Grand Valley State University

From the SelectedWorks of Matthew Reidsma

Winter March 1, 2013

Responsive Web Design for Libraries: Beyond the Mobile Web

Matthew Reidsma, *Grand Valley State University*

Available at: https://works.bepress.com/mreidsma/1/

Responsive Web Design for Libraries: Beyond the Mobile Web

by Matthew Reidsma, Grand Valley State University

At some point between the moment I wrote these words and the moment you are reading them on a printed page, this essay was coiffed and primped by a designer for optimum readability. That designer, working in the medium of print, knows with utmost certainly the size of the book, her canvas, and made her design decisions accordingly.

If you were reading this on the web, I wouldn't be able to predict the size of the "page" you were viewing, because web-enabled devices have never looked so different. We can no longer count on one or two common screen resolutions when we design a website. We're on track to have more mobile devices than people on the planet (World Bank 2012), and in the U.S. nearly half of adults who own cellphones have a smartphone (Pew 2012). What's more, recent data is showing that adults who own more than one Web-enabled device is growing, with over half of computer owners also owning a smartphone and 13% of Americans owning a laptop or desktop computer, a tablet, and a smartphone (Mitchell, Rosenstiel, and Christian 2012). We have more devices with a greater variety of screens to design for than ever before.

Giving Up Control

Web designers are just waking up to the realization that they no longer have the kind of control a print designer enjoys. But true to our desire for control, we address this problem by creating more of the same fixed-width layouts, but this time one roughly phone-shaped, maybe a tabletish one, and finally, the "regular" desktop site. We check for known mobile devices and browsers and send these visitors to separate websites. Our feeling of control returns, until new phones and tablets hit the streets, requiring us to look for another dozen, or two dozen devices to make sure each device gets to its appropriate site.

For the past twenty years, we have built fixed-width websites for a handful of screen sizes dictated by screen manufacturers. In the mid-nineties we made sites that were optimized for screens 480 by 640 pixels, and a few years later, we expanded to 600 by 800 pixels. In the last decade we hovered comfortably designing for screens 1024 pixels wide and 768 pixels high, and we used every inch of that canvas, because we knew the sizes of the screens our visitors had on their computers.

But the web itself has never had the kind of fixed canvas of the printed page. The screen never was the canvas. As Ethan Marcotte has pointed out, screen size is “one step removed from our *actual* canvas: the browser window” (2011, 3). We’ve never had a fixed canvas, because the users of our websites always had control of the size and shape of the browser.We’ve never actually had control.

Accepting this doesn’t mean that we have to abandon building beautiful sites and go back to pages of unstyled text. Long before mobile devices brought about the current crisis in web design, John Allsopp pleaded with web designers to “embrace the fact that the web doesn’t have the same constraints [as print], and design for this flexibility” (2000). The ability to build fluid websites has always been a part of the web, but now, more than a decade after Allsopp’s call to arms, we have a more sophisticated arsenal of standards-based tools to make websites that adapt to changes in the user’s world.

Tackling Mobile in the Library

When I came to Grand Valley State University in late 2010, they were already trying to find a solution to the increased mobile usage of their website. That year, the GVSU Library website traffic from mobile devices was a only .5% of all visits, while the University as a whole saw 4.5% of its traffic from mobile devices. Now just 18 months later, GVSU's library's mobile traffic has seen a tenfold increase. Visits from mobile devices for the first 6 months of this year hit 4.95%. That's nearly 1 in every 20 visitors to our website using a mobile device.

As an academic library, we knew this growth will only continue. Recent studies report that a whopping 77% of teens age 12–17 have a cellphone, and 25% of those are smartphones (Lenhart 2012). In addition, 45% of 18–29 year-old who browse the web on their phone use the phone as their primary device (Smith 2012). We needed to do something about our fixed-width website, which was barely usable on small-screen devices.

I developed a few core goals for the new site. First, all content and services would be available in full on all devices. I didn't want to make assumptions about what users on mobile devices wouldn't use. Our library would never think of limiting access to a patron in a wheel chair because we assumed they wouldn't be interested in a service, yet it is common practice to assume that mobile users only want hours and directions and to remove the rest of the website's functionality from a mobile site. I also wanted to avoid having 2 URLs for each piece of content, one for the mobile site and one for the desktop site, which can be frustrating for multiple-device users. We would have one site, not separate mobile and desktop websites.

I also wanted to take advantage of the capabilities of newer devices without shutting out older devices and assistive technologies. So I would build the site with clean, standards-based HTML and then enhance the site with CSS and JavaScript on more capable devices as much as possible. Like most libraries, we rely on many hosted vendor products to provide services to our patrons, so I wasn't in charge of much of the code being served to our patrons. Even though each vendor offered different levels of customization possibilities, I wanted all our disparate sites to appear to be a single, cohesive website. With no budget. And a team of one (me).

Responsive Web Design

In early 2010, Ethan Marcotte, a web design from Boston, coined the phrase Responsive Web Design to refer to a suite of techniques for building fluid, standards-based websites that adapt to user devices. By designing sites with a fluid grid, using CSS media queries to change styles according to device screen sizes, and using flexible images, Marcotte showed that we no longer needed to serve up separate websites to mobile devices. We could have a single code base, a single URL for each resource, and a great looking site, no matter the device. This was the approach I took to redesigning our site. You can follow along on your own device at http://gvsu.com/library.

Mobile First

In the past, I started all of my Web design projects in Photoshop or the browser, putting together mockups of a site that would be a fixed, desktop width. But I've started thinking differently about how I structure and build websites. Rather than starting with a vision of the site on a large screen and then stripping away features or styles as the screen size drops, I now start with the smallest screen as my base and then add styles to change the layout for devices with wider screens.

And this doesn't just affect how I write CSS. The order that your HTML is presented to the user is important. In most sites, we are presented with a list of navigation items to other pages before we ever get to the content. Yet no one comes to your website for the navigation: they come for the content.

When we moved from table-based layouts to CSS, we thought that by simply keeping our style attributes separate from our HTML we were separating style from structure. But we continued to put the navigation at the top of our document structure since that's where we expected it to appear on the page. By moving navigation below our content, we can improve the experience less-capable and assistive devices have visiting our sites and use CSS to progressively enhance the site for devices that have CSS capable browsers. In fact, Marcotte has recently been proposing that Web designers think of layout itself as an enhancement (Wroblewski 2012).

This project was completed in a little over a week because I had created a User Interface Pattern Library, a well-documented CSS library that gives me a head start on the common design patterns we use throughout our Web projects. By including these styles, I no longer had to worry about styling typography or lists. You can see our reference document or grab the code for our pattern library at http://gvsu.edu/library/ui. As I walk you through how Responsive Web Design helped us solve our content layout issues on multiple screen sizes, I'll ignore typographical styles and focus on the styles for layout.

Since we're starting with small-screen styles, we can treat each chunk of content as a block element. Because the screen is so narrow, we don't need to do much to adapt the layout of our content beyond setting up an appropriate HTML source order. However, navigation posed some special challenges. Because we've moved the navigation below the content, we need a way to give users quick way access to it without interfering with the content of the page. The simplest solution is to add an anchor link in the markup after the header that jumps the user down to navigation, like this:

```
<div id="gvsu-library_menu"><a href="#navigation">Menu</a></div>
    ...
<div id="navigation">
    <ul>...
```

Now the page is looking pretty good on small screens , but as I make the browser window wider, it starts to look less inviting. In fact, once the browser window gets beyond 800 pixels, the site starts to look silly [Figure 1]. I needed a way to enhance the layout as the screen size increased.

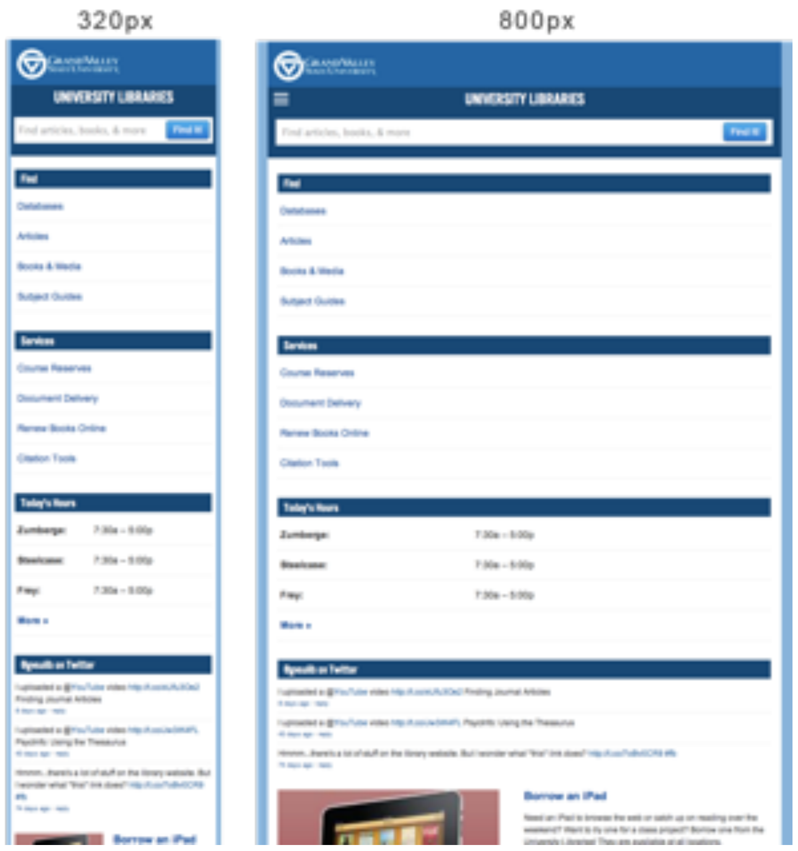


Figure 7.1 Although small-screen styles look ridiculous on larger screens, the site is still usable.

Media Queries

If you've been writing CSS for a while, you are probably familiar with the media type attribute. It allows you to make some of your styles conditional based on the type of device the page was loading on. You could have separate styles for screens, for projectors, and for print. You could even specify a stylesheet for "handheld," but support was poor in early mobile browsers.

The CSS3 specification pushed these media attributes farther, creating a way to check in with devices to see if certain conditions were being met. Now instead of serving the same CSS to all screens, for instance, we could specify a subset of CSS to be loading on screens of a certain width or aspect ratio. Here is a block of CSS that will only be applied to screens that are at least 600 pixels wide:

```
@media screen and (min-width:600px) {
   /* Awesome styles here */
}
```

This gives us a lot of flexibility on loading size-specific CSS, since we can target screen widths which will help capture a wide range of devices.

However, pixel values in media queries have some drawbacks. A better solution would be to set our queries against ems. Rather than being a fixed width measurement, ems are a relative measure that allows for user zooming. Unless you change the default body font-size in your style sheet, you can assume that by default 1em is 16 pixels. To convert pixels to ems, just divide the desired pixel width by the pixel equivalent of an em. In this case, instead of calling the stylesheet at 600 pixels, we'll call it at 37.5 ems (600 ÷ 16 = 37.5). So our query would read:

```
@media screen and (min-width: 37.5em) {
        /* Awesome styles here */
    }
```

Now we have a mechanism to load styles based on particular widths, but we still need to figure out how these elements are going to be displayed on larger screens. For that, we need a grid.

Fluid Grids

Grids can, to some extent, give us some of the control we loved about table-based layouts, where items were placed logically in columns and rows. Grids are, after all, systems "for ordering graphical elements of text and images" (Boulton 2005). Unlike tables, which reproduce inflexible grids in markup, our grids will simply guide us as we place our content on the page.

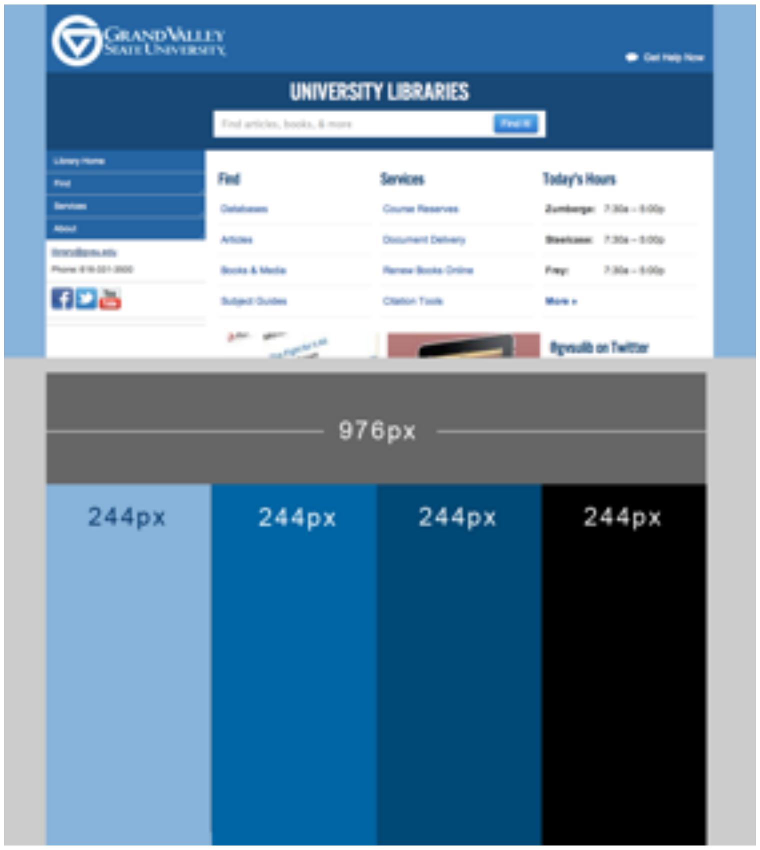


Figure 7.2 Our fixed-width mockup has four, equal columns. We can make this flexible with the magic of math.

It is easier to explain grids by working from large-screen styles down. I developed a mockup of our site on wide screens. Here, the content of the site is wrapped in a container 976 pixels wide, divided into four equal columns. So by dividing our total page width by four, we can begin to build our grid.

976 ÷ 4 = 244

Each of our columns will be 244 pixels wide [Figure 2]. But remember that our navigation comes in the HTML source *after* all of the content, meaning that we'll have to think of the content of the site in terms of three columns instead of four, while the navigation will serve as the stand-in fourth column. Here is the markup:

```
<div id="wrapper">
   <div class="line"><!-- First row of content -->
      <div class="column">
         <h3>Find</h3>
         ...
      </div>
      <div class="column">
         <h3>Services</h3>
         ...
      </div>
      <div class="column">
         <h3>Today's Hours</h3>
         ...
      </div>
   </div><!-- End .line -->
   <div class="line"><!-- Second row of content -->
      ...
```

```
    </div>
    ...
    <div id="navigation">
        ...
    </div><!-- End #navigation -->
</div><!-- End #wrapper -->

#wrapper {
    margin: 0 auto;
    width: 976px; }
.line {
    float: right;
    width: 732px; }
.column {
    float: left;
    width: 244px; }
#navigation {
    float: right;
    width: 244px; }
```

This works well for our site on large screens such as desktops or laptops, but the fixed-width layout gives us some problems on smaller screens or browser windows. But we don't have to give up our grid in order to make the page respond more gracefully to the widths of our users' screens. Instead, we can convert our pixel-based layouts to fluid, relative values using percentages. If we divide the desired width of our element by the width of the containing element, we can get the relative width of our elements expressed as a percentage. So for the markup above, when we divide the width of the column by the width of its container (.line) and multiply by 100, we get the relative width of each column:

(244 ÷ 732) * 100 = 33.333333333333%

So we can change our CSS to

```
.column {
   float: left;
   width: 33.333333333333%; }
```

The navigation column, however, is a child of the #wrapper div, so we need a different value for the relative width of this column:

(244 ÷ 976) * 100 = 25%

So we can change our CSS to

```
#navigation {
   width: 25%; }
```

We can calculate the relative width of the .line container in the same way, by dividing its value in relation to the parent element:

(732÷ 976) * 100 = 75%

So we can change our CSS to

```
.line {
   float: right;
   width: 75%; }
```

But we are still left with a fixed container, so our relative widths don't adjust to changes in screen size, since their container is a fixed width. But determining the relative width of the outermost containing element is tricky, because we don't *know* what the width of the element containing this element is. Nonetheless, we can use some guesswork and trial and error to find an appropriate value.

One way to do this is to look at our existing analytics and determine what different screen sizes commonly visit your site. We're not going to necessarily lock ourselves in to making a site that targets one specific width, but we can use this information to help understand what relative size your outermost containing element should be. For instance, if 50% of your large-screen visits come from screens 1024 pixels wide and the other half come from screens 1200 pixels wide, we can use the formula above to find the percentages your .line div would be for each of those containers.

$(976 \div 1024) * 100 = 95.3125\%$

$(976 \div 1200) * 100 = 81.333333333333\%$

Since in this hypothetical case, our audience is split evenly between these two options, we could take the average of the two values and make our .line 88.3229165%, but we might also us this as a starting place to try out a few resolutions and see what works best. In this case, 88% works pretty well on both screen sizes, so we can change our CSS now to read:

```
#wrapper {
   margin: 0 auto;
   width: 88%; }
```

Now our four-column layout is flexible, adjusting to differences in screen sizes, but we left out something important in calculating layouts: margins and padding. When calculating the relative widths of elements, determining the containing element was straightforward. But margins and padding are a different beast. For margins, divide the desired width of the margin by the width of the element's container. If we wanted each column to have a 6 pixel left margin, we would divide the margin width by the expected width of the .line, which is the containing element:

(6 ÷ 732) * 100 = 0.819672131148%

(There is no need to round these results, since computers are more comfortable than we are with complex numbers.)

For padding, we need to divide the desired padding width of the element itself, not its container. For instance, if we want to add 6 pixels of horizontal padding to each column in our layout, we would divide the desired pixel width by the width of .column:

(6 ÷ 244) * 100 = 2.459016393443%

But because the padding is added to the width of the element, we need to remove the same relative amount from our width element to accommodate the padding. So our CSS would now look like this:

```
.column {
   float: left;
   margin-left: 0.819672131148%;
   padding: 0 2.459016393443%;
   width: 22.540983606557%; }
```

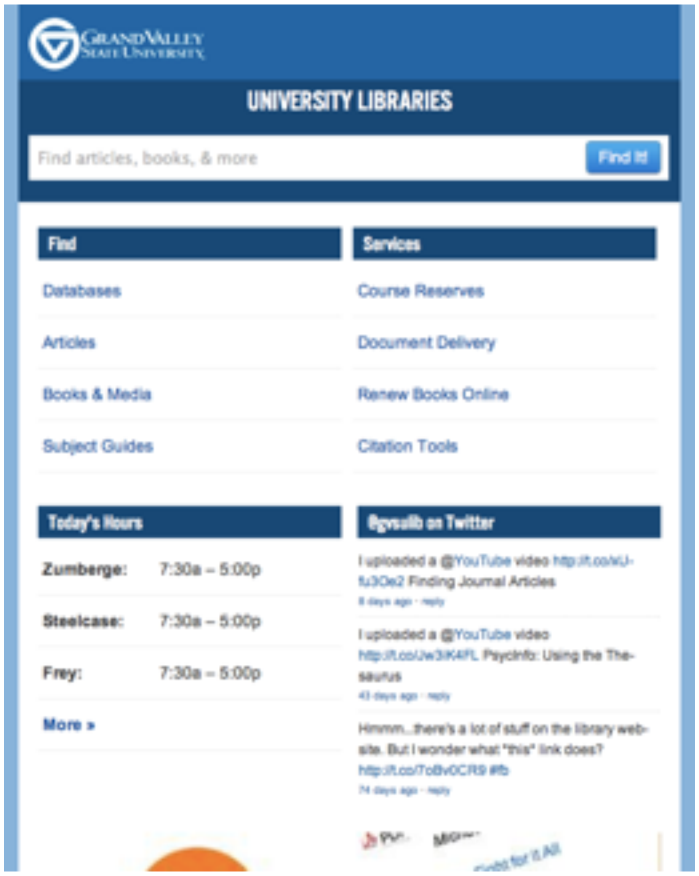
Now we have a nice, flexible four-column layout that looks great on wider screens. But even though we've built a flexible grid under our design, things start to look cramped at smaller screen sizes. We need to use media queries to load styles for progressively larger screens.

Because most websites are designed with wide, fixed-width layouts, most mobile devices scale these sites to be displayed on smaller screens. If you've ever loaded a website on your phone or tablet and then found yourself frantically pinching and zooming to make the text readable, you're familiar with this behavior. Because we don't want these devices to turn our newly responsive website into a pinch and zoom mess, we need a way to tell them to display our site at the scale and width of the device. All we need to do is add the following meta tag to the head of our document.

```
<meta name="viewport" content="initial-scale=1.0, width=device-width" />
```

Getting Responsive

Now at our smallest size, the site is a single column, while at the largest size, it will have four columns. The best way to determine where to put in some media queries to enhance the layout is to slowly make the window bigger and add new styles when things start to look crummy.



GRAND VALLEY STATE UNIVERSITY

UNIVERSITY LIBRARIES

Find articles, books, & more | Find It!

Find	Services
Databases	Course Reserves
Articles	Document Delivery
Books & Media	Renew Books Online
Subject Guides	Citation Tools

Today's Hours

Zumberge:	7:30a – 5:00p
Steelcase:	7:30a – 5:00p
Frey:	7:30a – 5:00p

More »

@gvsulib on Twitter

I uploaded a @YouTube video http://t.co/wIJ-fu3Oe2 Finding Journal Articles
8 days ago · reply

I uploaded a @YouTube video http://t.co/Jw3iK4FL PsycInfo: Using the Thesaurus
43 days ago · reply

Hmmm...there's a lot of stuff on the library website. But I wonder what "this" link does? http://t.co/ToBv0CR9 #fb
74 days ago · reply

Figure 7.3 As the screen gets larger, we have more room for columns.

As we start to scale up, the white space after the list items becomes excessive at around 480 pixels (30em). So we'll add a media query to switch to a 2-column layout here to make better use of the screen width [Figure 3]:

```
@media screen and (min-width: 30em) {
   .column {
      float: left;
      padding: 0 1%;
      width: 48%; }
}
```

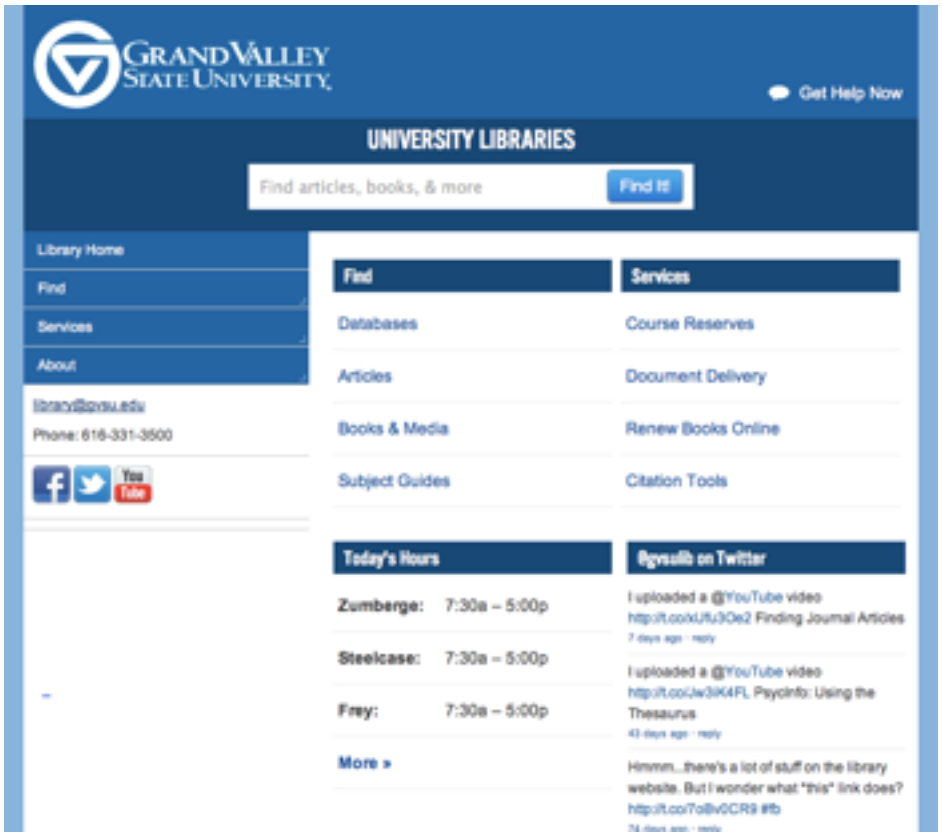
The rest of the page layout looks fine, so we'll continue to scale the width of the browser window up. At around 650 pixels wide (40.625em), the extra room in each column starts to feel excessive, so we'll move to a three column layout. The third column will be our left-side navigation. We need to set a relative width for the .line divs that contain our columns as well as a width for the #navigation div. In addition, we should hide the anchor link for small screens that gives users easy access to navigation. Also, the search bar is getting a little bit long, so we'll scale it down and center it. [Figure 4]. The new media query looks like this:

```
@media screen and (min-width: 40.625em) {
   .line {
      float: right;
      width: 66.66666666%; }

   #navigation {
      width: 33.3333333%; }

   #gvsu-library_menu {
      display: none; }
```

```
    #search-box {
        margin: 0 auto;
        width: 48%; }
}
```



GRAND VALLEY STATE UNIVERSITY

Get Help Now

UNIVERSITY LIBRARIES

Find articles, books, & more | Find It!

Library Home
Find
Services
About

library@gvsu.edu
Phone: 616-331-3500

Find	Services
Databases	Course Reserves
Articles	Document Delivery
Books & Media	Renew Books Online
Subject Guides	Citation Tools

Today's Hours

Zumberge:	7:30a – 5:00p
Steelcase:	7:30a – 5:00p
Frey:	7:30a – 5:00p

More »

@gvsulib on Twitter

I uploaded a @YouTube video http://t.co/xUfu3Oe2 Finding Journal Articles
7 days ago · reply

I uploaded a @YouTube video http://t.co/Jw3iK4FL PsycInfo: Using the Thesaurus
43 days ago · reply

Hmmm...there's a lot of stuff on the library website. But I wonder what "this" link does? http://t.co/7o8v0CR9 #fb
74 days ago · reply

Figure 7.4 Once the screen gets wide enough, we can move our side navigation into place.

Now the site is really coming along, but as the screen gets wider, it again feels more stretched. By 900 pixels (56.25em) the three-column layout is feeling a bit stretched, so it's time to use those styles we created above for our four-column grid. Since our campus homepage sets a fixed width of 976 pixels on the content container once the screen is at least 1024 pixels, I added the following style to make the #wrapper stop scaling.

```
@media screen and (min-width: 64em) {
    #wrapper {
        width: 61em; /* 976px / 16px = 61em */ }
}
```

Now we have a basic responsive website that adapts its layout to the size of the screen. And since we began with a foundation of clean, semantic HTML, we can be sure that the site will work great on any web-capable device.

Internet Explorer, or, What Else is New

While media queries enjoy great support in modern browsers, Internet Explorer 8 and earlier ignore them. If we start with mobile styles and then use min-width queries to enhance for larger screens, Internet Explorer will simply load the styles for small screens and ignore the rest. If you're an academic librarian like me, you have a campus full of computers running ancient versions of Internet Explorer, so simply leaving this unaddressed isn't an option. But there are a few ways to work around this limitation.

First, there are several JavaScript polyfills that will patch browsers that don't support media queries. Scott Jehl's Respond.js is the most lightweight (http://github.com/scottjehl/Respond). If you can't use JavaScript, you could restructure your media queries to begin with your wide screen styles, and then use max-width: media queries to scale the site down to smaller screens. IE will ignore all of the styles in media queries, so it will only load the initial styles for wide screens. This is the approach our campus uses, but it makes me uncomfortable. Less capable devices that support CSS but not media queries will get served a "desktop" website instead of a basic, small-screen optimized site. In my experience, scaling down from a desktop site also makes for more CSS.

The solution I chose was to create an IE-specific stylesheet that loads after your media queries, passing all of the wide-screen styles to early versions of Internet Explorer. That would look something like this:

```
<!--[if lt IE 9]>
   <link rel="stylesheet" type="text/css" href="ie.css" />
<![endif]-->
```

Since respond.js conflicted with some aspects of our campus CMS, this was the solution I took for the library website. IE users still won't get a responsive site, but users will less-capable mobile devices will get a site styled for mobile.

Flexible Images

The last element of responsive web design is flexible images. Since our website (and most library websites) are not image-heavy, making our images adapt to their containers is fairly easy. We just need one line to our CSS:

```
img {
   max-width: 100%; }
```

Now our images will never be larger than their containing elements. As the containers scale down, so will the images. Of course, if you are building a digital library with a lot of large images you probably will need a more sophisticated approach to dealing with images. There are several solutions under discussion for how to serve up different images to devices based on screen sizes, but none of them have been implemented by browser makers yet. If you need to serve up a lot of images, I recommend looking at Scott Jehl's Picturefill (https://github.com/scottjehl/picturefill), which replicates one of the proposed solutions in JavaScript.

User Response

Since we've only just launched the responsive design on the pages in our CMS, it is too soon to get any useful data on how our users are making use of the site. We also haven't advertised the change and don't intend to. Our primary audience is students, and as more and more of them use mobile devices, their expectations for the sites they visit increase, as well. We're expected to have a site that works as well on a mobile device as it does on a desktop computer, and no one congratulates you when you meet their expectations.

Further Challenges

Now that our library homepage and CMS are responsive, I'm faced with the task of replicating this design in all of our hosted vendor systems. Some vendors give us a lot of flexibility in how we design the user interface, such as Illiad, our Interlibrary Loan system. Others, give us less control. In order to gain control of the styles of our 360Link Link resolver from Serials Solutions, I wrote a script that scrapes the page and then rewrites it on the fly in our library's template. Others, like Serials Solutions Summon Discovery Service, give us no control over the design of the page. On those systems that do give us the ability to change things, however, we want to be consistent.

This is perhaps the greatest challenge for libraries employing Responsive Web Design. Because we likely will never have a consistent template that works across all of our vendor products, I may end up writing different CSS for every product in order to achieve the illusion of consistency. In addition, some of our services, like LibGuides and Serials' Solutions Summon, already redirect mobile users to a separate site. For everything else, Responsive Web Design has allowed us to offer all of our services to our users regardless of the device they use.

References

Allsopp, John. 2000. “The Dao of Web Design.” *A List Apart*. Accessed August 16, 2012. http://www.alistapart.com/articles/dao/.

Boulton, Mark. 2005. “Five simple steps to designing grid systems – Preface.” Accessed August 17, 2012. http://www.markboulton.co.uk/journal/comments/five-simple-steps-to-designing-grid-systems-preface.

Lenhart, Amanda. 2012. “Teens, Smartphones & Texting.” *Pew Internet & American Life Project*. Accessed August 16, 2012. http://www.pewinternet.org/Reports/2012/Teens-and-smartphones/Summary-of-findings.aspx.

Marcotte, Ethan. 2010. “Responsive Web Design.” *A List Apart*. Accessed August 16, 2012. http://www.alistapart.com/articles/responsive-web-design/.

Marcotte, Ethan. 2011. *Responsive Web Design*. New York: A Book Apart.

Mitchell, Amy, Tom Rosenstiel, and Leah Christian. 2012. “Mobile Devices and News Consumption: Some Good Signs for Journalism.” *The State of the News Media 2012*. http://stateofthemedia.org/2012/mobile-devices-and-news-consumption-some-good-signs-for-journalism/.

Pew Internet & American Life Project. 2012. “A Closer Look at Gadget Ownership.” Accessed August 16, 2012. http://pewinternet.org/Infographics/2012/A-Closer-Look-at-Gadget-Ownership.aspx.

Smith, Aaron. “Cell Internet Use 2012.” *Pew Internet & American Life Project*. Accessed August 16, 2012. http://www.pewinternet.org/Reports/2012/Cell-Internet-Use–2012.aspx.

World Bank. 2012. “Information and Communications for Development 2012: Maximizing Mobile.” Accessed August 16, 2012. http://www.worldbank.org/ict/IC4D2012.

Wroblewski, Luke. 2012. “An Event Apart: Rolling Up Our Responsive Sleeves.” Accessed August 17, 2012. http://www.lukew.com/ff/entry.asp?1494