EVALUATING THE EFFICIENCY OF THREE RESPONSIVE IMAGE TECHNIQUES ON AN IMAGE-HEAVY, RESPONSIVE WEBSITE

By
Audrey Anne Brockhaus

A Senior Project Presented in Partial Fulfillment of the Requirements for the Degree
Bachelor of Science

Arizona State University
Polytechnic Campus

October 2014

Abstract:

Responsive websites must appeal to desktop users, while also keeping page size to a minimum to accommodate slow wireless speeds on mobile. One of the largest factors in page-load times is the size of raster images on the page. Raster images can't be increased in size without losing resolution, and are therefore often loaded in the highest possible resolution and then scaled down for smaller screens. However, this hurts the performance of these sites on mobile devices by forcing them to download larger images than they will ever need to display. This project evaluated the impact of three techniques to deliver appropriately-sized images on a responsive, image-heavy website. Load times and performance were measured on both desktop and mobile devices. Picturefill 2.0 was found to be the fastest way to deliver responsive images to mobile devices on the sites built for this project. Based on the results of the tests conducted on the three responsive websites, the researcher would strongly recommend the use of Picturefill 2.0 on image-heavy responsive sites until the majority of browsers implement native support for the proposed HTML5 `<picture>` element.

TABLE OF CONTENTS:

Chapter I: Introduction and Statement of the Problem

Introduction

The general consensus in the web development community is that responsive design (where the page layout adjusts to fit the size of the device it is being displayed on) is vastly preferable to having separate mobile and desktop websites. However, this causes a problem for raster images, which can't be increased in size without losing resolution.

Generally, responsive websites will load in the largest possible version of an image and then scale it down for smaller screens. However, this hurts the performance of these sites on mobile, because the device downloads larger images than it will ever need to display. Mobile phones, with their slower connections and limited processing power, perform poorly when accessing sites designed with desktop and laptop computers in mind. The speed of the mobile web today has been compared to the wired web of 1996 (Zakas, 2013).

This project evaluated the impact of three techniques to deliver appropriately-sized images on a responsive, image-heavy website. Load times and performance were measured on both desktop and mobile devices.

Need for the Project

All mobile internet users will benefit from a faster web if sites use tested techniques to minimize load times. Load time is especially important for mobile users because mobile connections are slower and less reliable than desktop connections. Mobile users also often pay for data usage, which can be minimized by making site assets like images as small as possible.

Any individual business who increases the speed of their site can expect to lose fewer customers. According to research conducted by Microsoft, users will visit a site less often if it is merely 250 milliseconds slower than a comparable competitor's website (Lohr, 2012). Another case study performed by Strangeloop found that a one-second delay in loading time on a mobile device decreased page views by 9.4% and decreased the site's conversion rate (number of page visitors who end up making a purchase) by 3.5% (Bixby, 2011). For businesses on the web, fractions of a second matter.

Significance of the Project

This project provided a detailed evaluation of the impact on performance of three responsive image solutions on an image-heavy website. Although there have been many articles exploring responsive image solutions, the researcher has found no examples of different solutions being tested on an otherwise identical site. This project allowed for a direct comparison of the tested techniques, and the results of that comparison can provide guidance to web developers that will be applicable to their own image-heavy sites.

Statement of the Problem

Responsive sites must accommodate desktop users, while also keeping page size to a minimum to accommodate the slower wireless speeds on mobile. One of the largest factors in page-load times is the size of images on the page (Yahoo! Developer Network, 2011), especially raster images (for example, a photograph saved as a jpeg), which can't be increased in size without losing resolution. The two most common solutions to this problem are:

- **CSS Media Queries:** Have separate mobile, tablet, and desktop assets, switching out which image is displayed using media queries in CSS. This method has several drawbacks, the largest being that because browsers pre-load images before processing the CSS at all, if the image tags are in the html the browser actually loads all sizes of the images, while only displaying the appropriate size (Alexander, 2013). This has a major negative impact on page-load times. Another drawback is that the images must be saved in several sizes, making site maintenance more difficult.
- **Fluid Images (Browser Re-sizing):** Make the browser re-size the images by giving the image a width in relative units (such as percentages or ems). This requires all devices to load the largest possible size of the image, and it is extra work for the browser to re-size the image, but this method prevents the browser from having to load extra images or to process additional media queries. It also allows for a flexible layout that looks great at any size (rather than and adaptive layout where the images "snap" between sizes when it hits a media query). A drawback of this technique is that it can cause problems if the images are animated, for example in a slideshow.

Another, less common technique is:

- **JavaScript:** Use JavaScript to detect screen size and load the appropriate images. The script can ensure that browsers don't load images larger than they will display. This technique adds the weight of the JavaScript file/library to the size of the site and may prevent the browser from pre-loading images as quickly as it usually would.

All three of these techniques increase page-load times compared to a static, non-responsive site.

Research Objectives: The researcher built three responsive, image-heavy sites, which were identical except for the method used for responsive images. Using these three sites the researcher evaluated:

1. Which of the three methods above deliver the fastest page-load time, on both desktop and mobile? This was measured using Akamai Mobitest and browser extensions for Chrome and Firefox.
2. Do each of the three methods above work for a wide range of devices, including older smartphones and older versions of Internet Explorer? This was tested on actual devices.
3. Which of the three methods above is the most practical to implement and maintain from a web developer's perspective?

Considering the answers to all of the questions above, the researcher determined which method is best for an image-heavy, responsive website, given the current state of the technology and browser support.

Limitations of the Project

There are many more solutions to the responsive-images problem than were tested in this project. Server-side and third-party solutions are not yet widely implemented, but they are promising. One example is Adaptive Images, which uses PHP to scale the images on the server and send the appropriate-sized assets (Wilcox, 2014). Server-side solutions require specific configurations on the server that were beyond the researcher's current budget and back-end expertise to implement for this project.

There are trade-offs involved in all responsive image solutions, and the appropriate solution will vary depending on the content of a site. On an image-heavy site it might make sense to include a JavaScript library to serve up the correct image sizes for each page. But on a site with few raster images the size of the JavaScript file itself and the time it takes the browser to parse the script might outweigh any benefit provided by loading (potentially) smaller versions of the images.

Page load times, which are being compared for this project, are affected by many factors outside of the site itself, and are not consistent even for the same site from page load to page load. It is therefore not possible to come up with a definitive page-load time for a website. The researcher has taken the average of three page loads for each site on each device measured in order to reduce the effect of random fluctuation on these measurements.

This project will not deliver an answer about which responsive image technique will result in faster load times and better performance in every situation. This project will explore and document performance in one specific example of an image-intensive website, which will have general implications for other similar image-heavy responsive websites.

Definition of Terms

Bandwidth: The maximum rate that information can be transferred. Measured in bits per second (Zakas, 2013).

HTTP Request: A request initiated by the web browser for a file from a server. To load a typical web page the browser must make many http requests for all of the assets that make up the page, including the html file, CSS stylesheets, JavaScript, and all images on the page. Each separate file requires an http request. Reducing the number of http requests required to load the page is an important way to reduce page loading time (Zakas, 2013).

High pixel-density device: A device that has more pixels per inch (ppi) than a standard desktop screen. Modern smartphones have at least two times the pixel density of desktop screens (Android, 2014). For example, and image set to 300px wide on a mobile screen could actually display 600px of detail or more if the device were given a high-resolution asset.

Latency: "The delay experienced between request and response" (Zakas, 2013). This delay is usually measured by round-trip time, the time it takes a given network packet to travel from source to destination and back. Latency is greatly affected by the distance a signal needs to travel.

Wireless networks have high latency compared to wired connections. Wireless signals also travel a very circuitous route to get from the user to the server, and delays can be experienced at these nodes (Grigorik, 2013). So even with a high network speed, there is a perceptible delay between user input (such as a mouse click, or form submission) and output (page response).

Media Query: A CSS3 feature that allows style rules to change based on the query condition. Media queries of screen width and height are heavily used in responsive websites (Breheny, Jung, & Zurrer, April 30, 2012).

Polyfill: Code (usually JavaScript) that allows features that you expect to work natively in modern browsers to work in older browsers (Sharp, 2010). For example, the HTML5 video element is not supported in older browsers, so you could use a polyfill which would test for the video element and load a video plugin like Adobe Flash if the video element isn't supported by the browser. Polyfills allow developers to take advantage of new elements and features without losing support for older browsers.

Raster Image: An image that is made up of a grid of pixels. Raster images are also known as bitmap images. Examples of raster image formats include jpeg, png, gif, and tiff. Raster images have a fixed resolution, and cannot be increased in size without visibly degrading the quality of the image. The majority of logos and icons on the web are raster images, in addition to the vast majority of photographs found on the web.

Responsive Website: A website that adapts to the size of the device it is being viewed on or the size of the display window. The size and placement of page elements change to take the best advantage of the available space and the type of display (for example, touch vs. mouse). A responsive website will deliver a good user experience across a wide range of devices. (Breheny et al., April 30, 2012)

W3C: The World Wide Web Consortium is an "international community that develops open standards to ensure the long-term growth of the Web."(The World Wide Web Consortium, 2014) The W3C writes the standards for the functionality of web technologies including html and CSS, and browser developers write their browsers to support those standards. The W3C, by publishing these standards, tries to ensure that there is a basic level of consistency in how web pages are interpreted and displayed by browsers.

Summary

Images on responsive sites must be large enough to accommodate desktop users, while also keeping page size to a minimum to accommodate the slower wireless speeds on mobile. The researcher built three responsive, image-heavy sites, and tested three methods of delivering appropriately-sized images to evaluate their impact on page-load time.

Chapter II: Review of Related Literature

Mobile Web and the Move to Responsive Design

Usage of mobile phones to access the web dramatically increased with the introduction of the iPhone in 2007 (Wrobelewski, 2012). Unfortunately, wireless connection speeds were generally slow, and the quality of the user experience was low. As recently as 2012, the mobile web was approximately 1.5 times slower than desktop (Jain, Tikir, & Grigorik, April 19, 2012).

One solution to this problem was to build a separate mobile site, which would have fewer features and be designed to be viewed on a small screen. Mobile sites use user-agent detection to redirect users to the mobile site if they are visiting the site on a mobile device. This was a popular option and considered a best practice for several years (Nielsen, 2012). Separate mobile sites have fallen out of favor for several reasons:

- The user-agent detection that sites rely upon to route visitors to the mobile site was never reliable (Lawson, April 19th, 2012). User-agent detection needed to be updated often to keep up with the vast array of devices capable of accessing the web, and directing a user to the wrong site could deliver a terrible user experience.
- Mobile connection speeds have improved (Jain & Tikir, 2013), making it more feasible to serve up the full version of the site.
- Separate sites make site maintenance much more difficult, because even simple content edits need to be made in more than one place.
- The redirect from the desktop site to the mobile site adds additional http requests as the browser is directed from the main site to the mobile site (Grigorik, 2013). Extra

http requests are one of the main culprits in slow-loading mobile sites, because of network latency overhead (Zakas, 2013).

Major browsers (Firefox, Chrome, Internet Explorer, Safari) began to fully support CSS3 media queries between 2009-2011, which made responsive web design possible (W3Schools, 2014a). Ethan Marcotte coined the term "Responsive Web Design" in an influential article in *A List Apart* in 2010 (Marcotte, 2010). Marcotte convincingly argued that with the explosion of different devices (smartphones, tablets, gaming consoles) that could access the web, it was not feasible to create separate sites targeted to specific devices or display sizes.

The current consensus in the web development community is that one responsively-designed website is preferable to having separate mobile and desktop websites (Breheny et al., April 30, 2012; Lawson, April 19th, 2012).

The Importance of Page Load Time and the Drawbacks of Responsive Design

Page load times have proven to be of utmost importance in studies of user behaviors, and even delays of less than a second can have impacts on user behavior and bounce rates.

According to research conducted by Microsoft, users will visit a site less often if it is merely 250 milliseconds slower than a comparable competitor's website (Lohr, 2012). Walmart studied the behaviors of site visitors and compared it to the load times those users had experienced on the site. They found that for every 1 second of improvement in page load time they experience a 2% increase in their conversion rate (Bixby, February 28, 2012). Another case study from Strangeloop found that a one-second delay in loading time on a

mobile device decreased page views by 9.4% and decreased the site's conversion rate (number of page visitors who end up making a purchase) by 3.5% (Bixby, 2011).

These studies demonstrate that page load times have a dramatic (although perhaps subconscious) impact on users. On large sites with heavy traffic and sales volume the 2% change in conversion rate experienced by Walmart in the study referenced above could equate to millions of dollars in revenue lost with one additional second of load time.

Unfortunately, responsive designs are not typically fast on mobile devices. One of the main culprits is the size of images on the site—mobile sites are often being asked to load images much larger than they can display on their small screens.

New Methods for Responsive Images

The Picture Element

The W3C has teamed up with the Responsive Images Community Group (RICG) to work on adding a solution to the responsive images problem to the HTML5 standard. Their stated goal is to create "a markup-based means of delivering alternate image sources based on device capabilities, to prevent wasted bandwidth and optimize display for both screen and print" (The Responsive Images Community Group, 2013). In 2012 this group published a draft proposal for a new `<picture>` element which will be able to contain a "source set" of several image files and rules about which image the browser should load and display based on window size, pixel density, and available bandwidth (The Responsive Images Community Group, 2013). Below is an example of the markup for a picture element with three possible image sources (a large, small, and fallback image):

```
<picture>
	<source
		media="(min-width: 800px)"
		srcset="images/backyard.jpg">
	<source
		media="(min-width: 500px)"
		srcset="images/flower-bed.jpg">
	<img
		src="images/flower.jpg"
		alt="My backyard garden">
</picture>
```

Once implemented in the majority of modern browsers, the `<picture>` element promises to be an excellent solution for loading the appropriate sized images without any extra http requests. However, as of this writing, 99.8% of browsers used in the U.S. do not support the `<picture>` element (A. Deveria, 2014b).

Picturefill

Picturefill is a JavaScript polyfill developed by Filament Group that enables the functionality of the `<picture>` element in browsers that don't natively support it (Jehl, 2014; Jehl, 2014).

Picturefill 2.0 was released as an alpha in early 2014, and the markup needed to implement it closely resembles what the `<picture>` element is expected to look like once it is finalized by the W3C. This will allow for a smooth transition from using Picturefill to using the natively supported `<picture>` element (Wright, 2014).

Picturefill 2.0, like the proposed `<picture>` element, also supports an additional 2x image option. If a 2x image is declared in the srcset, that (larger) image will be loaded on devices that have high pixel-density screens, as smartphones do.

Chapter III: Methodology

Building the Sites:

The researcher built three responsive, image-heavy sites, which were identical except for the methods used for responsive images.

- Browser Re-Sizing Method: http://audreybrockhaus.com/responsive-images/browser/responsive.html
- CSS Media Queries: http://audreybrockhaus.com/responsive-images/css/responsive.html
- Picturefill 2.0: http://audreybrockhaus.com/responsive-images/picturefill/responsive.html
    - A second version of the Picturefill 2.0 site was also tested, with the option to load higher-resolution assets for high pixel-density screens: http://audreybrockhaus.com/responsive-images/picturefill2x/responsive2x.html

According to Yslow, images account for the vast majority of total page weight of these sites (about 8850k out of the site's total of approximately 9050k). The images used were un-optimized jpgs in order to maximize their weight and accentuate any differences between the three tested methods. The images are photographs taken by the researcher. The site was built with four rigid breakpoints in order to minimize the need for the browser to resize images. This was done to test the effect of browser resizing on the performance of the site that uses browser resizing method, which needs to resize the images in any case.

Images on the site do not have “alt” tags. This was to prevent the browser resizing and Picturefill sites from being unfairly penalized on page weight for supporting this attribute. When images are loaded as background images in the CSS they don’t have alt tags, which is a problem that is discussed below.

The three methods tested were:

Using the browser to re-size large images: The researcher used images at the largest possible resolution, and gave the images a flexible width by setting the image width to 100%, letting their size be determined by the containing div. The browser scales the images to fit.

Switching out images using media queries: The researcher saved mobile, tablet, and desktop images in sizes appropriate for each. These images were displayed depending on display size using CSS media queries. All images default to the smallest size and then scale up from there depending on the media queries to prevent the browser from downloading extra images or larger images than are needed.

Picturefill 2.0: The researcher used the `<picture>` element with Picturefill 2.0, which serves up the appropriate images from the server for the current display size. Picturefill, like the CSS method, requires that images are saved in several different sizes. It also requires a small (7kb) script.

Two Picturefill sites were created—one basic one for the purpose of a fair comparison with the other two methods, and one that used Picturefill’s ability to query the pixel density of the device and serve up higher-resolution images to high pixel-density screens. This second Picturefill site will be referred to from here on as “Picturefill 2x” for brevity’s sake.

Testing the Sites:

The researcher tested four aspects of each of the three pages:

Page Load Time: Page load times were measured and analyzed using online tools:

- Mobile load times were measured using Mobitest: http://mobitest.akamai.com/m/index.cgi
- Desktop loading times were measured using browser extensions for Firefox and Chrome
- Yahoo's YSlow was used to count http requests and determine the relative weight of page elements: https://developer.yahoo.com/yslow/

Load times were measured three times for each site, and the average was used, to minimize random fluctuations of page load times.

Overall Page Weight: Page weight is simply the size of all of the files required for the page. This was measured from the server. The percentage of the page weight that is made up of images was also measured by Yahoo's YSlow browser extension.

Support: The researcher loaded each of the three pages on a variety of devices and browsers, testing whether the image scaling works. This was measured on a pass/fail basis against a matrix of browsers and devices, including versions of popular desktop browsers (Chrome, Firefox, Internet Explorer), Android phones, and iPhones 4 and 5.

Implementation: The researcher recorded the amount of time, starting from the base (static) version of the site, it took to implement each responsive image solution. The researcher also provided qualitative data about implementation of each solution from the perspective of a developer.

Chapter IV: Results of the Project

1. Picturefill 2.0 delivered the fastest page-load time on mobile devices, CSS Media Queries were faster on desktop

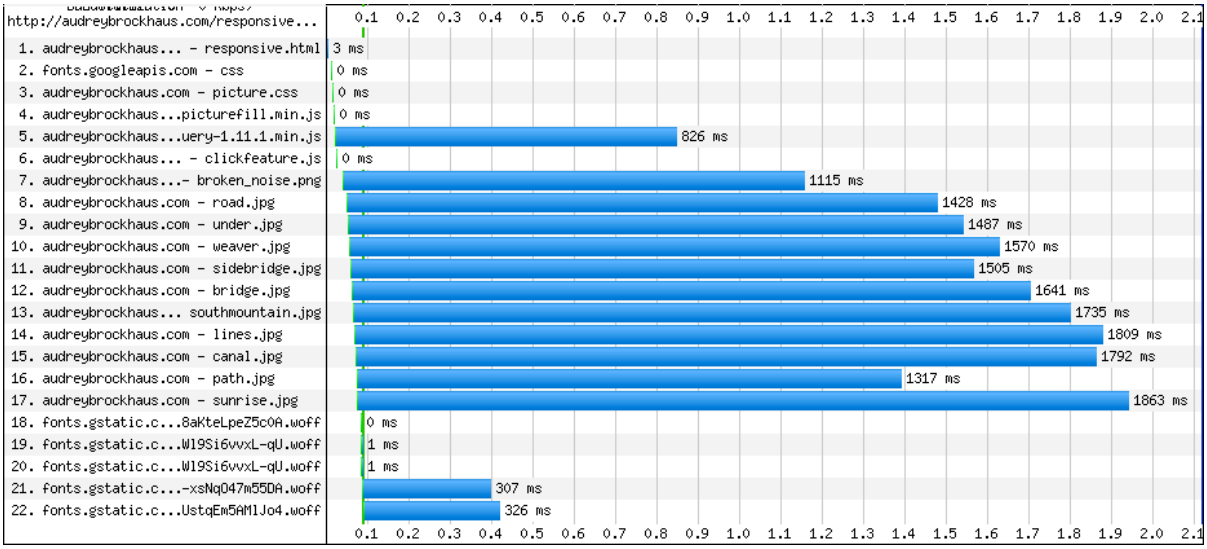
Table 1 shows page load times recorded on each site on 5 different devices. Each number is the average of 3 measurements. The fastest load time for each device is highlighted in pink. Methods that delivered high-resolution images to high pixel-density screens are highlighted in gold. Unsurprisingly, the two methods that did not deliver high resolution images (the CSS media query method and the Picturefill method without the optional 2x attribute) were the fastest to load on the mobile devices tested. This is a speed/quality tradeoff that must be weighed by each developer.

Table 1. Page load times for sites using different methods to load responsive images

	Browser	CSS	Picturefill 2.0	Picturefill 2.0 2x
Mobile: iPhone 5, iOS7, AT&T	7.83s	2.57s	2.37s	3.63s
Mobile: NexusS Android 2.3	8.46s	2.98s	1.86s	1.90s
Mobile: iPhone 4S	16.62s	3.08s	2.91s	7.31s
Desktop Loading Time: Chrome 37.0	2.39s	1.25s	2.31s	2.40s
Desktop Loading Time: Firefox 27.0	2.85s	1.05s	2.18s	2.06s

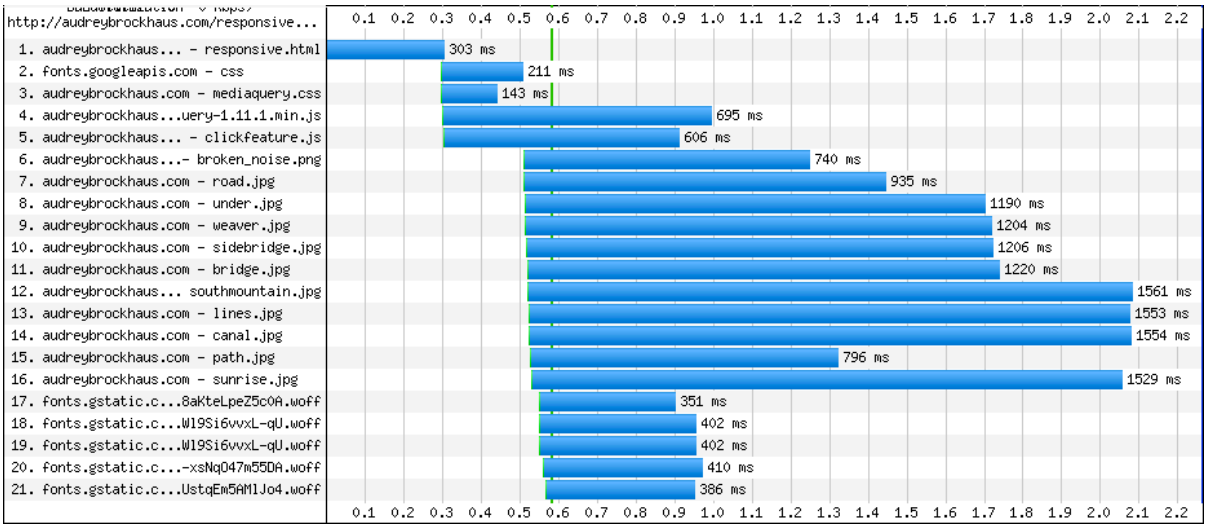
Looking in more detail at the loading times for page assets provides insight into why the Picturefill method is fastest on mobile devices. Below are waterfall charts for each of the four sites loading on the iPhone5 through Mobitest. Notice that the images on the Picturefill site begin to load within 0.1 seconds, as the html is being loaded. The images on the CSS media query site don't begin to load until the CSS is finished loading at 0.5 seconds.

Figure 1: Page load waterfall chart for the Picturefill method on an iPhone5 (Mobitest)



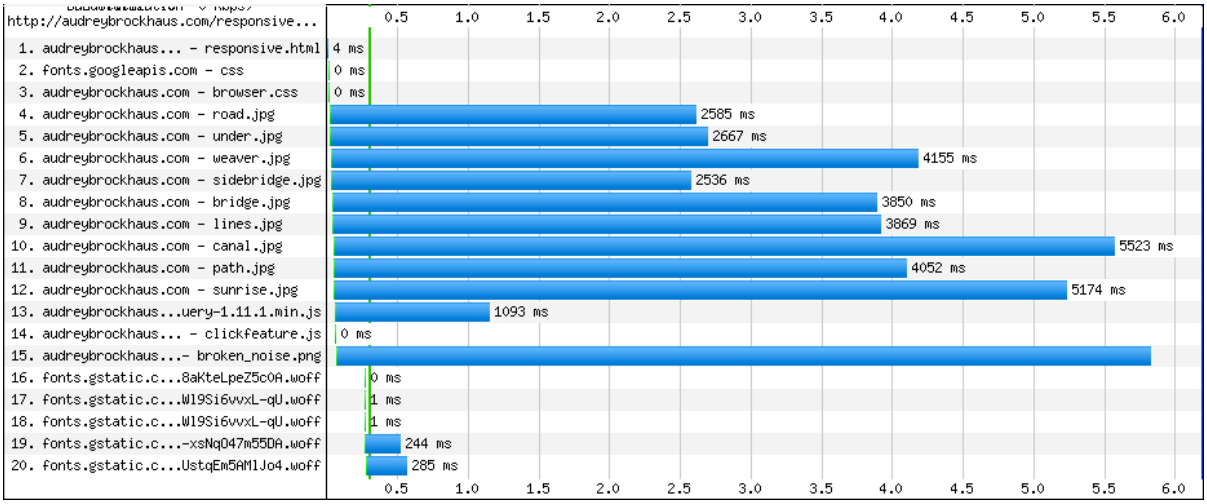
By allowing the browser to choose the appropriate image for the display size within the html itself, the `<picture>` element allows the browser to pre-load images in the html (as it normally would) without pre-loading unnecessarily large images. The script needed to implement Picturefill 2.0 (picturefill.min.js) had a negligible effect on load time, taking 0ms to load.

Figure 2: Page load waterfall chart for the CSS media query method on an iPhone5 (Mobitest)



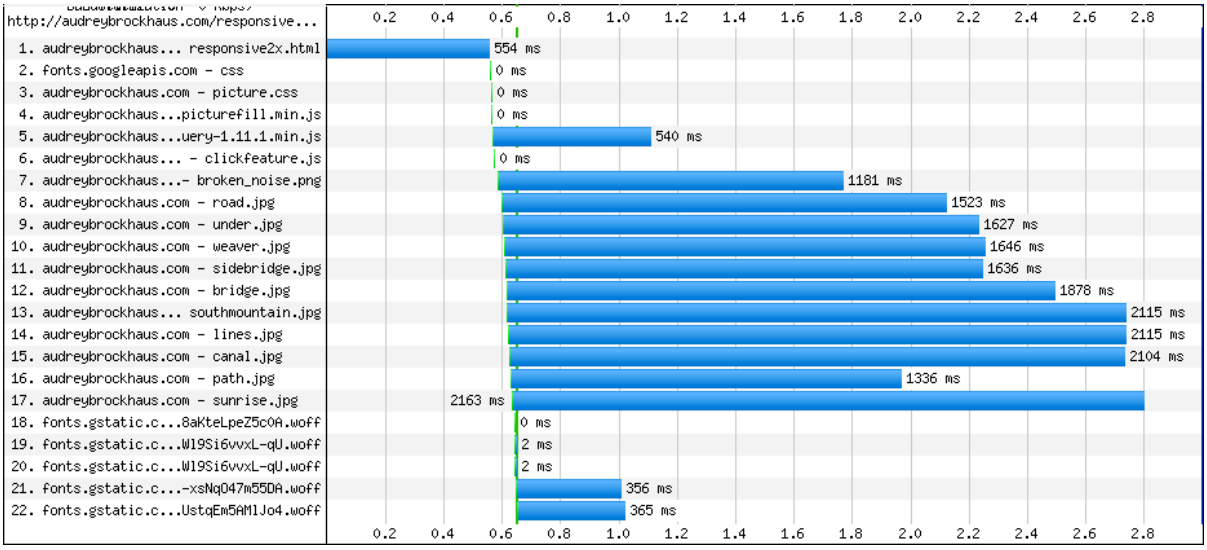
Notice above that each image is loaded only once, in one size. Using CSS media queries to switch out images often results in all sizes and versions of an image being loaded (Alexander, 2013). This problem was avoided in the construction of this site by keeping all of the images in the CSS and having the CSS default to mobile-sized assets and using media queries to scale up from there if necessary.

Figure 3: Page load waterfall chart for the browser re-sizing method on an iPhone5 (Mobitest)



Images on the browser re-sizing site begin to load as quickly as images on the Picturefill site. However, they take much longer to load because they are much bigger files (the browser re-sizing method takes the highest/desktop resolution of an image and scales it down).

Figure 4: Page load waterfall chart for the Picturefill 2x method on an iPhone5 (Mobitest)



As expected, the Picturefill 2x method is slower than the Picturefill method, because larger images are being downloaded. But interestingly, about half of the longer page-load time of the Picturefill 2x method was caused by the browser taking much longer to load the html. This suggests that the queries of device pixel density in the `<picture>` tags are themselves taking up a considerable amount of load time.

2. Compatibility/Browser Support

Internet Explorer 8 did not correctly size the images on any of the four sites. .Internet Explorer 8 is known to provide poor support for image re-sizing and background images in general (A. Deveria, 2014a). Otherwise, each of the four sites displayed and performed as expected on each of the browsers and devices they were tested on, except for the CSS media query method. On Safari the featured image (the large image at the top of the page) was very slow to load on the CSS site when it was changed by clicking on one of the smaller images. There was no perceptible delay in Safari on the other three sites.

According to W3Counter, as of September 2014 3.66% of internet users use Internet Explorer 8, and 10.60% use Safari (W3Counter, 2014). Because the CSS site is basically functional in Safari, the lack of support for IE8 is the primary concern with these four responsive sites.

Table 2. Browser and device support for sites using different methods to load responsive images

	Browser	CSS	Picturefill	Picturefill 2x
Chrome 37.0	Pass	Pass	Pass	Pass
Firefox 27.0	Pass	Pass	Pass	Pass
Safari 7	Pass	Marginal Pass	Pass	Pass
Internet Explorer 8	Fail	Fail	Fail	Fail
Internet Explorer 9	Pass	Pass	Pass	Pass
Internet Explorer 10	Pass	Pass	Pass	Pass
iPhone 5 iOS7	Pass	Pass	Pass	Pass
iPhone 4	Pass	Pass	Pass	Pass
Galaxy S2, Android 2.3	Pass	Pass	Pass	Pass

3. Developer's report on site construction, performance, and maintenance:

Browser Re-Sizing Method: Easy, simple, and slow

The browser resizing method was the easiest method to implement, taking only 3 hours to implement starting from a static, non-responsive site. This is reflected in the size of the stylesheet for this site, which is only 125 lines, compared with 282 lines for the CSS media query method.

Images were only saved in one folder and at one size. Any file 1232 x 819 pixels or larger could be dropped in the images folder and work within this layout without being resized. When the browser resizes the images, only one version of each picture needs to be saved on the server. This accounts for the fact that the browser resizing method had the

smallest total size on the server (9.51MB, compared to 13.8MB for the CSS media query method and 11.7MB for the Picturefill methods).

The researcher also anticipates that site maintenance would be easy and nearly self-explanatory on the browser resizing site. There are three classes of images, .one (for images meant to span an entire row at desktop size), .two (for images meant to span half of a row at desktop size), and .three (for images meant to span a third of a row at desktop size). To add new images to the site all that would need to be done would be to put the image tag into the html and apply the desired class to its container. No edits to the CSS or JavaScript files would be necessary.

Images on mobile devices were sharper using this method than the CSS method. That is because mobile screens have higher pixel densities than desktop screens, and will display more detail than the media query would lead you to believe if a large version of an image is sized down for the mobile screen.

Browser resizing did not appear to impact the performance of the featured image functionality.

However, none of these benefits make up for the unacceptable loading times experienced when testing the site on mobile.

CSS Media Query Method: Complicated, fragile, low-resolution, and unsemantic

The CSS media query method took the longest to develop—six hours starting from a static, non-responsive site, which is twice as long as the browser-resizing method took to develop. In addition to each image needing to be saved out in four sizes, the stylesheet and

JavaScript for the featured image functionality were all more complicated than either the Picturefill or browser re-sizing methods.

Different sizes of images could be saved directly in the html and manipulated using 'display: none' in the CSS, hiding the unneeded images depending on display size. This would have been easier from a development perspective and would have used CSS media queries to display the appropriate image for the display size. However, because browsers pre-load images before rendering the CSS, all versions of the image would have been loaded before the CSS was rendered (Alexander, 2013), which would have been disastrous for page load times. So instead, images were set as background-images in the CSS itself.

Loading images in the CSS as background images and defaulting to the smallest version of the images speeds up page load compared to browser resizing because it prevents the page from pre-loading images larger than the screen requires. However, it has several drawbacks from a development perspective:

- *Search Engine Optimization and Accessibility:* CSS background images aren't represented in the html of the page. This means that none of the images have "alt" attributes, which search engines use when indexing sites (Google, 2014). Alt attributes are used for screen readers to describe the contents of the image for visually impaired users. They are also meant to give the user a description of the image's content if the image fails to load.
- *Images are not nodes in the html, and therefore can't be cleanly or easily manipulated by JavaScript:* The slideshow on the browser resizing and Picturefill sites will work without being edited, no matter how many additional pictures are added to the page, or where the images are saved, because it is manipulating the html

elements in the page itself. On the CSS site, new images must be added in four sizes, using specific filenames, and saved in specific folders. Any changes or additions to the page will require significant time and effort editing the CSS, HTML, and JavaScript in order to work.

The CSS media query method also had an important drawback from a performance perspective, in that it did not deliver high-resolution images to high-pixel-density devices. Although the effect is subtle, there is a noticeable difference in image quality between the CSS media query and the Browser-resizing method.

Picturefill 2.0: Verbose, but easy

The Picturefill method took 4 hours to implement starting from a static, non-responsive site. Like the CSS media query method, the Picturefill method required that images be saved out in several different sizes. It also required that `the <picture>` element and the paths to each of the alternate images be entered into the html. The html file for the Picturefill method is larger than the other three methods, at 153 lines, because each `<picture>` element contains an array of image sources. Downloading the Picturefill JavaScript file and adding it to the page took only minutes. The JavaScript and CSS files were simple, and comparable to the browser resizing method.

Picturefill also gives developers the option to serve up high-resolution images for high-resolution displays. This choice can be made on an image-by-image basis, so important images can have a high-resolution option, while other things, like a simple icon, can be sent at the lower resolution.

The developer anticipates that site maintenance for the Picturefill site would be easy. Like the Browser resizing method, images can be added to the Picturefill site without editing

the CSS or JavaScript files. The additional difficulty of adding the `<picture>` tag and the several different image sources is more than made up for by the gains in page speed displayed in the tests above.

Chapter V: Conclusions and Recommendations

Conclusions:

1. The Picturefill 2.0 method was fastest on mobile, CSS media queries were fastest on desktop.
2. The browser re-sizing site took up less space on the server, but this did not translate into faster page-load times.
3. Browser support for all three methods is generally good, although there were some problems with the CSS media query method.
4. The browser re-sizing method was easiest to implement, but Picturefill 2.0 implementation was also fast and easy

Picturefill 2.0 is the fastest way to deliver responsive images to mobile devices on the site constructed for this project. Based on the results of the tests conducted on the three responsive websites, the researcher would strongly recommend the use of Picturefill 2.0 on image-heavy sites until the majority of browsers implement native support for the html5 `<picture>` element.

Many developers default to the browser re-sizing method. Not only is it by far the easiest to implement, but it also makes intuitive sense that the simplest method with the smallest weight on the server would also be the fastest to load. Unfortunately, as this project has demonstrated, this assumption is dead wrong. The browser resizing method delivered by far the longest load times on mobile devices in the tests conducted for this project.

Although inserting images as background images in the CSS delivered the fastest desktop load time in this experiment, because of the difficulty of implementing and

maintaining this structure, the low resolution on high pixel-density displays, the invisibility of images to search engines, the difficulty of manipulating images in JavaScript that are not nodes in the html, and the impact on accessibility for visually impaired visitors, the researcher would not recommend this method.

Recommendations for future research:

Alternatives to raster images:

Icons, logos, and other simple images can be displayed as vector images instead of raster images, avoiding the responsive resizing problem altogether. The performance and browser support for SVGs and other web vector formats is an important area to investigate.

Other methods for responsive raster images:

There are server-side methods for detecting device display size, re-sizing images on the fly, and serving up appropriate images for each device. These methods are less common and less practical to implement for a developer than the three methods studied here, but may be worthwhile for large corporate sites if they do deliver faster load times. The performance of server-side solutions would be interesting to compare to the front-end solutions studied here.

CSS Image Sprites:

An image sprite is several images put into a single large image (W3Schools 2014b)—which part of the large image is shown is controlled by manipulating the element size and background position in CSS. CSS image sprites are not appropriate for a collection of large feature images like those on the sites tested in this project, but they are an important way to reduce http requests for icons and other functional images. As discussed above, http requests

are a major reason that mobile sites are slow, because of wireless network latency. All four sites tested in this study need either 17 or 18 http requests. It would be interesting to test the effects of reducing http requests using sprites on page-load times.

References

Alexander, S. (2013). **Choosing A responsive image solution.** ***Smashing Magazine***. Retrieved from http://www.smashingmagazine.com/2013/07/08/choosing-a-responsive-image-solution/

Android. (2014). **Display metrics.** Retrieved from http://developer.android.com/reference/android/util/DisplayMetrics.html

Bixby, J. (2011). **Case study: The impact of HTML delay on mobile business metrics**. Retrieved from http://www.webperformancetoday.com/2011/11/23/case-study-slow-page-load-mobile-business-metrics/;

Bixby, J. (February 28, 2012). **4 awesome slides showing how page speed correlates to business metrics at Walmart.com**. Retrieved from http://www.webperformancetoday.com/2012/02/28/4-awesome-slides-showing-how-page-speed-correlates-to-business-metrics-at-walmart-com/

Breheny, R., Jung, E. & Zurrer, M. (April 30, 2012). **Responsive design – harnessing the power of media queries**. Retrieved from http://googlewebmastercentral.blogspot.jp/2012/04/responsive-design-harnessing-power-of.html

Deveria, A. (2014a). **Can I use background image?** Retrieved from http://caniuse.com/#search=background image

Deveria, A. (2014b). **Can I use picture?** Retrieved from http://caniuse.com/#search=picture

Google. (2014). **Webmaster tools: Image publishing guidelines.** Retrieved from https://support.google.com/webmasters/answer/114016?hl=en

Grigorik, I. (2013). **Optimizing the critical rendering path for instant mobile websites**. Retrieved from https://www.youtube.com/watch?v=YV1nKLWoARQ&feature=youtu.be;

Jain, A., & Tikir, M. M. (2013). **Is the web getting faster?**. Retrieved from http://analytics.blogspot.com.ezproxy1.lib.asu.edu/2013/04/is-web-getting-faster.html

Jain, A., Tikir, M. M., & Grigorik, I. (2012). ***Global site speed overview: How fast are websites around the world?***. Google Analytics Blog. Retrieved from http://analytics.blogspot.com/2012/04/global-site-speed-overview-how-fast-are.html

Jehl, S. (2014). **Picturefill: A responsive image polyfill.** Retrieved from http://scottjehl.github.io/picturefill/

Lawson, B. (April 19th, 2012). **Why we shouldn't make separate mobile websites**. Retrieved from http://www.smashingmagazine.com/2012/04/19/why-we-shouldnt-make-separate-mobile-websites/

Lohr, S. (2012, February 29, 2012). **For impatient web users, an eye blink is just too long to wait**. *The New York Times*, pp. A1.

Marcotte, E. (2010). **Responsive web design**. Retrieved from http://alistapart.com/article/responsive-web-design

Nielsen, J. (2012). **Mobile site vs. full site**. Retrieved from http://www.nngroup.com/articles/mobile-site-vs-full-site/

Sharp, R. (2010). **What is a polyfill?** Retrieved from http://remysharp.com/2010/10/08/what-is-a-polyfill

The Responsive Images Community Group. (2013). **Responsive images community group.** Retrieved from http://www.w3.org/community/respimg/

The World Wide Web Consortium. (2014). **W3C.** Retrieved from http://www.w3.org/

W3Counter. (2014). **September 2014 market share**. Retrieved from http://www.w3counter.com/globalstats.php

W3Schools. (2014a). **CSS3 @media Query**. Retrieved from http://www.w3schools.com/cssref/css3_pr_mediaquery.asp

W3Schools. (2014b). **CSS Image Sprites**. Retrieved from http://www.w3schools.com/css/css_image_sprites.asp

Wilcox, M. (2014). **Adaptive images.** Retrieved from http://adaptive-images.com/

Wright, T. (2014). **Picturefill 2.0: Responsive images and the perfect polyfill**. Retrieved from http://www.smashingmagazine.com/2014/05/12/picturefill-2-0-responsive-images-and-the-perfect-polyfill/

Wrobelewski, L. (2012). **Mobile first.** Retrieved from http://aneventapart.com/news/post/video-luke-wroblewski-mobile-first-live-at-an-event-apart;

Yahoo! Developer Network. (2011). **Best practices for speeding up your web site**. Retrieved from https://developer.yahoo.com/performance/rules.html

Zakas, N. C. (2013). **Building web sites that perform well on mobile devices remains a challenge.**. *Association for Computing Machinery, February 17, 2013*