet), the label changes to just “Pricing.” This particular example might not be a big deal, but my main concern is that this is being regarded as “responsive text.” It’s not. It’s simply hiding bloated content, and if the content is not important enough to show on smaller screens, then it’s probably not vital at any size.

Does the change in wording mean that information on Sifter’s plans is offered only to users on large devices, or is the “Plans” part redundant? We can assume not, because the tab at all screen sizes links to the **`…/plans`** page. This is potentially confusing for users on small devices: they clicked on “Pricing” but are sent to a page that outlines the plans first.

To show and hide the “Plans &” part of the tab, Sifter’s designer has wrapped the element in a span. For a single menu item, this isn’t the end of the world, but good luck going down the path that Frankie Roberto demonstrates with his paragraphs. I can’t imagine what a nightmare it would be to maintain multiple versions of actual page content and then tie them into breakpoints! (Not to mention our earlier question about whether text that is hidden at certain sizes is redundant in the first place.)

Hopefully, we all know to avoid hiding content with **`display: none ! important;`**. Responsive design is many things; its many little tricks and techniques constitute a wonderful approach to making websites flexible. But hiding elements on a screen in this way should not be one of them.

IT’S JUST A DEMO, THOUGH, RIGHT?

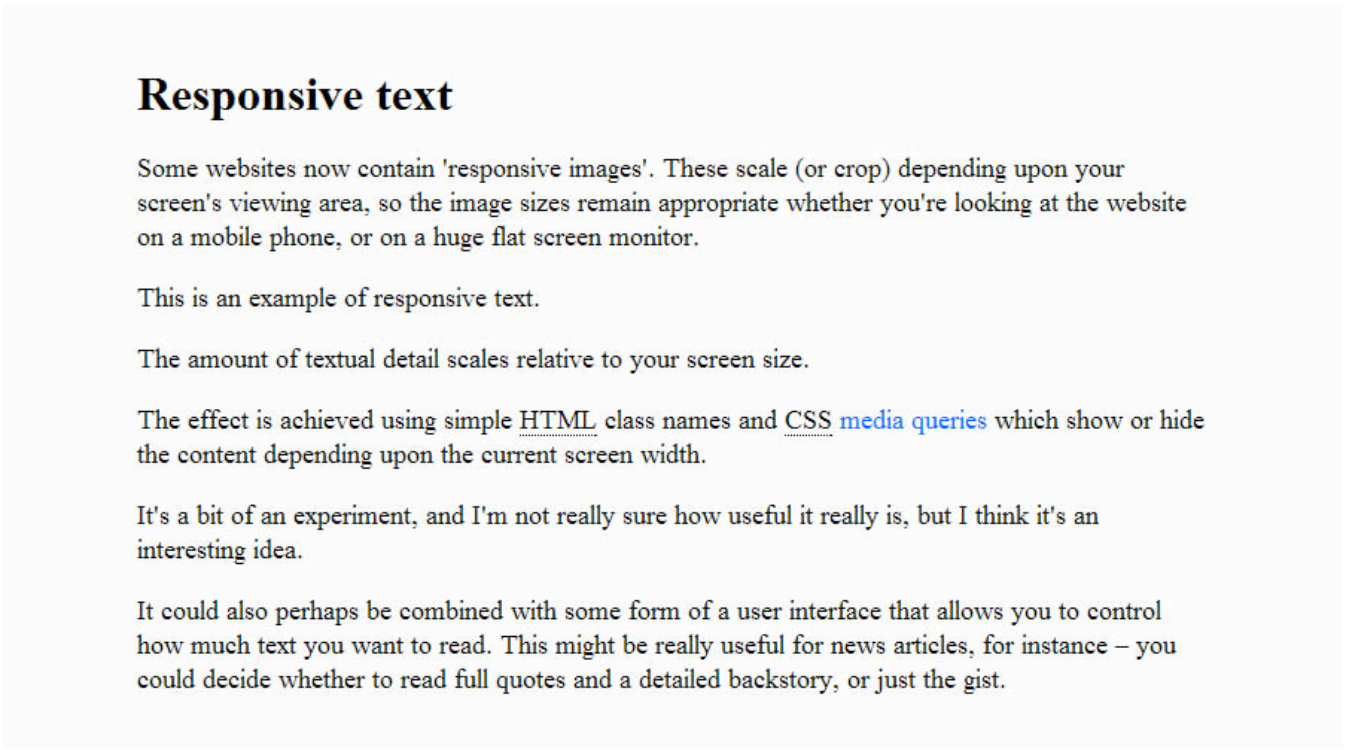
Frankie Roberto’s demo is just that: a demo. He’s clear about that, and he offers a suggestion for a use case. I applaud the effort — everyone should experiment with the Web. The Sifter website is a live website, though — not a demo or proof of concept — and what it has done is being described as “responsive text.”

I'm a huge fan of the concept of "one Web." If you find you have to hide parts of your content on smaller devices, then you might need to refocus your efforts and write less bloated copy or reconsider your wording of page elements.

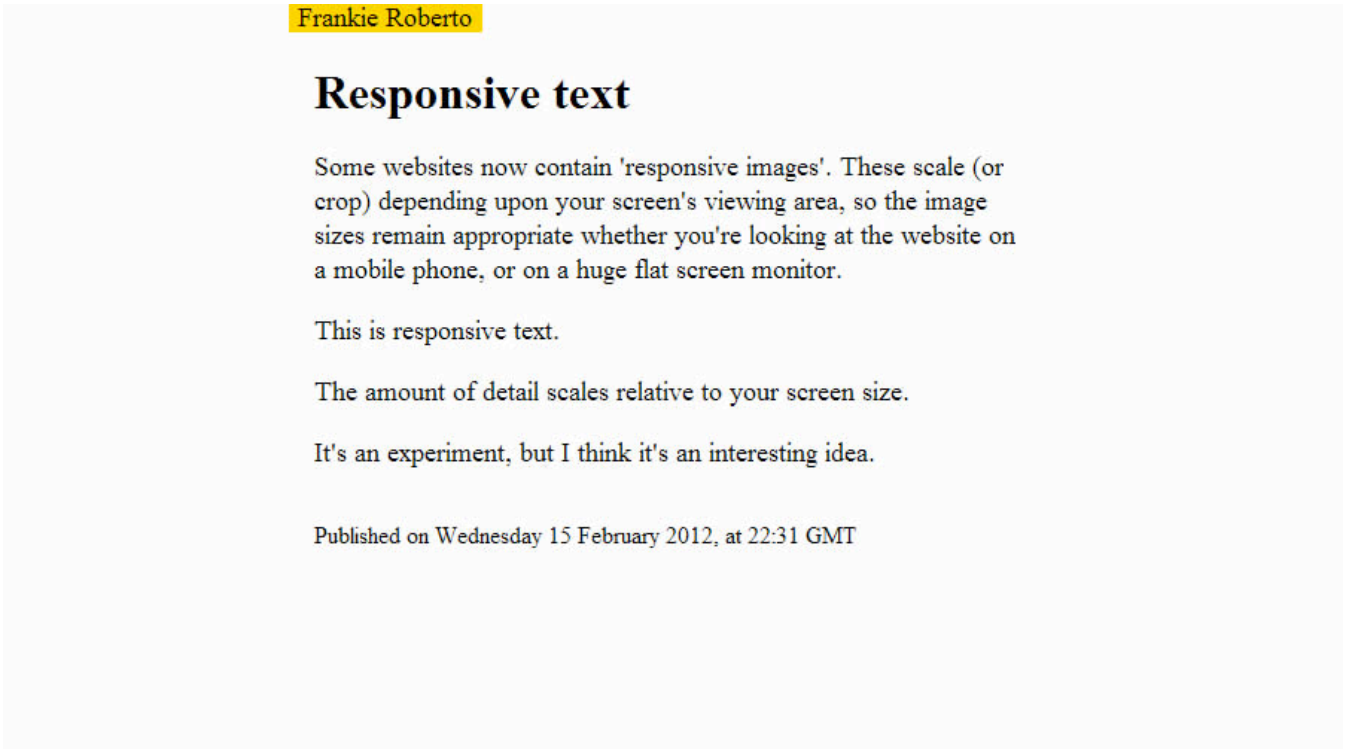
One of the joys of working "mobile-first" is having to maintain a sharp and critical eye in order to cut bloat (a capacity we should always exercise, of course). Responsive text seems to be the polar opposite of this approach. You are practically admitting from the outset that too much text is on the page. You are making the dangerous assumption that someone on a small device wouldn't want to read the hidden text.

Maintenance Problems Will Come Hard And Fast

Frankie Roberto achieves a clever effect in his demo. On a large screen, you see all of the copy. And as the screen shrinks, so does the amount of content (and vice versa, of course).



Roberto's full content, on a desktop screen.



On a smaller screen, the content is reduced.

Achieving this in the demo is easy. A CSS class is applied to the excess paragraphs to hide them.

SOME POTENTIAL PROBLEMS TO CONSIDER

- The copy will have to be highly structured in order for it to be readable when parts of it are hidden on small devices. For example, if a content block has 10 lines, then it should still flow when lines 2, 5, 9 and 10 are hidden on a tablet and lines 2, 4, 5, 9 and 10 are hidden on a phone.
- The writer would need some mechanism in the CMS for flagging the breakpoints in the content. The method for updating content would end up being rather technical as a result.

If the message you are communicating on a small screen is sound, then there is nothing you could really “enhance” it with. Anything you add would simply be bloat.

Are There Any Potential Uses For Responsive Text?

I don't think there are. But I realize this is just my opinion, and I encourage readers and the wider Web community to evaluate it for themselves and disprove me with solid examples.

When discussing this on Twitter the other day, I got interesting responses from a number of fellow designers. Many agreed that whatever you display on a small screen should be your content everywhere, because that is the distillation of your message.

Roberto (@frankieroberto) suggests a potential use case for adaptive news content; for example, showing a summary, a mid-length version or the full article depending on the device. This does sound like a useful way to digest news, but in such a fast-moving environment, I can't imagine copyeditors would thank you for assigning them to write content that adapts so extensively and still makes sense in these different contexts. But it's something to think about.

Stephanie Rieger points out that producing bloat-free content on a big website can be incredibly time-consuming:

> *@welcomebrand @froots101 Discussions with stakeholders reveal last round of copywriting took 6 mths End result, hide text on 'lesser' screen*

No argument there. I'm working on rebuilding a large website, too, and am encountering the same issues. But I'm not sure that hiding content based purely on screen size is wise. If it's not relevant or worth displaying, don't simply hide it: delete or unpublish it.

In Conclusion

My interpretation of the Sifter website and what its designer is trying to achieve may be wrong (this is an opinion piece, remember!). Feel free to tell me as much in the comments below. But from my quick look at the design, code and copy, I won't be embracing responsive text anytime soon, despite it being an interesting experiment and endorsed by some very clever folks.

I struggle to think of a use case that withstands the basic scrutiny that I apply to content for my clients, which is that if all of the content is not good enough to show on all devices, then the amount of content is not optimal. I recognize that this is a harsh stance, so do check out the code and experiments covered here so that you can make up your own mind.

Remember, just because something is "responsive," it might not be best for your project.

How To Use CSS3 Media Queries To Create A Mobile Version Of Your Website

Rachel Andrew

CSS3 continues to both excite and frustrate web designers and developers. We are excited about the possibilities that CSS3 brings, and the problems it will solve, but also frustrated by the lack of support in Internet Explorer 8. This article will demonstrate a technique that uses part of CSS3 that is also unsupported by Internet Explorer 8. However, it doesn't matter as one of the most useful places for this module is somewhere that does have a lot of support — small devices such as the iPhone, and Android devices.

In this article I'll explain how, with a few CSS rules, you can create an iPhone version of your site using CSS3, that will work now. We'll have a look at a very simple example and I'll also discuss the process of adding a small screen device stylesheet to my own site to show how easily we can add stylesheets for mobile devices to existing websites.

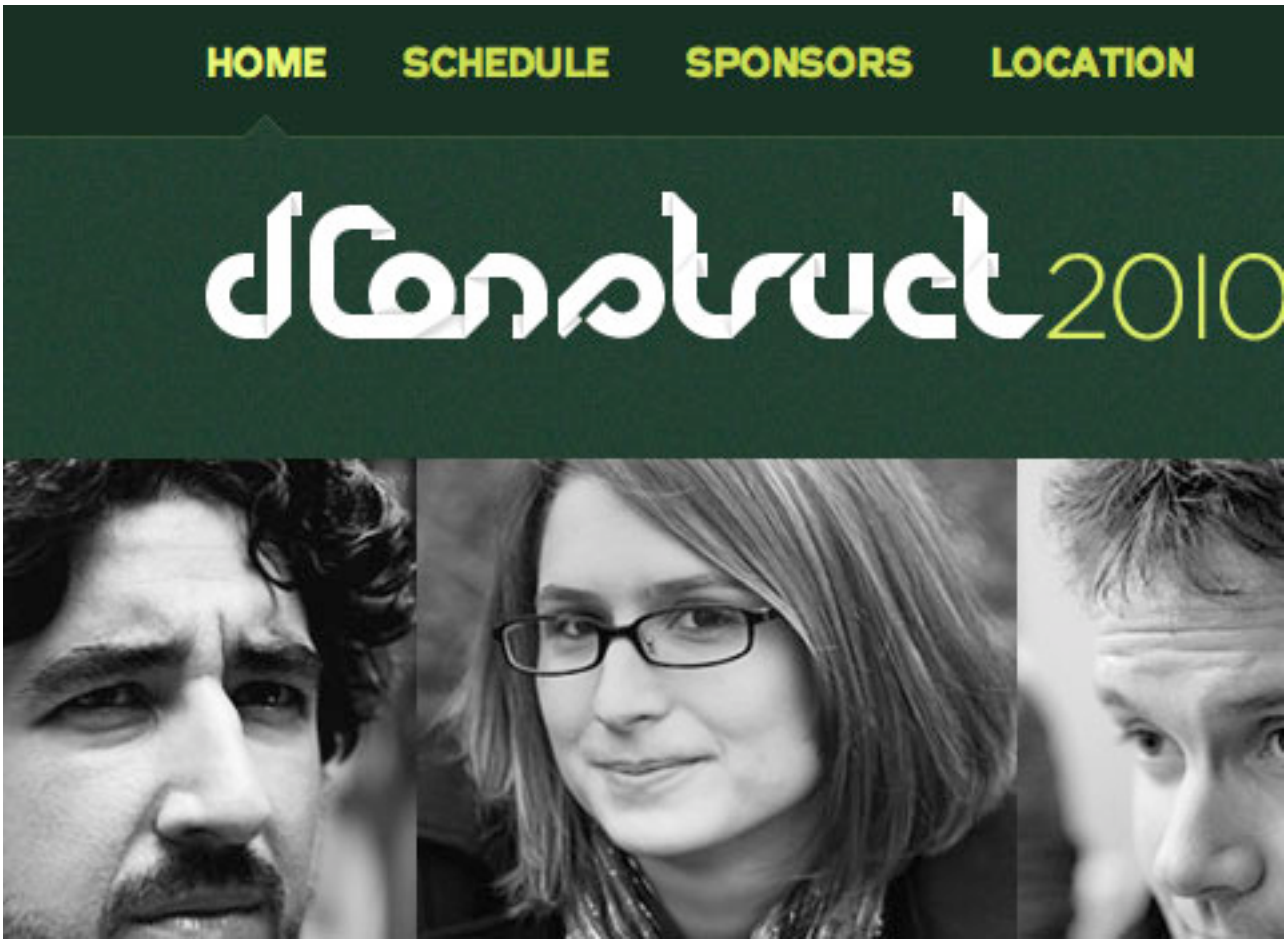
Media Queries

If you have ever created a print stylesheet for a website then you will be familiar with the idea of creating a specific stylesheet to come into play under certain conditions – in the case of a print stylesheet when the page is printed. This functionality was enabled in CSS2 by *media types*. Media Types let you specify a type of media to target, so you could target print, handheld and so on. Unfortunately these media types never gained a lot of support by devices and, other than the print media type, you will rarely see them in use.

The Media Queries in CSS3 take this idea and extend it. Rather than looking for a *type* of device they look at the *capability* of the device, and you can use them to check for all kinds of things. For example:

- width and height (of the browser window)
- device width and height
- orientation – for example is a phone in landscape or portrait mode?
- resolution

If the user has a browser that supports media queries then we can write CSS specifically for certain situations. For example, detecting that the user has a small device like a smart phone of some description and giving them a specific layout. To see an example of this in practice, the UK web conference dConstruct has just launched their website for the 2010 conference and this uses media queries to great effect.



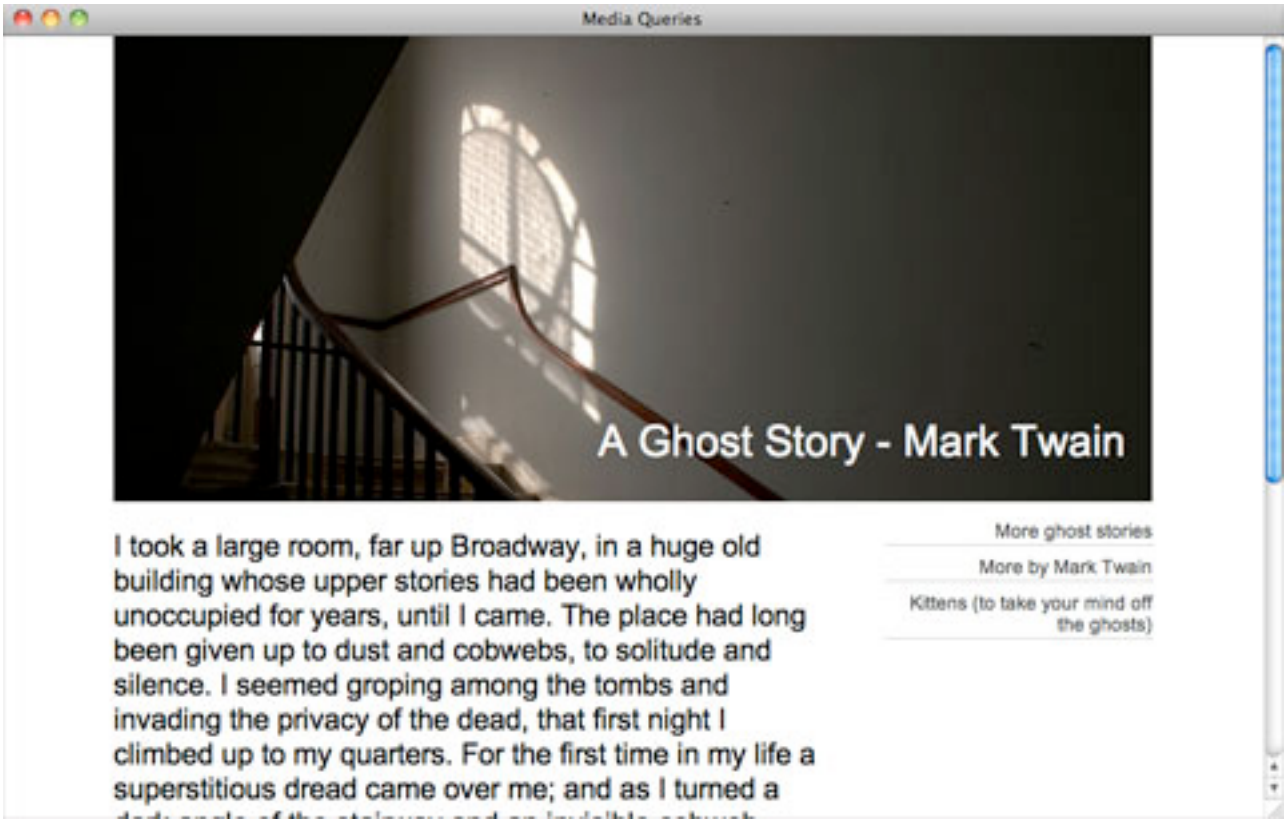
The dConstruct 2010 website in Safari on a desktop computer

The dConstruct 2010 website on an iPhone

You can see from the above example that the site hasn't just been made smaller to fit, but that the content has actually been re-architected to be made more easy to access on the small screen of the device. In addition many people might think of this as being an iPhone layout – but it isn't. It displays in the same way on Opera Mini on my Android based phone – so by using media queries and targeting the device capabilities the dConstruct site caters for all sorts of devices – even ones they might not have thought of!

Using Media Queries to create a stylesheet for phones

To get started we can take a look at a very simple example. The below layout is a very simple two column layout.



A very simple two column layout

To make it easier to read on a phone I have decided to linearize the entire design making it all one column, and also to make the header area much smaller so readers don't need to scroll past the header before getting to any content.

The first way to use media queries is to have the alternate section of CSS right inside your single stylesheet. So to target small devices we can use the following syntax:

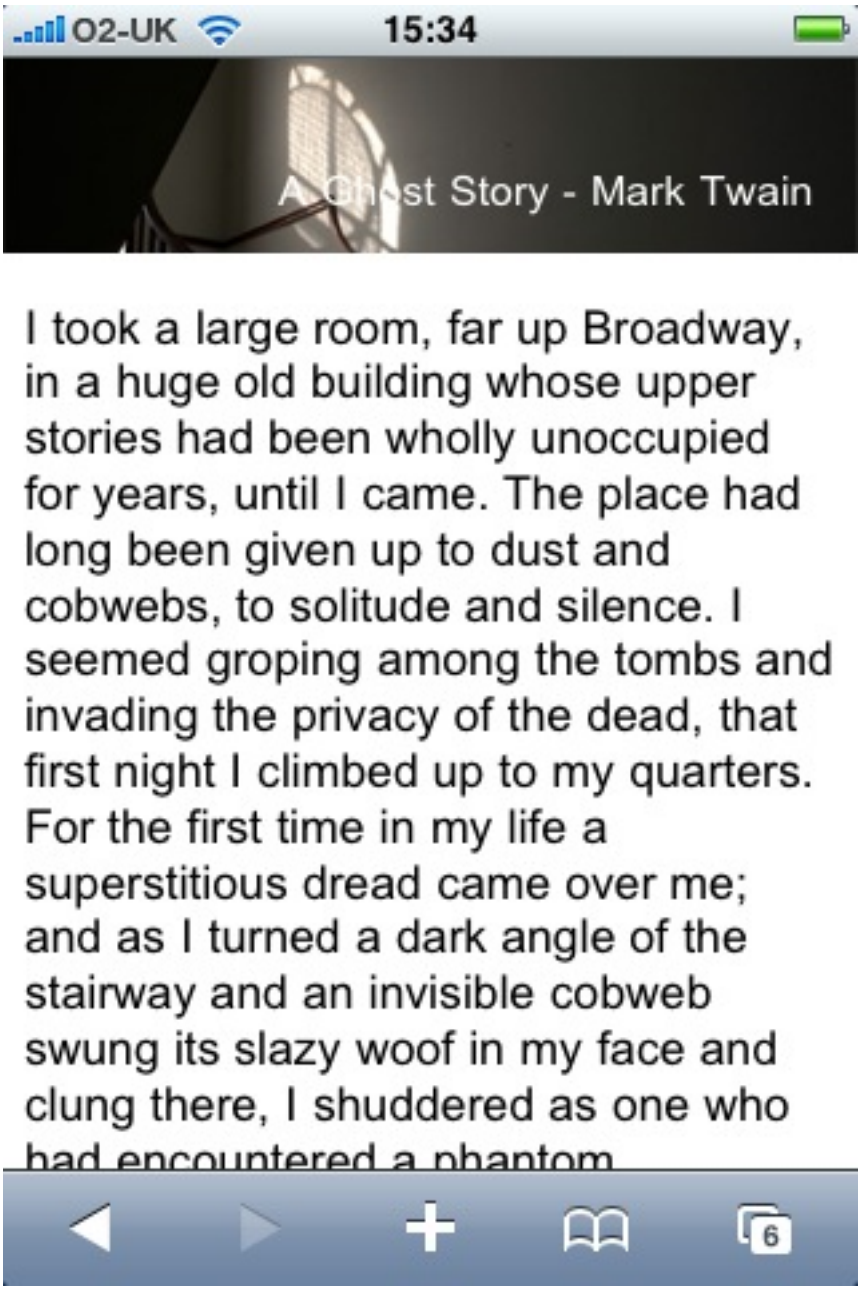
```
@media only screen and (max-device-width: 480px) {
}
```

We can then add our alternate CSS for small screen and width devices inside the curly braces. By using the cascade we can simply overwrite any styles rules we set for desktop browsers earlier in our CSS. As long as this section comes last in your CSS it will overwrite the previous rules. So, to linearize our layout and use a smaller header graphic I can add the following:

```
@media only screen and (max-device-width: 480px) {
  div#wrapper {
      width: 400px;
  }
  div#header {
      background-image: url(media-queries-phone.jpg);
      height: 93px;
      position: relative;
  }
  div#header h1 {
      font-size: 140%;
  }
  #content {
      float: none;
      width: 100%;
  }
```

```
    #navigation {
        float:none;
        width: auto;
    }
}
```

In the code above I am using an alternate background image and reducing the height of the header, then setting the content and navigation to float none and overwriting the width set earlier in the stylesheet. These rules only come into play on a small screen device.



My simple example as displayed on an iPhone

Linking a separate stylesheet using media queries

Adding the specific code for devices inline might be a good way to use media queries if you only need to make a few changes, however if your stylesheet contains a lot of overwriting or you want to completely separate the styles shown to desktop browsers and those used for small screen devices, then linking in a different stylesheet will enable you to keep the CSS separate.

To add a separate stylesheet after your main stylesheet and use the cascade to overwrite the rules, use the following.

```
<link rel="stylesheet" type="text/css" media="only screen and
(max-device-width: 480px)" href="small-device.css" />
```

Testing media queries

If you are the owner of an iPhone, Android device or other device that has a browser which supports media queries you can test your CSS yourself. However you will need to upload the code somewhere in order to view it. What about testing devices you don't own and testing locally?

An excellent site that can help you during the development process is ProtoFluid. This application gives you a form to enter your URL – which can be a local URL – and view the design as if in the browser on an iPhone, iPad or a range of other devices. The screenshot below is the dConstruct site we looked at earlier as seen through the iPhone view on ProtoFluid.

The dConstruct 2010 website in ProtoFluid

You can also enter in your own window size if you have a specific device you want to test for and know the dimensions of it's screen.

To use ProtoFluid you need to slightly modify the media query shown earlier to include max-width as well as max-device-width. This means that the media query also comes into play if the user has a normal desktop browser but using a very tiny window.

```
@media only screen and (max-width: 480px), only screen and
(max-device-width: 480px) {
}
```

After updating your code to the above, just refresh your page in the browser and then drag the window in and you should see the layout change as it hits 480 pixels. The media queries are now reacting when the viewport width hits the value you entered.

You are now all ready to test using ProtoFluid. The real beauty of ProtoFluid is that you can still use tools such as FireBug to tweak your design, something you won't have once on the iPhone. Obviously you should still try and get a look at your layout in as many devices as possible, but ProtoFluid makes development and testing much simpler.

Note that if you don't want your site to switch layout when someone drags their window narrow you can always remove the max-width part of the query before going live, so the effect only happens for people with a small device and not just a small browser window.

Retrofitting an existing site

I have kept the example above very simple in order to demonstrate the technique. However this technique could very easily be used to retrofit an existing site with a version for small screen devices. One of the big selling points of using CSS for layout was this ability to provide alternate versions of our design. As an experiment I decided to take my own business website and apply these techniques to the layout.

THE DESKTOP LAYOUT

The website for my business currently has a multi-column layout. The homepage is a little different but in general we have a fixed width 3 column layout. This design is a couple of years old so we didn't consider media queries when building it.

My site in a desktop browser

ADDING THE NEW STYLESHEET

There will be a number of changes that I need to make to linearize this layout so I'm going to add a separate stylesheet using media queries to load this stylesheet after the current stylesheet and only if the max-width is less than 480 pixels.

```
<link rel="stylesheet" type="text/css" media="only screen and
(max-width: 480px), only screen and (max-device-width: 480px)"
href="/assets/css/small-device.css" />
```

To create my new stylesheet I take the default stylesheet for the site and save it as small-device.css. So this starts life as a copy of my main stylesheet. What I am going to do is go through and overwrite certain rules and then delete anything I don't need.

SHRINKING THE HEADER

The first thing I want to do is make the logo fit nicely on screen for small devices. As the logo is a background image this is easy to do as I can load a different logo in this stylesheet. I also have a different background image with a shorter top area over which the logo sits.

```
body {
  background-image: url(/img/small-bg.png);
}
#wrapper {
  width: auto;
  margin: auto;
  text-align: left;
  background-image: url(/img/small-logo.png);
  background-position: left 5px;
  background-repeat: no-repeat;
  min-height: 400px;
}
```

LINEARIZING THE LAYOUT

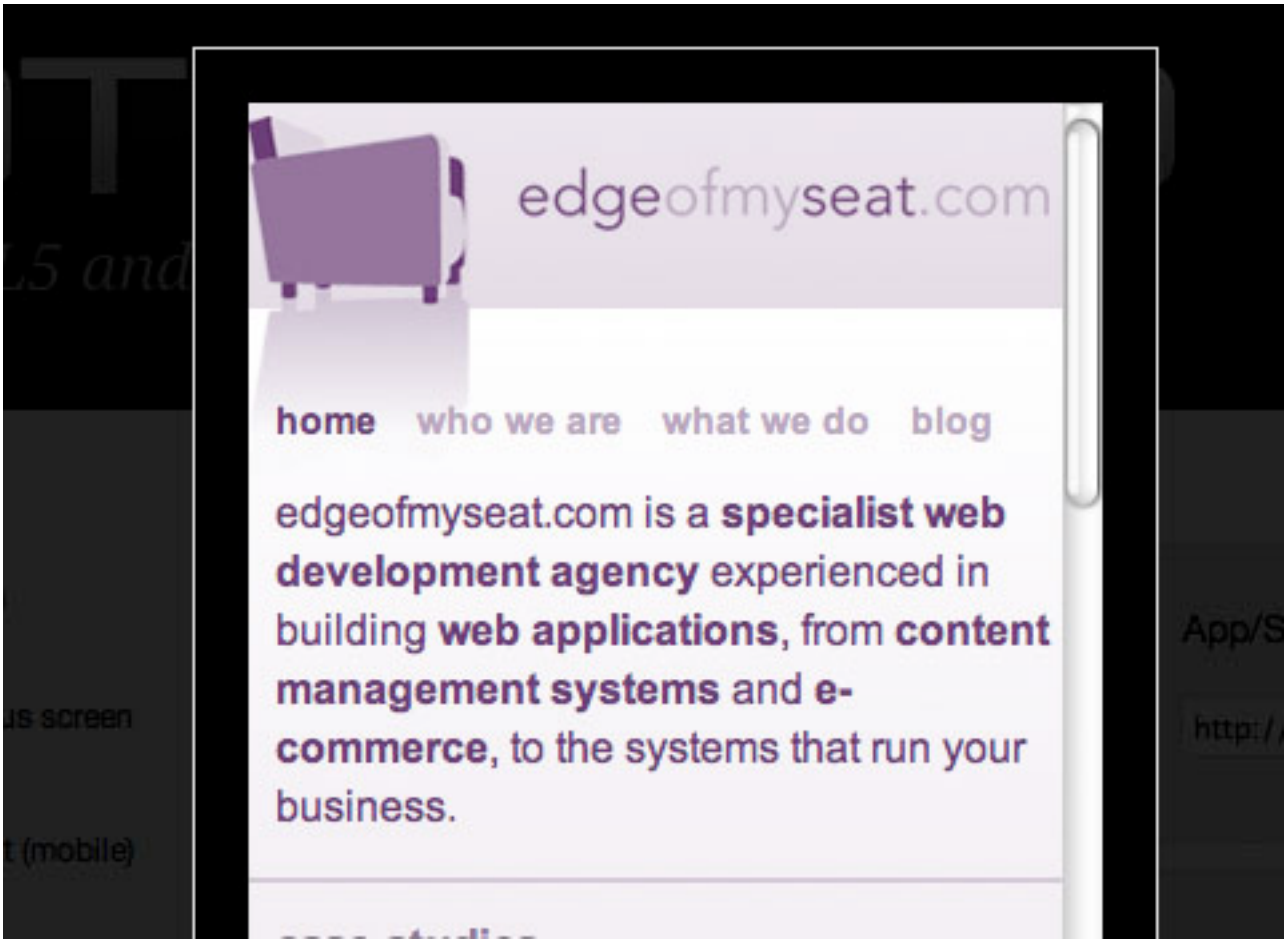
The next main job is to linearize the layout and make it all one column. The desktop layout is created using floats so all I need to do is find the rules that float the columns, set them to float: none and width:auto. This drops all the columns one under another.

```
.article #aside {
  float: none;
```

```
    width: auto;
  }
```

TIDYING UP

Everything after this point is really just a case of looking at the layout in ProtoFluid and tweaking it to give sensible amounts of margin and padding to areas that now are stacked rather than in columns. Being able to use Firebug in ProtoFluid makes this job much easier as my workflow mainly involves playing around using Firebug until I am happy with the effect and then copying that CSS into the stylesheet.



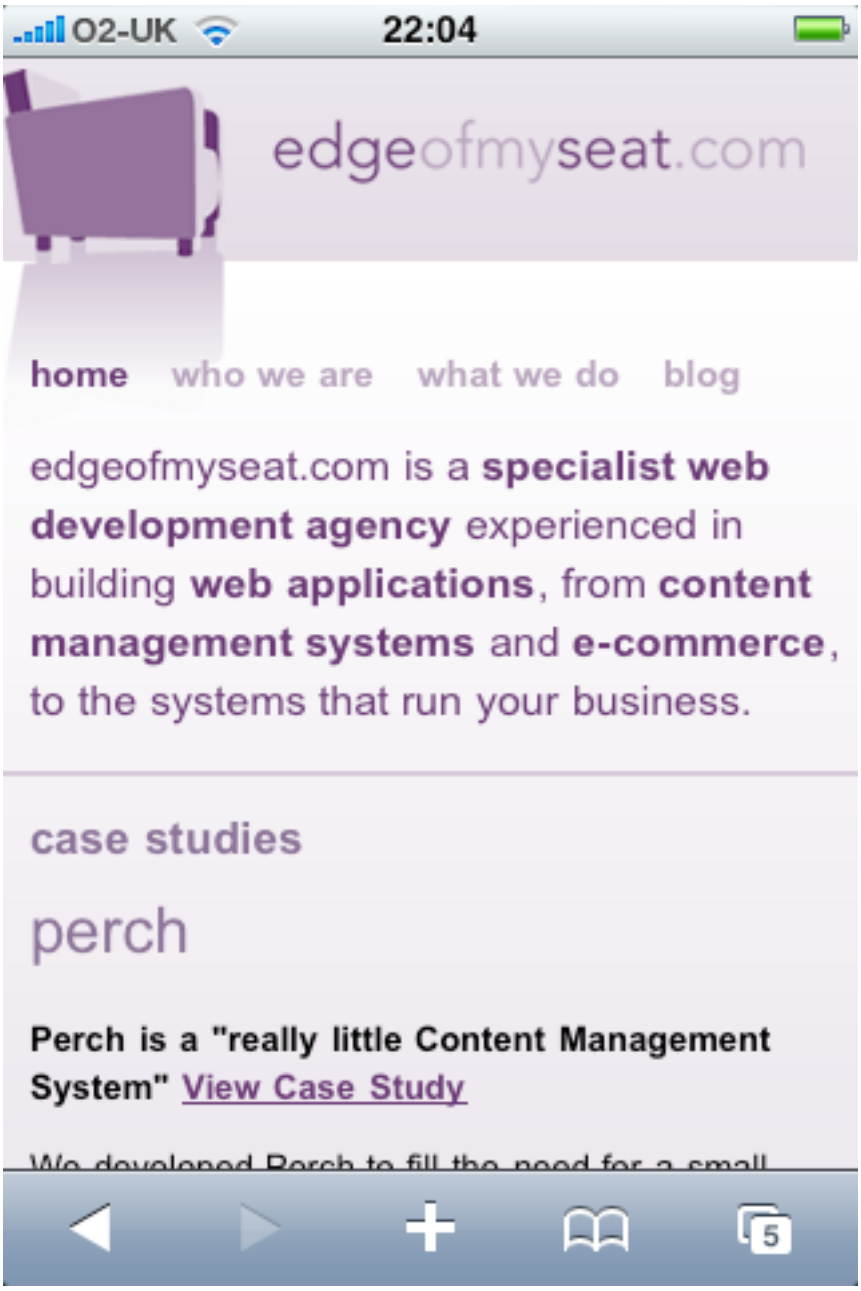
Testing the site using ProtoFluid

TESTING IN AN IPHONE

Having created my stylesheet and uploading it I wanted to check how it worked in a real target device. In the iPhone I discovered that the site still loaded zoomed out rather than zooming in on my nice readable single column. A quick search on the Safari developer website gave me my answer – to add a meta tag to the head of the website setting the width of the viewport to the device width.

```
<meta name="viewport" content="width=device-width" />
```

After adding the meta tag the site now displays zoomed in one the single column.



The site as it now displays on an iPhone

This was a very simple retrofit to show that it is possible to add a mobile version of your site simply. If I was building a site from scratch that I would be using media queries on, there are definitely certain choices I would make to make the process simpler. For example considering the linearized column orders, using background images where possible as these can be switched using CSS – or perhaps using fluid images.

Our desktop layout features a case studies carousel on the homepage, this wasn't easy to interact with on a touch screen device and so I checked using JavaScript if the browser window was very narrow and didn't launch the carousel. The way this was already written meant that the effect of stopping the carousel loading was that one case study would appear on the screen, which seems a reasonable solution for people on a small device. With a bit more time I could rewrite that carousel with an alternate version for users of mobile devices, perhaps with interactions more suitable to a touch screen.

Providing support for Media Queries in older browsers

This article covers the use of media queries in devices that have native support. If you are only interested in supporting iPhone and commonly used mobile browsers such as Opera Mini you have the luxury of not needing to worry about non-supporting browsers. If you want to start using media queries in desktop browsers then you might be interested to discover that there are a couple of techniques available which use JavaScript to add support to browsers such as Internet Explorer 8 (and lower versions) and Firefox prior to 3.5. Internet Explorer 9 will have support for CSS3 Media Queries.

More inspiration

To see more interesting use of Media Queries have a look at Hicksdesign where Jon Hicks has used Media Queries to not only provide a better experience for mobile device users, but also for regular web browser users with smaller windows. Also, have a look at Simon Collison's website and Ed Merritt's portfolio for other examples of this technique.

Try it for yourself

Using Media Queries is one place you can really start to use CSS3 in your daily work. It is worth remembering that the browsers that support media queries also support lots of other CSS3 properties so your stylesheets that target these devices can also use other CSS3 to create a slick effect when viewed on an iPhone or other mobile device. If you have implemented media queries on your site, or try this technique after reading this article, let us know in the comments.

Device-Agnostic Approach To Responsive Web Design

Thierry Koblentz

This is a different take on Responsive Web design. This article discusses how we can better embrace what the Web is about by ignoring the big elephant in the room; that is, how we can rely on media queries and breakpoints without any concern for devices.

The Challenge

Let's start our journey by looking at these online tools:

- Responsive Design Testing
- Responsive.is
- Responsinator
- BriCSS

Those pages let people check websites through a set of pre-built views based on various device sizes or orientations. Bricss goes one step further as it allows you to "customize" viewports by setting any dimensions you want.

Now check the-great-tablet-flood of 2011.

Do you get my drift? Trying to check layouts against specific sets of dimensions is a losing battle. Besides, using existing devices to set break-

points is not what I'd call a "future proof" approach, as there is no for standard sizes or ratios.

I don't want to go the "consider it to be harmful" route, but I want to point out that tools like these, or articles promoting a device approach (i.e. Device Diagram for Responsive Design Planning), make people focus on the wrong end of the problem, reinforcing the idea that *responsive* is all about devices.

To me, it seems more realistic to check our layouts through viewports of arbitrary dimensions and shapes. We don't need anything fancy, we can simply drag the bottom right corner of our favorite desktop browser to enter: "Device Agnostic Mode".

The Goal

The goal is to surface content, to style boxes as columns so they bring sections above the fold. The question is: when should we bring a box "up"?

Content Is King!

If we consider that content is king, then it makes sense to look at it as the corner stone of the solution. In other words, we should set break-points according to content instead of devices.

THE PRINCIPLE

The content of a box *dictates* its width. It is the minimum width of adjacent containers that create break points (a size at which we can display boxes next to each other).

- Decisions are made keeping these points in mind:

- The width of a box should be as small or as wide as possible without impairing readability.
- The max-width of a box should take into consideration the importance of following boxes. This is because the wider the box, the wider the viewport must be to reveal subsequent boxes.
- The goal is *not* to bring everything above the fold (we don't want to fill the viewport with clutter).

In Practice

MARKUP

For this exercise, we will consider 5 main blocks:

```
<div class="grid-block" id="header"></div>
<div id="wrapper">
    <div class="grid-block" id="main"></div>
    <div class="grid-block" id="complementary"></div>
    <div class="grid-block" id="aside"></div>
</div>
<div class="grid-block" id="contentinfo"></div>
```

The wrapper will allow us to:

- mix percentages and pixels to style boxes on the same row
- set a maximum width for a group of boxes

CSS

To build our grid we will rely on **`display:inline-block`** mainly for horizontal alignment and inline flow. But note that this choice also gives us an extra edge to play with (more on this later).

Also note that we will override this styling with **`float`** to achieve some specific layouts.

```
body {
    margin:auto;            /* you'll see why later */
    text-align:center;      /* to center align grid boxes */
    letter-spacing: -0.31em;/* webkit: collapse white-space
between units */
    *letter-spacing: normal;/* reset IE < 8 */
    word-spacing: -0.43em;  /* IE < 8 && gecko: collapse
white-space between units */
}
.grid-block {
    letter-spacing: normal; /* reset */
    word-spacing: normal;   /* reset */
    text-align:left;        /* reset */
    display:inline-block;   /* styling all grid-wrapper as
inline blocks */
    vertical-align:top;     /* aligning those boxes at the top
*/
    *display:inline;        /* IE hack to mimic inline-block
*/
    zoom:1;                 /* part of the above hack for IE
*/
    width:100%;             /* boxes would shrink-wrap */
}
/**
 * rules below are meant to paint the boxes
 */
.grid-block {
```

```
    height: 150px;
}
#header {
    background: #d6cac1;
}
#main {
    background: #ad9684;
}
#complementary {
    background: #7a6351;
}
#aside {
    background: #000000;
}
#contentinfo {
    background: #3d3128;
}
```

This produces a bunch of rows.

CONTENT-DRIVEN PROCESS

We define the width of each box according to its content. These values will then be used to set breakpoints. Note that the values below take into consideration a 10px gutter between columns.

Header
content: logo, navigation, search box
type: banner
minimum width: n/a
maximum width: n/a

Main
content: diverse (article, blog entry, comments, etc.)
type: main box that holds the meat of the page

minimum width: 420px [1]
maximum width: 550px [1]

Complementary
content: directory entries, tweets, etc.
type: multi-line text box with media
minimum width: 280px
maximum width: 380px

Aside
content: Ads
type: 230px wide images
fixed width: 250px or 490px (2 ads side by side)

Contentinfo
content: resources, blog roll, etc.
type: lists of links
minimum width: 220px
maximum width: 280px

The minimum and maximum widths above only come into play when the box is displayed as a column.

Breakpoints

The width of the containers establishes our breakpoints. Breakpoints are viewport's widths at which we decide to display a box as a column (instead of a row).

HOW DO WE "PICK" BREAKPOINTS?

Until we are able to use something like grid layout, we are pretty much stuck with the HTML flow, and thus should rearrange boxes while respecting their source order. So we go down our list, and based on the minimum width values, we create various combinations. The values below show widths at which we rearrange the layout, styling rows as columns, or changing the width of a specific column.

470PX

- header
- Main
- Complementary
- Aside (250) + Contentinfo (220)

530PX

- header
- Main
- Complementary (280) + Aside (250)
- Contentinfo

700PX

- header
- Main (420) + Complementary (280)
- Aside
- Contentinfo

or:

- header
- Main (420) + Complementary (280)
- Aside + Contentinfo

950PX

- Main (420) + Complementary (280) + Aside (250)
- Contentinfo

1170PX

- Main (420) + Complementary (280) + Aside (250) + Contentinfo (220)

1190PX

- Main (420) + Complementary (280) + Aside (490)
- Contentinfo

1410PX

- Head (240) Main (420) + Complementary (280) + Aside (250) + Contentinfo (220)

All of the above are potential breakpoints — each value could be used to create different layouts for the page. But is that something we *should* automatically do? I think not. At least not without considering these two points:

How close are the breakpoints?
We have 2 that are 20 pixels apart (1170px and 1190px); should we set both of them or should we drop one? I think that above 900px, chances are that desktop users may easily trigger a re-flow in that range, so I would not implement both. In other words, I think it's okay to go with close breakpoints if the values are below 800px — as there is less chance to confuse users when they resize their browser window.

Should we try to create as many columns as we can?
Bringing more ads above the fold may make more sense than bringing up a list of links that you'd generally keep buried in your footer. Also, you may choose to give more breathing room to your main content before bringing up boxes that the user does not really care for.

GETTING READY FOR MEDIA QUERIES

For the purpose of this article, we'll use every single one of our breakpoints to create a new layout, which should also demonstrate that it is not necessarily a good idea.

```
/**
 * 470
 */
@media only screen and (min-width: 470px) and (max-width:
529px) {
    #aside {
        width: 250px;
        float: left;
```

```
    }
    #contentinfo {
        display: block;
        width: auto;
        overflow: hidden;
    }
}
/**
 * 530
 */
@media only screen and (min-width: 530px) and (max-width:
699px) {
    #wrapper {
        display:block;
        margin: auto;
        max-width: 550px; /* see comment below */
    }
    #complementary {
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
        padding-right: 250px;
        margin-right: -250px;
    }
    #aside {
        width: 250px;
    }
}
/**
 * 700
 */
@media only screen and (min-width: 700px) and (max-width:
949px) {
    #wrapper {
        display:block;
        margin: auto;
        max-width: 830px; /* see comment below */
```

```
    }
    #main {
        float: left;
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
        padding-right: 280px;
        margin-right: -280px;
        height: 300px;
    }
    #aside,
    #complementary {
        float: right;
        width: 280px;
    }
    #contentinfo {
        clear: both;
    }
}
/**
 * 950
 */
@media only screen and (min-width: 950px) and (max-width:
1169px) {
    #wrapper {
        display:block;
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
        padding-right: 250px;
        margin: auto;
    }
    #main {
        width: 60%;
    }
    #complementary {
        width: 40%;
```

```
    }
    #aside {
        width: 250px;
        margin-right: -250px;
    }
}
/**
 * 1170
 */
@media only screen and (min-width: 1170px) and (max-width:
1189px) {
    #main,
    #complementary,
    #aside,
    #contentinfo {
        float: left; /* display:inline here leads to rounding
errors */
    }
    #main {
        width: 36%;
    }
    #complementary {
        width: 24%;
    }
    #aside {
        width: 21%;
    }
    #contentinfo {
        width: 19%;
    }
}
/**
 * 1190
 */
@media only screen and (min-width: 1190px) and (max-width:
1409px) {
    #wrapper {
```

```
        display:block;
        box-sizing: border-box;
        padding-right: 490px;
        margin: auto;
    }
    #main {
        width: 60%;
    }
    #complementary {
        width: 40%;
    }
    #aside {
        width: 490px;
        margin-right: -490px;
    }
}
/**
 * 1410
 */
@media only screen and (min-width: 1410px) {
    body {
        max-width: 1740px;
    }
    #wrapper {
        float: left;
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
        width:100%;
        padding-left: 17%;
        padding-right: 16%;
        margin-right: -16%;
        border-right: solid 250px transparent;
    }
    #header {
        float: left;
        width:17%;
```

```
        margin-right: -17%;
    }
    #main {
        width: 60%;
    }
    #complementary {
        width: 40%;
    }
    #aside {
        width: 250px;
        margin-right: -250px;
    }
    #contentinfo {
        width: 16%;
    }
}
```

For the 530px and 700px breakpoints, there is a design choice to make. Without a max-width, we'd get everything flush, but the main box (**`#main`**) would be larger than the maximum width we originally set.

Demo

Please feel free to check out the demo of this technique.

The last thing to do is to create a layout to cater for IE6/7/8, as these browsers will ignore the media queries. To do so, we can use a Conditional Comment:

```
<!--[if lt IE 9]>
    <style>
    body {
        margin: auto;
        min-width: 850px;
        max-width: 1000px;
```

```
        _width: 900px;
    }
    #main {
        width: 55%;
    }
    #complementary {
        width: 25%;
        *margin-right: -1px; /* rounding error */
    }
    #aside {
        width: 20%;
    }
    #contentinfo {
        clear:both;
    }
    </style>
<![endif]-->
```

Conclusion

Not once in this article I referenced the width of a device, be it an iPad, a Xoom, or something else. Building a "content-aware grid" is a simple matter of choosing the "layout patterns" that you want, based on breakpoints that you set according to page content.

After I sent this article to Smashing Magazine, I ran into Deciding What Responsive Breakpoints To Use by @Stephen_Greig. So obviously, we are at least two who share the sentiment that relying on devices to create layouts is the wrong approach. But I'm curious to know what everyone else is doing, or even thinking? If we had more people pushing for this, the community could be less device-centric, and could start focusing on content again.

NEXT STEP: RESPONSIVE MEDIA

- Polyfilling picture without the overhead
- Experimenting with Context-Aware Image Sizing

FOOTNOTES

[1] According to Ideal line length for content this box should be styled width a min-width of 25em and a max-width of 33em. So if your base font-size is 16px (as it should be), this translates as 400 pixels and 528 pixels.

Content Prototyping In Responsive Web Design

Ben Callahan

Michelangelo once said,

> *The best of artists has no conception that the marble alone does not contain within itself.*

Translate this to the world of Web design and you might say,

> *No matter how great a designer you are, you're only as good as your content.*

While the reality of client work sometimes makes it challenging to gather and produce content prior to starting the design, this is now widely accepted as being necessary. You may have heard this referred to as "content-driven design." I'm not the first to suggest that our current approach to responsive Web design could be improved by imparting a bigger role to content in determining how our websites respond. However, I haven't seen many (if not any) practical explanations on how to do this. I'd like to start this conversation by introducing a theoretical concept called a "content prototype."

What Is A Content Prototype?

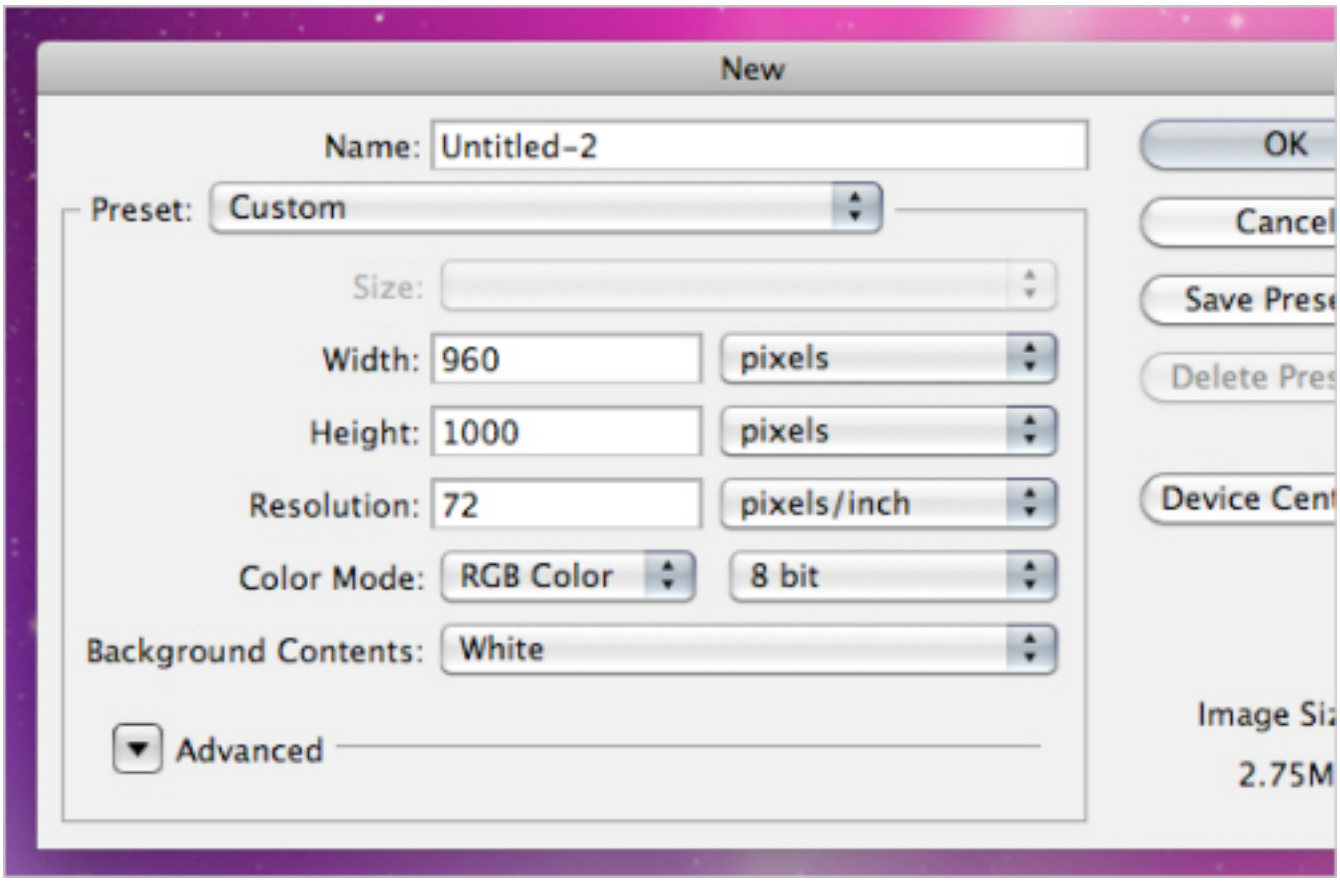
A content prototype is an HTML-and-CSS-based fluid-grid prototype, consisting of layout and typography, that consists of the project's actual content. Its greatest usefulness may be in determining where to apply media queries to make the Web design responsive.

For centuries, we have shaped our layouts and typefaces according to the meaning of the content. This has traditionally been done on fixed-width pages. We have inherited a fixed-width mentality in designing for the Web, when in fact the Web is not fixed-width. Users come to our websites for content. We should strive to present this content in the most appropriate and readable way possible.

Let's Get Theoretical

The following is a theoretical walk-through of how one might use a content prototype in real life. Again, this is intended to begin a conversation on how we can marry the concepts underlying content-driven design and those of responsive Web design.

Imagine that you are about to begin designing a website. The website will consist of a single page, which contains a block of text and a few short excerpts of related text. You've done your homework, and the content is fully written. You have solid documents of the architecture and wireframe that establish the priorities for this page. You also know that the website will be responsive. You've opened up your design tool of choice, and you're now looking at the "**`File → New`**" dialog. What to enter in those pesky little "width" and "height" fields?



Photoshop's new file dialog box.

Perhaps it doesn't matter.

Consider this. The goal of this process is to create a website that begs to be read at *any* resolution. So, start at whatever resolution you'd like, whatever you're comfortable with. Every resolution is important, not just the resolutions that last month's analytics say are the most popular.

Because we're following the principles of content-driven design, start with the highest-priority content on the page (the real content). Don't worry about anything other than the typeface, font size, column width and layout. Make it a pleasure to read. This is about as basic as you can get, because you haven't yet created icons, textures or illustrations; those elements are important, but they should support the content, and you can work on them later.

Next, code the simple page that you've designed using a fluid grid. This is critical; when your browser's window is about the same width as the canvas that you started with, the content prototype should look very much the same. This gives you the chance to play with the prototype in a browser and make informed decisions about where your media queries should fire. Using this method, the content will dictate where your fluid grid breaks down. These breakages are where you should apply media queries; they are opportunities for more dramatic changes. Make these changes, always focusing on the legibility of the content.

Following this pattern, you would add media queries at points where the fluid grid falls apart. Soon, you will have a full spectrum of resolutions, with beautiful and appropriate reading experiences. Once this is done, you will have a finished content prototype that demonstrates the readability of your content outside of the context of any device-specific resolutions.

BENEFITS OF A CONTENT PROTOTYPE

Thinking this way about the process of responsive Web design makes the content a filter through which all other decisions are made. The goal is to add a degree of cohesion between the message and the design of the website that would be difficult to achieve without such an approach.

Another challenge with responsive Web design is in testing usability across all resolutions. A round of usability testing with a completed content prototype could quite possibly give an early glimpse of problems with changes to the layout. The sooner you can identify these problems, the lower the cost of fixing them.

Additionally, content prototypes give you an opportunity to show that layout changes are possible before spending days designing every detail. The iterative nature of content prototyping invites collaboration between designer and coder — even if they happen to be the same person.

DESIGN IN THE BROWSER

If you are one of those designers who also codes, then you've probably recognized that this can all happen right in the browser. If you have the skills to do both, then by all means, start in the browser. With the emergence of CSS3 features like **`border-radius`**, **`text-shadow`** and **`gradient`**, designing in the browser is more feasible today than ever before.

We're all frustrated with our tools. Perhaps this is because they are all fixed-width tools! But regardless of whether we have the right tools for the job, we cannot continue to rely on common device-specific resolutions if we want to build websites that work well into the future. A content prototype gets our content into the browser as early as possible.

Problems With Content Prototypes

Obviously, this approach does have some limitations. Foremost, not every website is content-driven; many websites are workflow-driven. Without getting sidetracked by the whole question of what content is, we can recognize that this process wouldn't necessarily work if you don't have the content first.

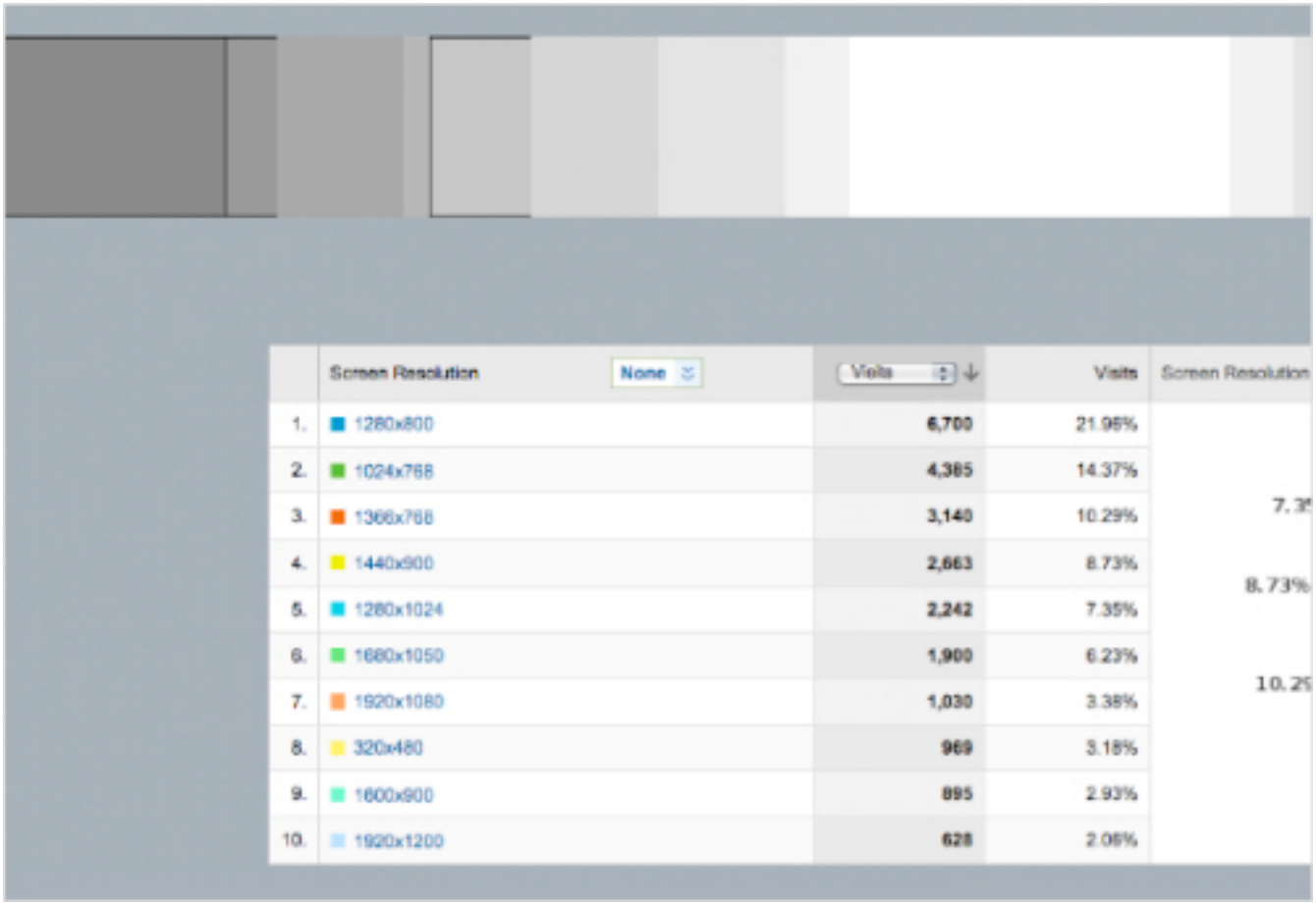
Also, if you're a designer who is not also a front-end developer, then a lot of back and forth will be required to complete the content prototype. This is not necessarily bad, but it will certainly take time.

Then there is the problem of what to do once you have a completed the content prototype. You could continue designing and adding to the code that you have. But if you do choose this route, then you will need to start with the smallest resolution first, because this is the best practice for coding a responsive website.

Alternatively, you could use this process merely to determine where your media queries should fall. In this case, this would be a lot of work just to find the proper breaks in the content's readability.

ALTERNATIVE APPROACHES

There are other ways to make these decisions. Using traffic data, some people might build a graph to determine the breaking points in their design.



	Screen Resolution	None	Visits	Visits	Screen Resolution
1.	1280x800		6,700	21.96%	
2.	1024x768		4,385	14.37%	
3.	1366x768		3,140	10.29%	7.3
4.	1440x900		2,663	8.73%	
5.	1280x1024		2,242	7.35%	8.73%
6.	1680x1050		1,900	6.23%	
7.	1920x1080		1,030	3.38%	10.2
8.	320x480		969	3.18%	
9.	1600x900		895	2.93%	
10.	1920x1200		628	2.06%	

Using traffic data to determine where to apply media queries.

Ethan Marcotte describes a similar process in his book *Responsive Web Design*, but he also suggests resolutions that are common among popular devices these days. Also, coding a responsive website that has already been fully designed could cause problems if you're not sure whether the layout can be achieved with CSS alone. As mentioned, content prototyping lets you experiment with these layouts before fully committing to a change for a given resolution.

The point of all this is to make our content more readable, independent of what device it's being viewed on. If content prototyping doesn't work, maybe we could find some way — other than relying on which devices are currently popular — to make content-driven decisions about the design and layout. The guys at Front have been experimenting with this as well, calling it "The Goldilocks Approach to Responsive Design." Their technique is to use an em-based layout (instead of the typical fluid-grid approach of percentage widths) to create a great reading experience at all resolutions.

A HELPFUL TOOL

One of the developers on our team recently created a media query bookmarklet, which displays in real time your browser's width and height and any media queries that have been triggered. This tool can be very helpful when doing responsive Web design. Feel free to experiment with it; I hope it simplifies the process for you as it has for us.

Building For The Future

The aim is lofty, designing for the future. Just as we build websites to be accessible to the widest audience possible — because that is the right way to build them — we should build websites that embrace the fluidity of the Web. A challenge is before us to find ways to present our content appropriately without knowing which devices it will be viewed on. We must shift our focus back to the user. A content-out approach is a user-centered approach.

It won't be long before we're interacting with the Web in ways we never imagined. We may be near a time when fixed-width websites are considered outdated. Whether or not that happens, our websites should be flexible enough to present readable content to all of our users. Moreover, assuming what a user with a small screen wants from your website can be dangerous. Mobile-specific websites are certainly appropriate at times, but there are a few reasons why a website should not (at a minimum) be built responsively.

In order to get better at what we do, we must keep pushing the process forward. If you have other ideas on how to separate our decisions about a website's design from popular device resolutions, or even knowledge of the benefits or problems of content prototyping, please share.

About The Authors

Ben Callahan

President and Founding Partner of Sparkbox, a web studio building sites that are just as brilliant for mobile as they are for the desktop.

James Young

James Young is Creative Director at Offroadcode. With a background working as a freelancer for years, James looks after all design work along with writing HTML/CSS for his clients.

Jeremy Hixon

Jeremy is a Senior User Experience Developer at AREA203 DIGITAL and has a personal blog and portfolio. He is a reader of fiction, watcher of educational television, generally loud-mouthed and opinionated.

Kayla Knight

Kayla Knight is a full-time freelance web designer and developer, and likes to blog a lot too. She also created and runs Freelance Mingle, a social network for freelancers.

Lewis Nyman

Lewis designs and builds stuff on the web and has been doing so for five years. He works as a full time Drupal soldier at Capgemini UK. He also is leading the 'Drupal Eight Mobile User Experience' charge and is on the organization team for the 'Drupal Eight Design Initiative'. He is often found in London typing about himself in the third person.

Rachel Andrew

Rachel Andrew is a front and back-end web developer and Director of edgeofmyseat.com, a UK web development consultancy and the creators of the small content management system, Perch. She is the author of a number of web design and development books including CSS Anthology: 101 Essential Tips, Tricks and Hacks (3rd edition), published by SitePoint and also writes on her blog rachelandrew.co.uk. Rachel tries to encourage a common sense application of best practice and standards adoption in her own work and when writing about the web.

Thierry Koblentz

Thierry is passionate about Web Design and CSS. He loves the challenge of solving problems and dreaming up original ideas that get him out of bed in the morning and keep him working late at night. He is a front-end engineer at Yahoo! and owns TJK Design and ez-css.org.