Responsive Design

Glen Salmon May 2012

Synopsys

Early responsive design is intended to address content layout however, the same concepts may be applied to user experience. The principals of responsive design are easy to understand, the implementations are more difficult. The difficulty comes from the introduction of complexity in the development of responsive layouts and from the current deficiencies in the standards for markup and style. The complexity is partially mitigated with the creation and use of well structured frameworks. However, discipline is imperative to avoid fragmenting the framework and introducing bugs which are more difficult to identify in responsive web implementations. The deficiencies in the markup and style implementations are mitigated with various methods including dynamic style languages, JavaScript to override image selection, and server-side detection and substitution techniques.

This paper separates designs into two categories - content-centric and interaction-centric designs. The general rule is that style based responsive design is well suited for content centric use cases whereas web application frameworks focus on the interactions of the application use cases. There is overlap between content layout and web applications and the fundamental difference is in how each addresses and controls layout.

Desktop vs Mobile Design

Traditional desktop web layout has been dominated by either fixed width or fluid width pages. Often, fixed width is chosen for rich layouts - aka images, columns, strict control of flow and word wrap, and where deterministic positioning is important. The popularity of fixed layouts is due to the control it provides in knowing the end result of both the visual content and the navigational controls. Fluid widths are more difficult to create and less predictable when it comes to how text and graphics will flow. A good fluid design is not easy and almost always imperfect for some percentage of the content.

Historically, designing for mobile devices was based upon traditional web design techniques. Content layout was tailored to device specifications – a desktop browser experience, a netbook, a Blackberry 9000 experience, etc. Often, each experience was individually developed, tuned, and tweaked.

What Change?

From a statistical significance point of view, mobile device usage, for the purposes of accessing the web, has doubled each of the past three years and is predicted to surpass desktop browser usage within two years.

With the rapid growth of mobile devices, web content needs a layout solution that supports both desktop and mobile users.

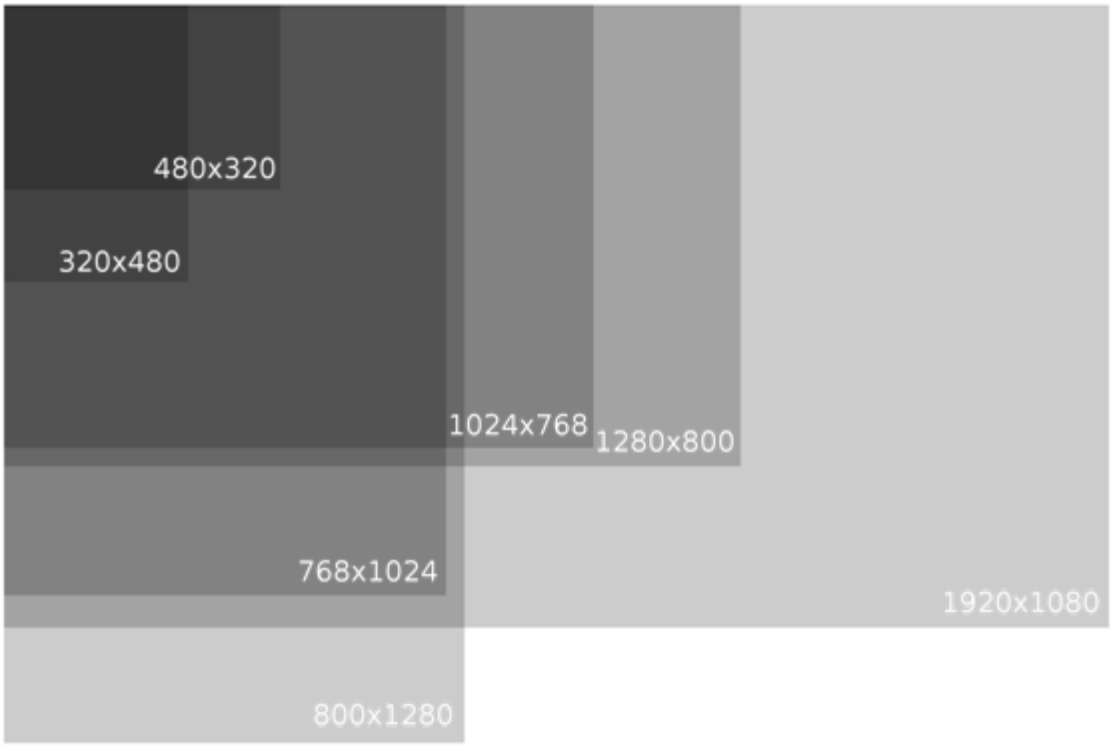
Recently, the demand for mobile device support has started to influence traditional design philosophies. Two (non exclusionary) philosophies are "*Mobile First*" and "*Responsive design*" - both addressing the need to support mobile devices as part of content layout and delivery.

Responsive design for Content

Responsive Web Design for Content focuses on the desire to support available screen real estate across a very broad range - often without exact knowledge of the target screens. The same can be said of fluid width layouts. However, unlike fluid layouts, responsive design retains the control of content flow provided by fixed width layouts. The impetus behind responsive design is the dramatic range of screen real estate, for example from a 320 pixel mobile phone to a 1920 pixel HD monitor.

Note: Smaller and larger screens exist but these limits make for practical boundaries within the scope of a design conversation.

Responsive design breaks the continuum of screen real estate into a reasonable number of "thresholds". A common set of thresholds include 960 (screens), 768 (tablet portrait), 480 (mobile landscape), and 320 (mobile portrait). As the device landscape evolves, these thresholds will change or have additions.



There can be any number of thresholds. The correct number and their values is determined as part of the design process. Too many thresholds will add complexity during the development phase. Too few thresholds will result in excess wasted screen real estate.

A number of "grid frameworks" are available as a template for responsive design. The most common grid layouts are based on 960 pixels. There are a few as large as 1440 pixels.

In responsive design, the layout of content is wire framed at each of the planned thresholds. Through the use of CSS media queries, layouts are defined as one or more columns. Grid layouts tend to use 16 or 12 columns. The columns are a divisible portion of the largest "minimum" defined, for example 960. In most cases, there are also whitespace gutters applied. At each step in the list of "minimums", the widths and offsets of the columns are recomputed. Each of these steps is represented with a media query.

Content in a column is contained in a display block and is "floated" from the left.

When the computed size of columns is below a usable threshold, the offset is removed and the column width is increased to the threshold. In more elaborate cases, groups of columns may be transitioned. This is best understood by an example.

For documentation purposes assume our example has an upper limit of 960 pixels; only 4 columns; and a usable threshold of 160 pixels.

page width is 960, Gutters and margins width are 10 pixels = 4 columns each width is 227 pixels

Column 1	Column 2	Column 3	Column 4

page width is 768, Gutters and margins width are 10 pixels = 4 columns each width is 179 pixels

Column 1	Column 2	Column 3	Column 4

In both of these cases, column #1 has an offset equal to the left margin; column #2 has an offset equal to the left margin + column #1 + the gutter, column #3 and #4 progress in similar fashion.

page width is 480, Gutters and margins width are 10 pixels = 4 columns each width is 107 pixels

Column 1	Column 2	Column 3	Column 4

At this point, the column width is too small.

When the available screen width drops to the 480 threshold, the responsive layout keeps column #1 and #2 as they are but changes the offset of column #3 to be the same as column #1 and then column #4 matches column #2. Thus, now we only need to fit 2 columns on the page.

page width is 480, Gutters and margins width are 10 pixels = 2 columns each width is 225 pixels

Column 1	Column 2
Column 3	Column 4

Changing the Paradigm

Responsive design solves many of the shortcomings of both fixed layouts and fluid layouts. In doing so, it makes it possible to design a web experience that works well for desktop browser users, tablet users, and smartphone users.

However, it only addresses layout and it adds design work and development complexity. Responsive design requires that mobile user experience be part of the initial design practice. It is more work to convert a fixed or fluid layout to Responsive design than it is to implement Responsive design from the start. This is one of the motivators for the "Mobile First" philosophy. It is safe to assume additional tools and education are required to accomplish Responsive design.

There are frameworks such as 960 Grid and dynamic CSS options such as LESS and SASS. These provide structure to both the design and development of Responsive design.

Navigation and Control

Modern navigation and controls are best implemented using JavaScript and one of the well supported frameworks such as Dojo, jQuery, Sencha, etc. This section is worth a paper of its own and not the focus here.

Web Content Layout vs Web Applications

There is an important distinction between content layouts and applications. While there is content within applications, the richness and complexity of web applications demand more than responsive design. Obviously applications and content are different but making a determination as to which a given project resembles is seldom black or white. Here are a few questions to ask:

- Does the web project have "navigation" vs "controls"?
- Does content change dynamically?
- Is it bad if content scrolls out of view?
- Does important information fall "below the fold" (aka requires scrolling to view)?

When building a web application it is best to decide what framework will be used. The framework will often include layout control. This may eliminate the need to develop a responsive design in favor of the web application framework capabilities. This section is worth a paper of its own and not the focus here.

"Some" Best Practices in Responsive design

Here are a few (*just a few*) recommended practices when creating web layouts with Responsive design.

Wire-frame the Layout Design

Before any development can begin, there needs to be design work. Before the design work can begin, there needs to be an idea of the layout. For modelers, this is the "meta-design". Start by creating simple wire-frame boxes of the various content, UI control sets, navigation, etc. Layout the wire-frame boxes at the smallest planned threshold. Finally, layout the wire-frame boxes at the largest planned threshold.

The results of the two extreme layouts will provide important understanding of how the layout of both content and controls needs to change across the range of thresholds. From this work, the intermediate thresholds can be worked using the wire-frame boxes.

This is often an iterative process. It is extremely helpful to have designers comfortable with "interactive design" or "prototyping".

Interactive Design

Designers need to get away from "Photoshop as Architecture". This is not strictly a Responsive Design change. However, responsive design is focused on how layouts change with changes in screen real estate. Static images are deceptive since they have no behavior. It is not necessary for designers to become developers, but it is necessary for designers to become fluent in CSS and HTML and the generation of layout tools which generate these.

Device Agnostic but Device Aware

When determining the thresholds for your responsive layout, avoid implementing for a specific device. Being aware of families lets you find optimal thresholds that group devices. Think of these as classes of devices. For example: portrait smartphone, landscape smartphone, portrait tablet, desktop, widescreen, etc. As new classes of devices appear – such as the super-phone or tablet-phone – a new threshold may be needed but the layout will likely match with one of the adjacent layouts – eg, number of columns, positioning, etc may stay the same with only a new width.

Control layout scaling

By default, mobile devices attempt to make web pages look "as good as they can" on the mobile screen. They use a combination of anti-aliasing and viewport scaling. This will conflict with your control of layout in responsive design. It is most noticeable when looking at a page on a mobile device and then rotating the device. The content will change size rather than responding to the new screen width. You solve this by setting the viewport scaling.

```
<meta name="viewport" content="user-scalable=no, width=device-width minimum-scale=1.0, maximum-scale=1.0" />
```

Establish You Own Framework

There are a number of pre-built responsive grid frameworks. However, most of these will take as long to learn as the time needed to build your own framework. Building your own framework permits tailoring to design requirements. For example, most grid frameworks assume all columns are uniform, gutters exist between columns, columns have padding, etc. These assumptions may not work based on design requirements.

For grid layouts, the current standard is 960 pixels wide for the maximum. Given the number of 19:6 tablets sporting 1280 pixels landscape and the low cost of 1920x1080 pixel monitors, it is advisable to support or at lest plan for wider upper limits. A suitable set of thresholds might eventually include 320, 480, 768, 960, 1180, and 1800 pixels. However, the implementation of a 5x range suggests additional complexity in how and when columns collapse.

There is one exception to the "build your own" recommendation. If you are on a common UI platform - eg as a portal server such as WebSphere Portal or Tivoli Integrated Portal - it would be beneficial if that platform provided a Responsive Design layout option.

Use a CSS loader rather than embed all styling into a single file

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)" href="style480.css" />
```

Often, all of the CSS is created in a stylesheet such as "style.css" with the defaults at the top and them a series of media queries near the bottom with each one progressively changing the styling. A more efficient implementation is to have a "loader" CSS that moves the media query content into individual stylesheets. This may not have a significant

advantage for small web projects but as complexity grows and as CSS incorporates more and more image files; this practice eliminates loading unused content.

It is best to define major and minor break points. The major break points address layout while the minor break points address styling.

Use LESS or other Programmable Style Syntax

All responsive design has a base style and then alters that style based on the available screen real estate. Many of the style alterations are derived computationally. Using a stylesheet language like LESS reduces the complexity of the code, improves comprehension, and reduces bugs. LESS can be used dynamically or as a pre-compiler. The latter lends itself to the CSS loader method described above. Another option is SASS. However, LESS does not require special tools and supports dynamic CSS, which is helpful during development, as well as compiled CSS which integrated into build processes and can be minified for production and release.

Avoid "display: none;"

The use of the display style "none" is an easy way to remove content from the layout. However, it does not remove the content from the data transmission. Further, if the hidden content required JavaScript, which too would be downloaded and not used. For small amounts of text, "display: none;" is an easy option but for large content and especially for images, it is suboptimal.

Responsive Images

Images are the least mature element of responsive design. Currently, neither HTML nor CSS support dynamic determination for image resources. Standards groups are discussing solutions but it will be some time before there is consensus and browsers have implementations.

Caveat: There are a number of options for addressing images in a responsive design. Consider these temporary fixes and plan ahead to transition to alternatives as standards progress and better solutions are developed.

Using standard media queries will allow for images to be scaled but the entire image is still downloaded. This can be very inefficient.

> For documentation purposes assume our example header image spans the entire width at each threshold of the responsive design and the height remains proportional, being at most 300 pixels high.
>
> Sample Banner 960x300 PNG = 248KB
> Sample Banner 320x100 PNG = 38KB
>
> Using CSS to change the display size still results in the download of the entire image resource. However, if the choice of image resource is changed prior to it being downloaded, there is dramatic savings.

There are several options currently available for determining the correct images for the given threshold. Here are two options - a very simple and a very rich solution.

Option A - "Images do not change [significantly] for different thresholds"

*G*enerate the images at or near their native size and then apply CSS scaling for any small amount of adjustments.

Option B - "There is no pre-known awareness of images"

This is the most flexible and comprehensive option. It has three parts.

1) Add a snippet of JavaScript to create a cookie that holds the current screen resolution.

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+'; path=/';</script>
```

2) Create a server interface mechanism to retrieve (or generate and cache) the correct resource.

```
http://www.domain.com/image.php
```

Within the interface, the code must retrieve the original resource URI and the cookie which indicates the desired resolution. This information is used to test if a matching resource is already in the cache and return it. If no resource exists, it is generated, stored in the cache, and the result is returned.

3) Use .htaccess to redirect all image resource request through the server interface mechanism

```
RewriteEngine On
RewriteCond %{REQUEST_URI} !a-special-directory # ignore some directories
RewriteCond %{REQUEST_URI} !the-cache # ignore the cache directory
RewriteRule \.(?:jpe?g|gif|png)$ out-interface-mechanism.php
```

The "B" option provides the correct size and does not require advanced knowledge of the images. The server side implementation generates any image request at a given size and then caches it for future requests. In a multi-person development team, this eliminates the need for developers to change from standard resource markup.

Responsive images are a great example of the division of processing labor between the client and the server. The demands of one client can be handled and cached on the server so that the client (and similar clients) will benefit on future page loads. This concept extends beyond image files. Once the capabilities for a device are found, they could be remembered (at least for the remainder of the session) so that time isn't wasted on each page checking for capabilities and building the page accordingly. The server could dynamically build and possibly cache pages based on the capabilities found and submitted by the client script.