Retinafy your web sites & apps

by Thomas Fuchs

Version 1.1
Saturday, April 13, 2013

Thanks for purchasing!

Thanks for grabbing a copy of my book. I plan to release infrequent updates which, next to fixing typos should contain new information if any major things come along (for example, it looks like that browsers should support image variants soon, but it's too early to include this as it's not supported by real-world browsers yet).

Your feedback is more than welcome!

Please write to thomas@slash7.com with any corrections, suggestions and ideas!

Licensing & cost

This ebook is not open source. I reserve all commercial and moral rights to the book, materials, and supplied code (where not under an existing license).

However, I have no interest in limiting or taking away your fair use rights to excerpt, review, criticize, and parody. Go right ahead!

Why this book costs money & why you should pay

I want to share my hard-won expertise with the world for a reasonable price so I can keep on creating free and cheap content

and blog posts, open source like Zepto.js and Microjs.com and of course witty rants on Twitter for everyone to enjoy.

Please buy a copy if you've received this ebook without paying.

It costs the equivalent of a movie ticket and popcorn, two trips to McDonalds or a few Starbucks drinks. I know it'll be worth more than that to your projects and your business.

Acknowledgements and Thank-You's

First and foremost, this book wouldn't be in your virtual hands if it wasn't for my awesome wife Amy Hoy, who suggested that I'd tackle this subject matter. I'd also like to thank all of you who wrote in with comments and suggestions about the book.

Also, a big thank-you to technical reviewer Thijs van der Vossen. He knows a ton about this stuff, and the best thing is that you can hire him and his team at Fingertips. They offer design and development services for Web, iPhone, iPad, and Mac OS X apps, and, besides custom consulting, they also have convenient fixed-price packages, including one to make your web app ready for retina resolution displays.

Version history

Version	Date	Notes
1.0	July X, 2012	Retinafy all the things!
1.1	April 13, 2013	Added note on old IE background-size support (or lack thereof) Corrected “webkitBackingStorePixelRatio” property name Fixed several typos and formatting problems Added version history

Table of contents

Why bother with retina screens? 7

Resolution vs. Density vs. Pixel Ratios 8

So what is “Retina” exactly? 9

CSS pixels and device pixels 11

Locking down the CSS pixel ratio 12

Force-mapping CSS pixels to device pixels 12

Detecting retina screens in JavaScript 13

Targeting retina screens in CSS 14

Testing on Retina if you don’t have a device 16

Further reading 17

Text & Fonts 18

CSS backgrounds and units 20

Backgrounds and high-resolution images 20

Internet Explorer (before version 9) 22

Non-integer pixel ratios and CSS units 23

Bitmapped images 25

PNG 25

JPEG 31

GIF 35

Animated GIFs 35

Favicons 36

Touch icons 39

Touch startup images 40

Procedural and vector graphics 43

Canvas 43

Vector graphics (SVG) 46

Optimizing image file sizes 51

What to do when things go wrong 53

Bad scaling quality 53

Jagged edges 55

Icon fonts 57

JavaScript “retina” helper libraries 59

Quick Retina reference 60

Case Studies 61

Charm Customer Support 61

Every Time Zone 64

Why bother with retina screens?

I for one welcome our retina-screen overlords! Seriously speaking, high-density displays are the awesome. No one likes to look at a screen and think, gee those pixels are awesome, they're so… square! Finally, computer technology has caught up with how printed books looked for hundreds of years—pristine, with no pixelation, bleeding or other defects distracting from reading text and looking at images.

If you think the iPhone was the first device with a retina screen, you'd be wrong—long before the iPhone 4, the Nokia N770 internet tablet sported a 800x480 resolution on a 4.1 inch screen (that's a density of 225 dots per inch!). That was in 2005. Since then a few things changed, however and Apple can be credited with bringing real "retina" density to a larger market.

The Nokia at that time would only last 3 or 4 hours on a charge of batteries, the user interface was clunky and compared to what's available now, it was really, really slow. Of course, most importantly, the pixel density is even higher now, and with more than 300 dots per inch a human eye can no longer discern individual pixels at a normal viewing distance.

Resolution vs. Density vs. Pixel Ratios

It's easy to confuse these, so here's the gist: the display resolution of a screen describes the total amount of pixels available in each dimension—for example, the original iPhone has a resolution of 320×480 and a 2012 27-inch iMac clocks in at 2560×1440. Some people use terms like "VGA" resolution for some of the more popular resolutions (VGA is 640×480). Don't bother trying to memorize all of them, there are too many screen sizes these days.

Pixel density, on the other hand, is how many pixels fit into a fixed amount a space, and is most often given as pixels per inch (ppi) or dots per inch (dpi). These units are the same and can be used interchangeably. For example, the original iPhone sports 163 ppi, while the 27-inch iMac has a density of 109 ppi. It follows that the iMac compared to the old iPhone has a much higher resolution but a much lower pixel density.

An other term used is "pixel ratio" especially when dealing with high-density displays in JavaScript and CSS. The pixel ratio

specifies how a physical display pixel compares to a pixel on a "normal" density screen. A "normal" display is anything between about 100 ppi (desktop, laptops) and 175 ppi (mobile), give or take. As used in this ebook, "Retina" displays are roughly 200 ppi and up.

So what is "Retina" exactly?

Glad that you ask. First off, "Retina" is a marketing term Apple uses to say something is "high-density", but for the sake of simplicity I'm liberally applying this term to anything that's not a "normal" density, or more specifically any display where, by default, one CSS pixel is larger than one physical screen pixel.

A screen that is, in Apple-speak, called a "Retina" screen has twice the pixel density of a "normal" screen with the same physical size. That's four times as many pixels. There's also some in-between densities on some Android phones, for example the Google Nexus One phone has a 1.5× higher pixel density than a "normal" phone. This is equal to slightly more than 2× as many pixels. In Android-lingo, that's called a "high DPI" phone.

Here's an overview of screen DPI variations in use:

Screen type	pixels per inch	Example devices
desktop display	100 to 110	iMac 27"

Screen type	pixels per inch	Example devices
normal laptop screen	110 to 135	MacBook Air
normal tablet	130	original iPad
normal smartphone, “mdpi” Android phone	160	HTC Wildfire
“retina” laptop	220	Retina MacBook Pro 15”
“hdpi” Android tablet	216	Nexus 7
“hdpi” Android phone	254	Nexus One
“retina” iPad	264	2012 New iPad
“xhdpi” Android phone	316	Nexus S
“retina” iPhone	326	iPhone 4S

It’s clear from this table that “retina” doesn’t just mean pixel doubling. **Given that it’s clearly not possible to support all current and future pixel densities explicitly, it’s better to try to target the highest density as a default**, and only making adjustments for lower densities when absolutely necessary.

Always target high-density displays by default, don't make it an afterthought

You could use just high-resolution images, and you can get away with that as browsers do a decent job of scaling down images to "normal" screen densities. However, **in some cases you need to do variations and serve different images** depending on the screen density—for example icons may require this to stay sharp and legible.

CSS pixels and device pixels

The main thing you need to be concerned about when designing for and supporting retina screens is that a CSS pixel no longer equals a device pixels. That is, the "px" unit in CSS now stands for "device-independent pixels", which are mapped to physical pixels on your screen, in a factor of 1:2 on Apple's retina screens (this upscaling is true for other units as well).

Be aware that there are also devices out there that have a ratio of 1:1.5—namely "Hi-DPI" android phones, and there are even some devices around that have a lower than 1:1 ratio, called "Lo-DPI".

There's an upcoming standard that adds a lot of features dealing with resolution to CSS—but it will be a while until this is available in browsers.

If you're interested in the history of support for high-density displays in web browsers, don't miss this article from 2006 on the WebKit blog.

Locking down the CSS pixel ratio

If you're designing specifically for mobile devices, you may want to prevent the user from zooming in and out of your site, while keeping the device's pixel ratio settings as-is. You can achieve that via the the "viewport" meta tag:

```
<meta name="viewport" content="initial-scale=1, maximum-
scale=1">
```

Force-mapping CSS pixels to device pixels

You can also force-map CSS pixels to be the same size as physical screen pixels. However, you'll need to specify an actual number to the viewport meta tag, and your website will break horribly when the user changes the device orientation from horizontal to vertical, as you can't specify separate values. Watch out for the dragons:

```
<!-- force 1:1 mapping on iPhone 4 (horizontal) -->
<meta name="viewport" content="width=640, user-
scalable=no">
```

This is most useful in a web view inside of a native application.

Detecting retina screens in JavaScript

To detect retina screens in JavaScript, you can query the "window.devicePixelRatio" property on most browsers, with a fallback to a "matchMedia" query on Firefox and Internet Explorer 10:

```
function isRetina(){
  return (
    ('devicePixelRatio' in window &&
      devicePixelRatio > 1) ||
    ('matchMedia' in window &&
      matchMedia("(min-resolution:144dpi)").matches)
  )
}
```

At the time of this writing some browsers have no reliable way to detect Retina screens, but there's a solution that does work for some older browsers like Internet Explorer 9, that even when running on a high-density display (which are supported just fine on Windows 7 and 8), have no support for either the "devicePixelRatio" or for the "matchMedia" API.

You can use this polyfill to add support for the "matchMedia" API in browsers that don't come with support for it (your milage may vary as this relies on media query support in CSS, but it does work for IE 9, at least):

https://github.com/paulirish/matchMedia.js

Targeting retina screens in CSS

To specify retina-screen only CSS rules, you'll want to know about CSS media queries and specifically the "device-pixel-ratio" query. As with seemingly all web development topics, it would be too simple if it was easy out of the box. In this case, Apple first introduced "-webkit-min-device-pixel-ratio" with the iPhone, other vendors based their implementation on Apple's but then something completely different was standardized.

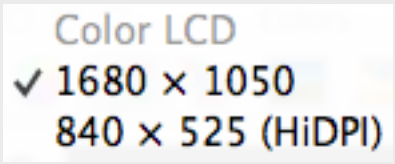
Luckily, a workaround is always around the corner, and here's how you target any and all high-density screens on any browser known to man:

```
@media (min--moz-device-pixel-ratio: 1.5),
       (-o-min-device-pixel-ratio: 3/2),
       (-webkit-min-device-pixel-ratio: 1.5),
       (min-resolution: 144dpi),
       (min-resolution: 1.5dppx) {
  /* your retina rules here */
}
```

Whoa—what's going here? This media query selects any screen that has a ratio of physical pixels to logical pixels of at least 1.5. It works on Firefox ("min--moz-device-pixel-ratio"), Opera ("-o-min-device-pixel-ratio"), WebKit-based browsers like Safari and Chrome ("-webkit-min-device-pixel-ratio"), Internet Explorer 9 and 10 ("min-resolution: 144dpi") and throws in the standardized way to do it as well ("min-resolution: 1.5dppx"), so you're good for the future (browsers don't support this directly yet).

Just specify rules for retina only within the query and you're golden.

Testing on Retina if you don't have a device

Device	Test environment
Android	Android Emulator that's included in the Android SDK. Install the Intel Atom x86 system image. You can emulate all Android screen densities.
iPhone Retina	iOS Simulator, included in Xcode.
iPad Retina	iOS Simulator, included in Xcode.
MacBook Pro Retina	To use "HiDPI" modes on a regular Mac, you'll need to enable them with a tool like Quartz Debug (this tool is included in "Graphics Tools for Xcode", in a separate download from Apple's developer site. Once enabled, HiDPI modes will show up in System Preferences and the "Displays" menu item: 

Here's a handy tip for the iOS Simulator—you can set a window scale via the "Window" → "Scale" menu item, so you can fit a simulated Retina screen on a laptop screen while working on the go. You can also use keyboard shortcuts: ⌘1 for 100% (default), ⌘2 for 75% and ⌘3 for 50%.

Note that on a retina MacBook Pro, the iOS Simulator automatically pixel-doubles "normal" density screens.

Further reading

Here's more documentation for Android devices:

Android API guide: Targeting Screens from Web Apps

Unfortunately Apple's main guide to creating content for Safari hasn't been updated for retina screens, but there's still a wealth of information to be found here:

Safari Web Content Guide

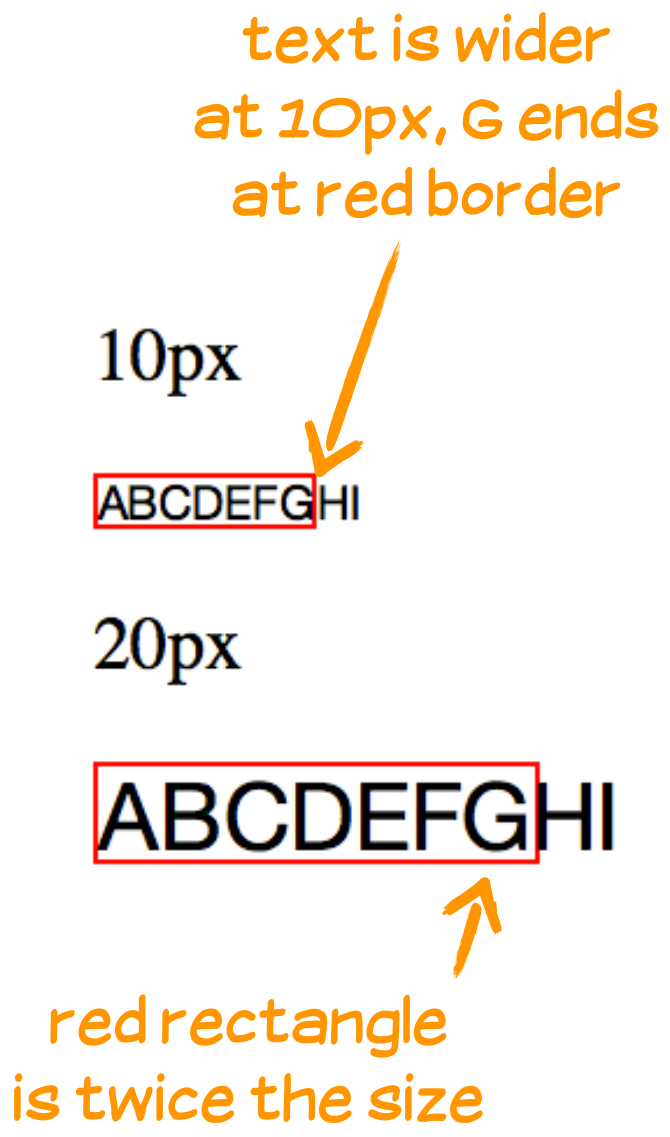
If you use retina web content inside a native iOS or OS X application, I'd encourage you to read Apple's guidelines:

High Resolution Guidelines for OS X

Support High-Resolution Screens (iOS)

Text & Fonts

Good font rendering engines, like the subpixel-engine that Apple uses, produce crisp and sharp text, even at very small sizes on non-retina displays. For CSS-based units, this becomes especially important when you use a “font-size” of “12px” or lower. OS X’s implementation of retina modes scale these linearly. This means that on a retina screen, a font that is specified as “10px” in CSS is not rendered just in size “20px” (which might yield a slightly different output), but it’s rendered like on a normal-density screen, just using four times as many pixels.



There’s a lot more info about font rendering in this excellent article about font rasterization, and Beat Stamm’s site Raster Tragedy.

Text is text, and won't flow differently on a retina screen

This means, there can be no surprises with "surprise line breaks" or other rendering differences. However, be aware that problems can be caused by different text rendering engines in browsers (for example, not even Safari and Chrome on OS X render the same text exactly the same); and between desktop and mobile operating systems. Care should be taken to have a little bit of extra room so these little differences won't cause problems.

CSS backgrounds and units

Backgrounds and high-resolution images

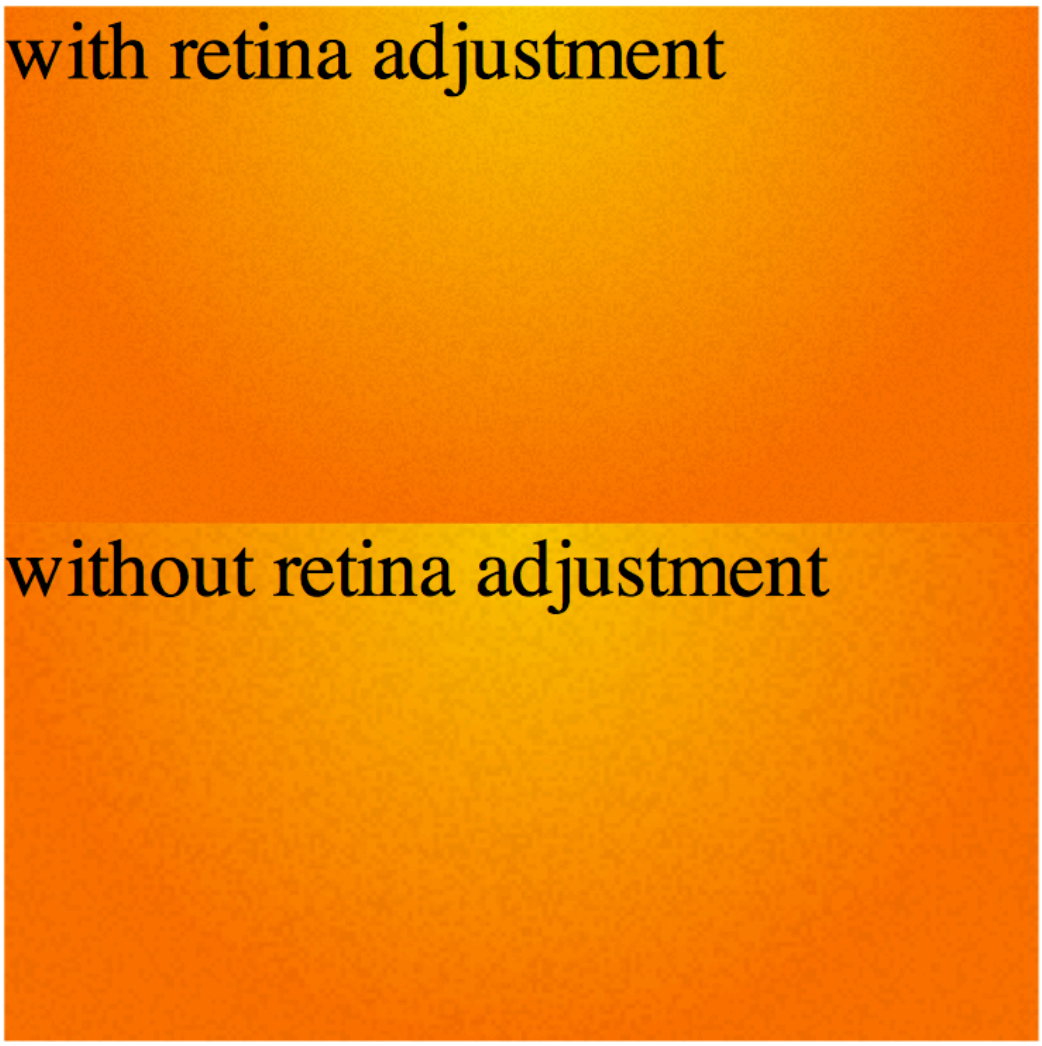
CSS3 allows for multiple background images on one and the same element, and this can be used to fine-tune the appearance of elements on retina displays. For example, here's how to get a grainy texture that is combined with a CSS3 gradient to look right on both normal and retina screens:

```
<style>
  #box {
    background: #ff6d00;
    /* sadly, cross-browser gradient definitions are painful */
    background: url(grain.png), no-repeat 0 50% -webkit-
gradient(radial, center top, 0, center top, 100, from(#fdc600),
to(#ff6d00));
    background: url(grain.png), no-repeat 0 50% -webkit-radial-
gradient(50% 0%, circle, #fdc600,#ff6d00);;
    background: url(grain.png), no-repeat 0 50% -moz-radial-
gradient(50% 0%, circle, #fdc600,#ff6d00);
    background: url(grain.png), no-repeat 0 50% -o-radial-
gradient(50% 0%, circle, #fdc600,#ff6d00);
    background: url(grain.png), no-repeat 0 50% radial-
gradient(50% 0%, circle, #fdc600,#ff6d00);
    width: 200px;
    height: 100px;
    background-size: 100px 100px, 100% 100%;
  }

  /*
    change the background size for the grain to (better)
    match display pixels, but leave the gradient to
    automatically size itself
  */
```

```
  @media (min--moz-device-pixel-ratio: 1.5),
         (-o-min-device-pixel-ratio: 3/2),
         (-webkit-min-device-pixel-ratio: 1.5),
         (min-resolution: 1.5dppx) {
    #box {
      background-size: 50px 50px, auto;
    }
  }
</style>

<div id="box">
  Woo-hoo, I look great on retina screens!
</div>
```



The trick is to adjust the “background-size” property—it supports

setting a size for each individual background image component. On a normal screen, the grain image, which itself is 100px by 100px in size is shown at exactly that size. On retina, it's shown at 50 CSS pixels by 50 CSS pixels in size, which means that each image pixel matches one physical device pixel (on the retina screen 1x1 CSS pixel equals 2×2 physical device pixels).

Internet Explorer (before version 9)

Unfortunately, Internet Explorer up to version 8 doesn't support the CSS "background-size" property. This means that if you have bitmap CSS background images (PNG, JPEG or GIF) that you need to work on older Internet Explorer versions, you will have to prepare two versions, one for retina screens, and one for normal screens.

The good news is that if you follow Google's lead on which browsers to support you don't have to bother with this—they support "the current version and version before", which as of this writing means Internet Explorer 10 and 9, which both do the "background-size" property just fine.

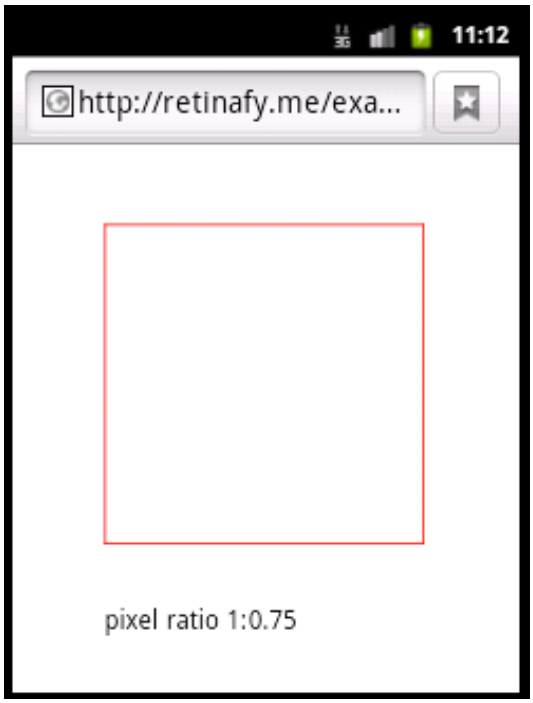
If you have trouble getting it to work, be sure that your page has a proper "<!DOCTYPE html>" declaration and doesn't try to set one of IE's compatibility modes.

Non-integer pixel ratios and CSS units

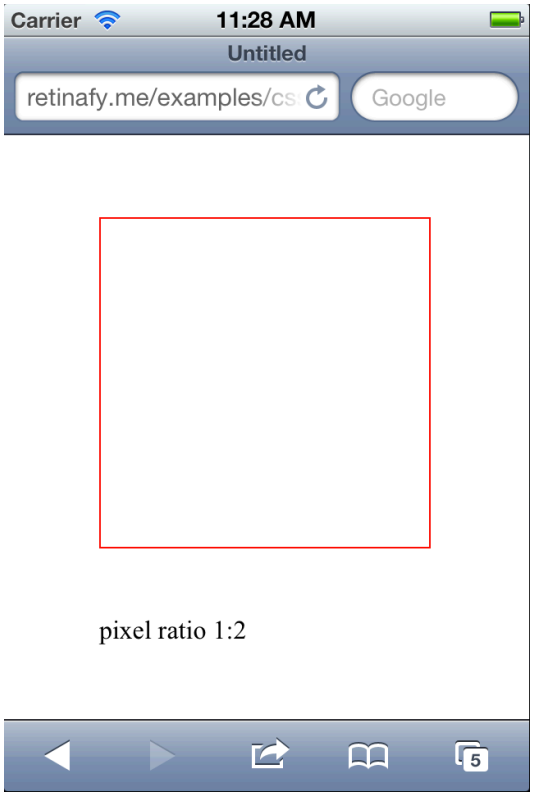
As discussed earlier, some devices have CSS pixel density ratios other than 1:1 or 1:2.

This leads to some peculiarities in CSS, because the number given to the "px" unit must be a full integer. For example on a "real" Retina screen, that is for a pixel ratio of 1:2, borders of elements can (by default) only have a width specified in multiples of 2 physical pixels ("1px" = 2 device pixels, "2px" = 4 device pixels, and so on).

It even means that on some displays, CSS pixels may be sometimes doubled omitted altogether; or are blended. You can see that a "1px" border in the example on the right (a "Lo-DPI" Android device) is slightly wider on the top and left of the square.



On the other hand, on a real high-density display (pixel ratio 1:2, the pixels are doubled completely) as seen on this screenshot from a Retina iPhone. Note that you can just use different units when specifying a CSS border width—border widths, like any other CSS width must always be multiples of CSS pixels, which, like it or not, is a peculiarity of CSS. If you use other units, those will be rounded to CSS pixels. (Note that this has nothing to do with the scale/zoom level of your website.)



Bitmapped images

PNG

The mainstay of your website's user interface images are very likely in the PNG format. There are four distinct ways to make these retina-ready:

1. **Convert to CSS**—if you use PNGs for background gradients or to make buttons look better, you'll probably be better off by making use of CSS3 features to style your elements. This is better in many ways, not only will you get a "retina version" for free, but your site will also load faster and will be easier to maintain.

2. **Convert to SVG**—for user interface elements like icons, arrows, pointers and so on, converting your PNG to vectors and using SVG is a great option.

3. **Use a larger version and scale down on normal screens**—this works great for most images, and you don't need to deal with the overhead of multiple versions; and you likely won't need extra markup or styles.

4. **Use two versions**—in some cases it's necessary to keep tight control and have separate versions for normal screens and high-density screens, especially when the version for

normal screens is very small (width or height below about 20-30 pixels).

Convert to CSS

If your image is any kind of **gradient** (even for complex, multiple-layer gradients) it's very likely a good idea to **convert to CSS gradients**. These have two big advantages, first they are rendered on the fly by the browser in the exact dimensions required, depending on scale level and pixel density—so you always get the best looking gradient possible. Plus, it's likely that you save a an extra HTTP request and the need to load a big external PNG file (non-trivial gradient images can become quite large). It takes a bit getting used to it (the syntax is quite verbose, especially with all the cross-browser variants required, but it's worth learning it!).

Two tools that can help with converting your existing gradients to CSS are the "Ultimate CSS Gradient Editor" and CSS Hat.

Convert to SVG

For more complex UI elements like icons, SVG vector graphics are a good choice. These are supported by all modern browsers with the major exception of older IE versions and Android 2 (which unfortunately most Android phones still run). If you don't have vector originals (e.g. Adobe Illustrator files) of your icons or UI elements you may have to draw them as vector graphics first—but any designer worth his or her money will draw stuff like icons

with a vector graphics illustration tool anyway. It's easy to export SVG graphics from Adobe Illustrator and you can stick those right into your HTML. See the section on SVG graphics for details and on how to fall back to PNG when required.

Use a larger version (scale down when required)

It turns out that for many types of graphics you can **just provide a single image**—the "retina" version, and **rely on the browser to scale down** a larger image.

This is both fast and yields great quality. As shown in the next section, this will likely not increase file sizes a lot either; however there are reasons why you may want to use two versions as described below.

The case studies section lists some examples of when to use downscaled PNG images and when to go the extra mile and have separate versions.

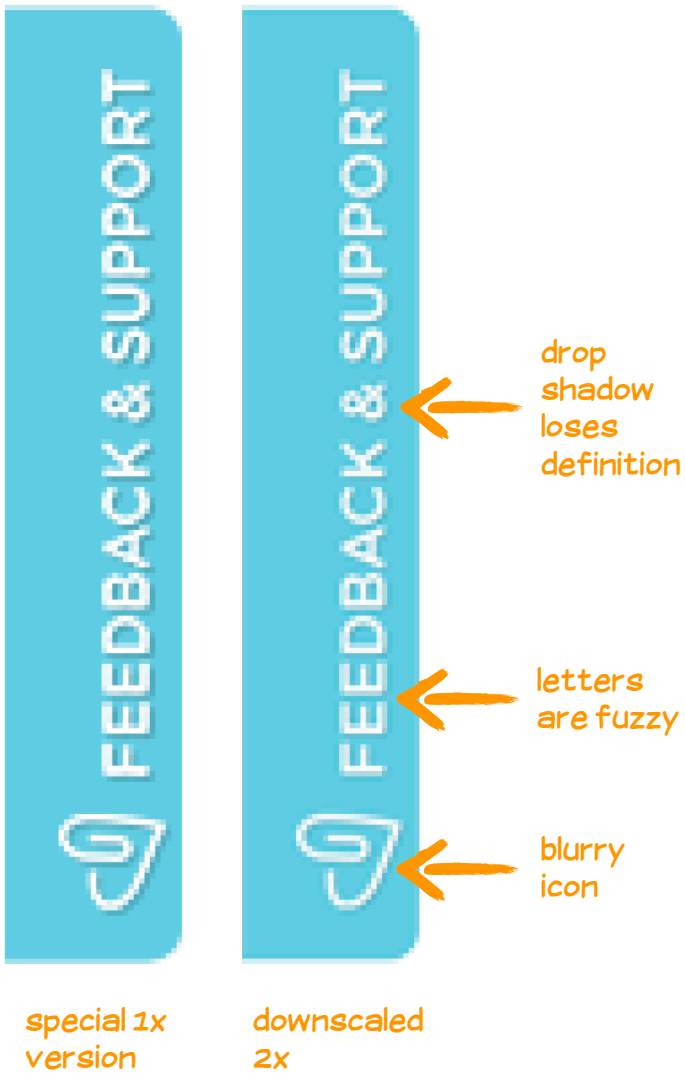
Using two versions—how? And why?

Some images might require special casing, for example an icon or small text rendered on a user interface element might get blurry at smaller sizes. Here's an example where I used two different image files, the feedback tab that you can see on http://retinafy.me, which is part of our Charm Customer Support SaaS app, and loaded in via a "third-party" JavaScript file:

The file size of the retina version, while sporting four times as many pixels, is only about 1.4 times as large as the normal version. The main reason why there's a normal version at all is the legibility of the small text—just scaling it down would yield a very blurry result.

Here's a comparison of the specially crafted 1x version and the downscaled 2x image. For this particular image it's quite clear why downscaling is not a good option—the text and icon get quite fuzzy and are harder to read:

It's quite easy to use two variants like this—don't use <img> tags directly, but instead just use elements with a background image and "background-size" property of "100%", combined with the aforementioned media query. If you use separate versions in more than a few places, it's probably a good idea to have a special section in your CSS file with all the "retina" exceptions in one place—this will save you a lot of typing or copy and pasting as the media query rules are quite a mouthful:




```
/* basic two-image variation CSS snippet */

#CHARM_TAB {
  background: url(feedback.png);
  background-size: 100%;
  width: 40px;
  height: 100px;
}
```

```
@media (min--moz-device-pixel-ratio: 1.5),
       (-o-min-device-pixel-ratio: 3/2),
       (-webkit-min-device-pixel-ratio: 1.5),
       (min-device-pixel-ratio: 1.5),
       (min-resolution: 1.5dppx) {

  /* on retina, use image that's scaled by 2 */
  #CHARM_TAB {
    background:url(feedback2x.png);
  }

}
```

A major issue to keep in mind is that the "background-size" property is that it is not supported on Internet Explorer 8 and lower.

If you still target older IE versions, you'll need to either provide separate images for backgrounds, or, where possible switch to IMG tags or just plain CSS (with degraded appearance).

"background-size" is not supported on older IE versions, so plan accordingly

JPEG

If you're using JPEG images for anything that's not content like photos, there's a high likelihood that you're doing it wrong—use PNGs instead and see the section on PNGs. For user interface images some exceptions do exist, like large backgrounds; but these can often be better implemented with CSS.

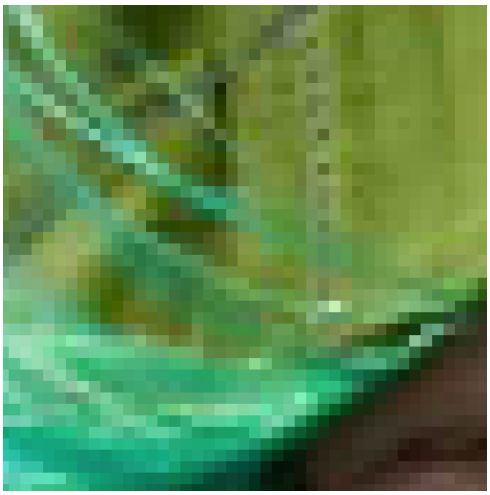
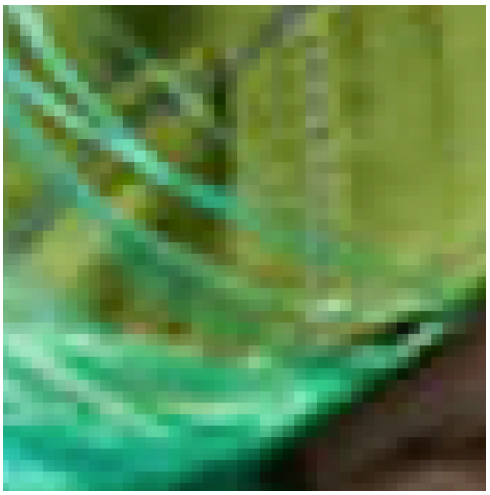
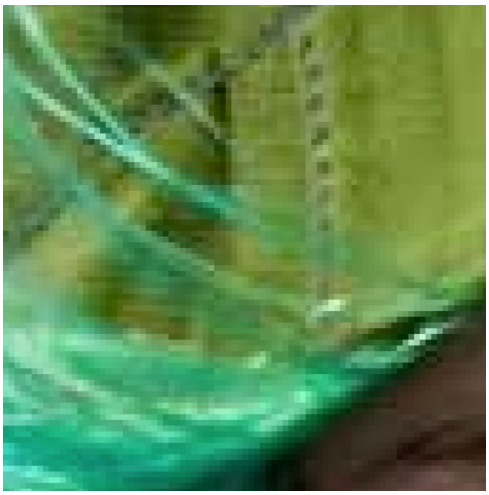
For JPEGs that are content, just use images with twice the resolution (or as high as available) and lower the JPEG quality.

For content images, double the resolution but lower the quality. Don't bother with two versions.

You can go really, really low with JPEG quality on retina screens because the pixels are so small. The typical JPEG problems like blocky looking artifacts and color issues are not as discernible, and as an added bonus downscaling the larger JPEG images on normal screens works really well.

left: JPEG, 500x332, quality=60, size=45386 right: JPEG, 1000x664, quality=25, size=54358 (+20%)

In this example, the JPEG quality is set to 25, which, when scaled down on a normal screen is largely indiscernible to a quality 60 image at 1/4 the amount of pixels. (See http://retinafy.me/examples for the test files used.)

	quality 60 normal JPEG	quality 25 hi-res JPEG
Normal screen		
Retina screen		

For the normal screen, using the downscaled hi-res image with the lower quality yields excellent results. Doing the reverse, using a high-quality lo-res image on the retina screen creates an abysmal washed-out image. The hi-res image on the retina screen speaks for itself with the added sharpness and detail.

This is an other example, using a quality setting of just 15, which actually results in the retina version of the image having a smaller file size than the original normal version:

left: JPEG, 500x508, quality=60, size=87802 right: JPEG, 1000x515, quality=15, size=83150 (-5%)

Again, the image quality on a normal screen is practically the same.

Because modern browsers like Safari allow free zooming with touch gestures (both on desktop and mobile, with mobile being the more widely used), it's a big advantage to use high-resolution images even on normal screens—zooming in will reveal more details.

There are a few things you should be aware about when using large JPEGs:

1. **iOS 4 and 5 has a two megapixel limit on JPEG size.** Anything larger will be shown down-sampled to 2 MP. (This has been substantially increased in iOS 6).

2. **Downscaling on normal screens can create aliasing artifacts on IE 9** (IE 7/8 are not affected). You should test your images on IE 9 to see if that is a problem (most of the time, it's barely visible).

3. **For some images, low quality settings might now be acceptable**—always be sure to test settings thoroughly and optimize as much as possible, but not more!

GIF

If you're using non-animated GIFs, convert them over to PNGs first. Once you've done that, see the section on PNGs.

Animated GIFs

For animated GIFs, you can't just use PNGs instead as they lack any animation capabilities. Besides being used for funny short videos, the main use of animated GIF files is small user interface graphics, such as icons (for example, a rain cloud on a weather map might be an animated icon). There's two ways to deal with these:

1. **Use a larger version and scale down on normal screens**—this works great for most images, and you don't need to deal with the overhead of multiple versions; and you likely won't need extra markup or styles.

2. **Use two versions**—in some cases it's necessary to keep tight control and have separate versions for normal screens

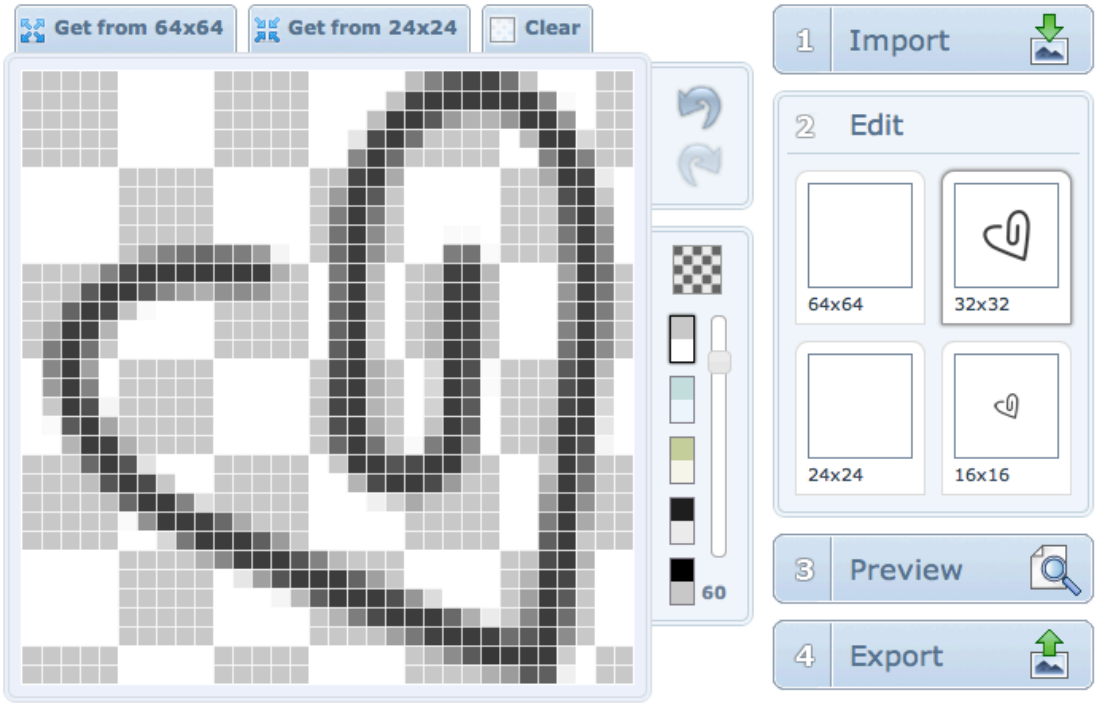
and high-resolution screens, especially when the version for normal screens is very small (width or height below about 20-30 pixels).

Favicons

Next to the URL/search field, the favicon is used in various other places, like tabs and bookmarks.

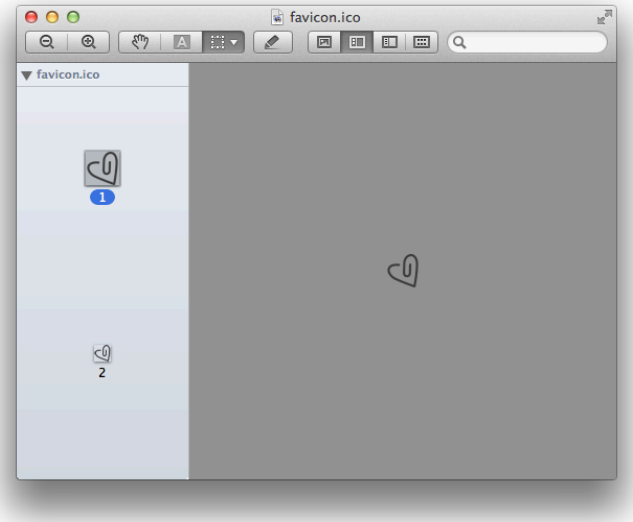
To make proper favicons that work across all browsers, notably older versions of IE, you'll need to create an .ICO file with two size variants: 16px by 16px for standard screens, and 32px by 32px for retina screens.

First, create the two icon variants in any image editor (I use Photoshop), and save them as 32-bit PNGs with an alpha channel. Next you need to use a specialized icon editor application to create the actual .ICO file. http://xiconeditor.com/ is a free web-based app that works well for this:



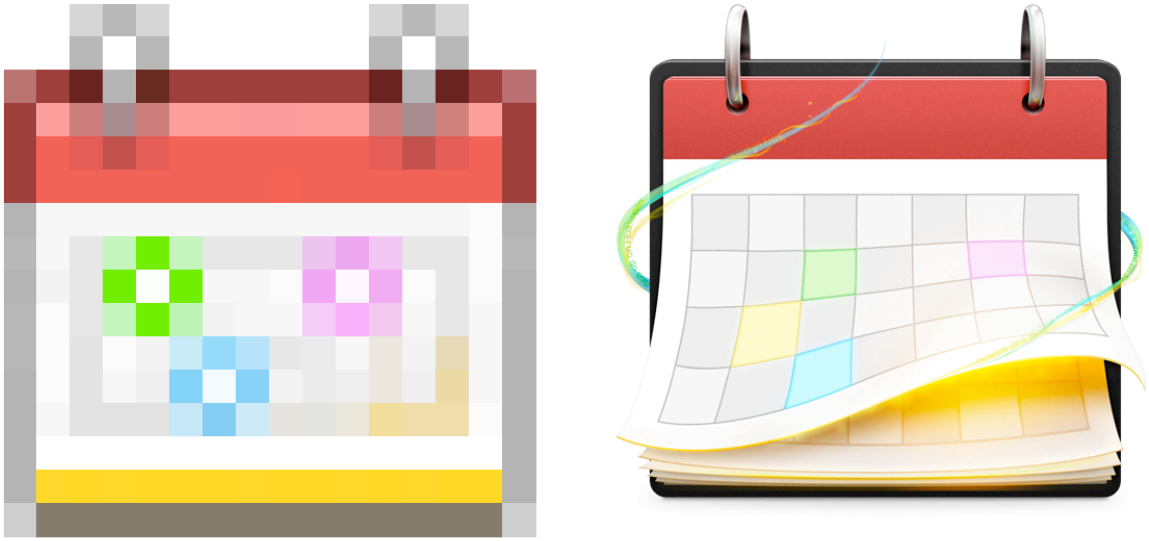
Import both size variants, and save your .ICO file as “favicon.ico” in your site's root folder.

You can easily check if you favicon is retina ready, just use Preview.app to take a peek. All size variants will show up.



When it gets to favicons, the devil is in the details—for example I had to manually tweak the 16×16 Charm heart pixel-by-pixel to make sure it stayed sharp and crisp.

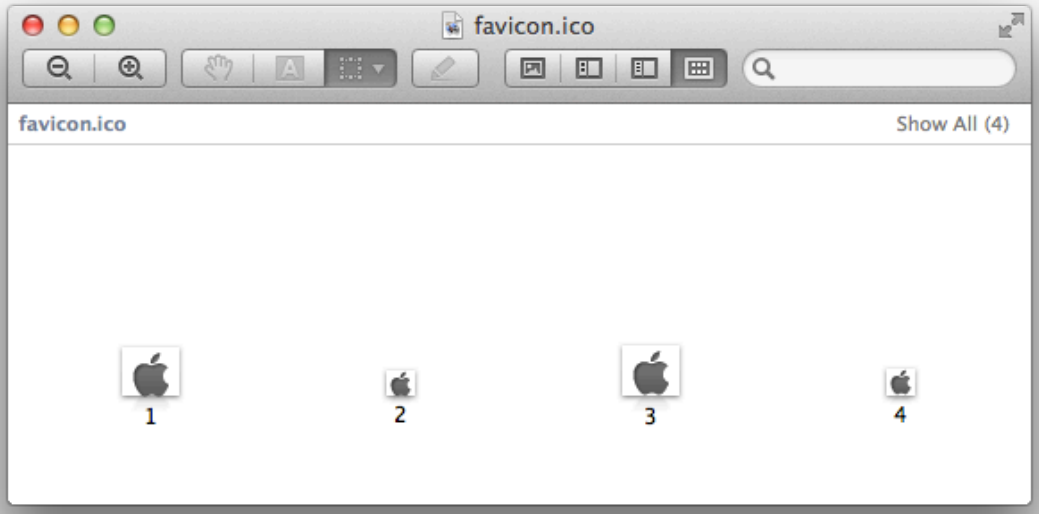
You can learn a lot about good icon design for various sizes from desktop apps, where icons have traditionally been available in varying sizes. Here's an icon from Fantastical, a OS X calendar app I use (and love!), showing both the smallest (16×16) and the largest (512×512) variation:



16X16 512X512

You can clearly see that the smaller version is not just an automatically scaled-down version of the big icon—that would result in a complete blurry mess. Instead, the small icon is hand-drawn, with unnecessary details removed and greatly simplified.

Note that you can have the same icon size in your favicon file more than once, with different alpha channels. For example, Apple.com's favicon.ico file has four different variants, with 1-bit and 8-bit transparency for both icon sizes:



In practice, you can get away with just two icons, 16px by 16px for standard screens, and 32px by 32px for retina screens, both using 8-bit transparency—only really old browsers won't be able to use this format.

Touch icons

Additionally, you will want to create a series of resolution variants for mobile home screen icons. This is separate from favicons and is configured via a meta tag.

For the widest support, you'll need to create four icons, in 57×57, 72×72, 114×114 and 144×144. You probably want to use a good photoshop template for this.

Once you have those saved as PNGs, use the following HTML snippet. This snippet is from Mathias Bynens' excellent post about touch icons, which has a lot more information about this topic.

```
<!-- from http://mathiasbynens.be/notes/touch-icons -->

<!-- For third-generation iPad with high-resolution Retina
display: -->
<link rel="apple-touch-icon-precomposed" sizes="144x144"
href="apple-touch-icon-144x144-precomposed.png">

<!-- For iPhone 4/5 with Retina display: -->
<link rel="apple-touch-icon-precomposed" sizes="114x114"
href="apple-touch-icon-114x114-precomposed.png">

<!-- older full-size iPads and iPad mini: -->
<link rel="apple-touch-icon-precomposed" sizes="72x72"
href="apple-touch-icon-72x72-precomposed.png">

<!-- For non-Retina iPhone, iPod Touch, and Android 2.1+
devices: -->
<link rel="apple-touch-icon-precomposed" href="apple-touch-
icon-precomposed.png">
```

Note that you need to specify the various sizes in this exact order for maximum compatibility. The icons will look gorgeous on any iOS or Android device when the user adds your URL as a home screen link.

Touch startup images

If your web app has special support for running in "home-screen" mode on a mobile device, or even is a HTML5 offline-capable web app, you'll want to specify startup images, to prevent an ugly white

flash when the user launches your HTML5 app. These startup images can be tricky to debug when they're not showing up.

Here's time-tested HTML code that works:

```
<link rel="apple-touch-startup-image" href="startup-ipad-
landscape-retina.png" media="screen and (min-device-width:
481px) and (max-device-width: 1024px) and
(orientation:landscape) and (min-device-pixel-ratio: 2)">
<link rel="apple-touch-startup-image" href="startup-ipad-
portrait-retina.png" media="screen and (min-device-width:
481px) and (max-device-width: 1024px) and
(orientation:portrait) and (min-device-pixel-ratio: 2)">
<link rel="apple-touch-startup-image" href="startup-ipad-
landscape.png" media="screen and (min-device-width: 481px)
and (max-device-width: 1024px) and
(orientation:landscape)">
<link rel="apple-touch-startup-image" href="startup-ipad-
portrait.png" media="screen and (min-device-width: 481px)
and (max-device-width: 1024px) and (orientation:portrait)">
<link rel="apple-touch-startup-image" href="startup-
retina.png" media="screen and (max-device-width: 480px) and
(min-device-pixel-ratio: 2)">
<link rel="apple-touch-startup-image" href="startup.png"
media="screen and (max-device-width: 320px)">

<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style""
content="black"">
<meta name="format-detection" content="telephone=no">
```

It's possible that there will be even more variations in the future—keep an eye out for different aspect ratios than the familiar 4:3 on iPads and iPhones.

The images themselves must use these exact resolutions:

Image	Resolution
startup-ipad-landscape-retina.png	1496×2048
startup-ipad-portrait-retina.png	1536×2008
startup-ipad-landscape.png	748×1024
startup-ipad-portrait.png	768×1004
startup-retina.png	640×920
startup.png	320×460

Procedural and vector graphics

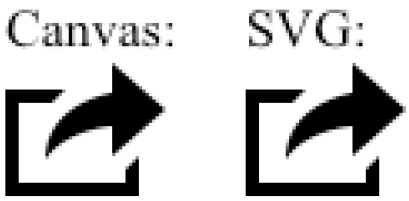
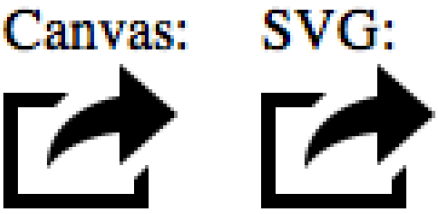
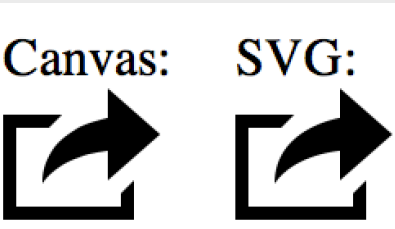
Canvas

If you use the canvas element to render bitmapped graphics and animations, be prepared to dig a little bit deeper than with most other assets. The Canvas element is designed to be a pixel-by-pixel affair and you might not want the browser to automatically scale the Canvas element up. Instead you can fine-tune how it should behave.

For Safari, note that there's a difference between mobile and desktop retina screens—on mobile retina screens Canvas elements are by default just pixel-doubled, so they look just like images that are not at retina resolution, or in other words, awful.

The trick is to set the resolution of the Canvas element independently from its size in CSS pixels. The Canvas' element "width" and "height" attributes controls the internal Canvas resolution, while the "width" and "height" CSS properties control the element size.

You can clearly see that Retina screens on iOS form a major exception here—because you'll need to do extra work to make Canvas elements work in high-resolution.

Device	Screenshot
iPhone (normal)	
iPhone (Retina)	
MacBook Pro (normal)	
MacBook Pro (Retina)	

To get your Canvas element to use all the available pixels on iOS, you'll need to set the resolution explicitly:

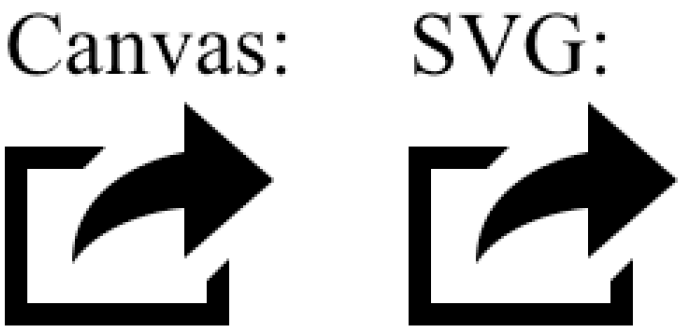
```
<canvas width="48" height="40" id="canvas" style="width:
48px;height:40px"></canvas>

<script>
var canvas = document.getElementById('canvas'),
    ctx = canvas.getContext('2d')

/* set up scaling for iOS */
if('devicePixelRatio' in window &&
  ctx.webkitBackingStorePixelRatio < 2) {
  canvas.setAttribute('width', 48 * devicePixelRatio)
  canvas.setAttribute('height', 40 * devicePixelRatio)
  ctx.transform(devicePixelRatio,0,0,devicePixelRatio,0,0)
}

/* ... */
</script>
```

Using this bit of trickery will ensure that your canvas elements look great everywhere (screenshot on iOS Retina):



Note that you may have to adapt some other code, like when you

draw bitmapped images in the canvas. If you are paying attention, you might be curious about this line:

```
   ctx.webkitBackingStorePixelRatio < 2) {
```

This tests for a new extension to Canvas contexts that was introduced with the Retina screen MacBook Pros. In essence, on OS X, Safari auto-doubles the pixels in a Canvas for you—so you don't have to set with internal width & height of the "backing store" explicitly as we do in the example. The downside is that if you do, on a MacBook Pro Retina, you'd quadruple the backing store (or 16 times as many pixels!), possibly leading to severe performance issues.

This test will make sure that you don't mess with the auto-doubling and that performance will stay snappy.

To learn more about these extra Canvas features, be sure to watch Apple's WWDC 2012 video "Delivering Web Content on High Resolution Displays". It's available free of charge if you register as a Safari Developer.

Vector graphics (SVG)

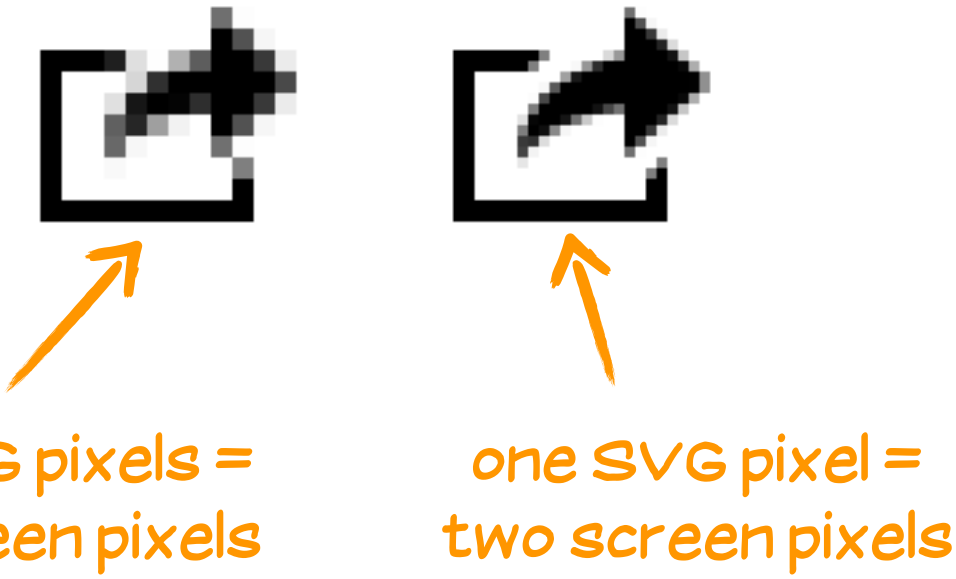
Scalable Vector Graphics, or SVG for short, is the red-headed stepchild of web design. Incredibly useful and space-saving, SVG offloads rasterizing of vector graphics shapes to the browser. Because everything is based on vectors, scaling things up is (usually) not an issue.

There are some issues with SVG, notably that very complex SVG graphics can come at a larger file size than the same as a bitmapped image, and that rendering can be a performance problem. Scaling down to very small sizes can cause blurriness, for example when rendering an icon at 16px by 16px. Additionally, SVG graphics are not supported on older versions of IE (before 9), and on Android 2.x, so you will have to have a fallback mechanism (most likely a bitmapped image) if you want to support these platforms. The specific "flavor" of SVG you'll likely want to work with is the one that is included in HTML5 and allows direct embedding of SVG inside of HTML, without any need to specify weird namespace attributes or resorting to "object" tags.

Here's an example of how a widely-compatible inline SVG snippet looks like:

```
<svg width="12px" height="10px" viewBox="0 0 12 10"
version="1.1" xmlns="http://www.w3.org/2000/svg">
<g><path
d="M12,3.5L8,0v2.25c0,0-5,0.75-5,5C4.5,5,8,4.75,8,4.75V7L12
,3.5z"/></g>
<g><polygon points="10,7 9,8 9,9 1,9 1,3 3.5,3 4.5,2 0,2
0,10 10,10"/></g>
</svg>
```

The SVG shapes and paths don't use units—the "width", "height" and "viewBox" attributes make sure that a mapping of one "SVG length unit" to one CSS pixel takes place. This means that on a regular screen, a SVG number of "1" means one physical pixel, and on a retina screen it's the same as 2 physical screen pixels.



HTML5 SVG inlining is widely supported across popular browsers, with the major exception of Android 2.x, which still (as of July 2012) is used on about 90% of all Android phones.

Browser	inline SVG
Internet Explorer	9+
Google Chrome	7+
Safari	5.1+
Firefox	4+
iOS Safari	5+
Android Browser	3+
Chrome for Android	18+

So what to do for older browsers that are still widely used, especially Internet Explorer 8 and earlier and some Android versions? You can use the fallback technique outlined here. This does two things—it adds a "no-svg" CSS class to the <body> element so you easily special case background images in your CSS, plus it replaces SVG images in "src" attributes of <img> tags:

```
<style>
  div.rss {
    background: url(rss.svg);
    width: 32px;
    height: 32px;
  }

  body.no-svg div.rss {
    background: url(rss.png);
  }
</style>

SVG with PNG fallback:<br>
<img data-svg="rss">

Background SVG with PNG fallback:
<div class="rss">

<script>
  // You'll likely need to adapt this to your needs
  // If you create new IMG tags with data-svg on the page
  // you'll need to call the window.updateSVGIMG method

  (function(global){
    var svg = !!('createElementNS' in document &&
        document.createElementNS('http://www.w3.org/2000/
svg','svg').createSVGRect)

    if (!svg) document.body.className += ' no-svg'
```

```
      ;(global.updateSVGIMG = function(){
        var i, src, extension = svg ? '.svg' : '.png',
          elements = document.getElementsByTagName('img')
        for (i=0;i<elements.length;i++)
          if (src = elements[i].getAttribute('data-svg')) {
            elements[i].src = src + extension
            elements[i].removeAttribute('data-svg')
          }
      })()
    })(this)
  </script>
```

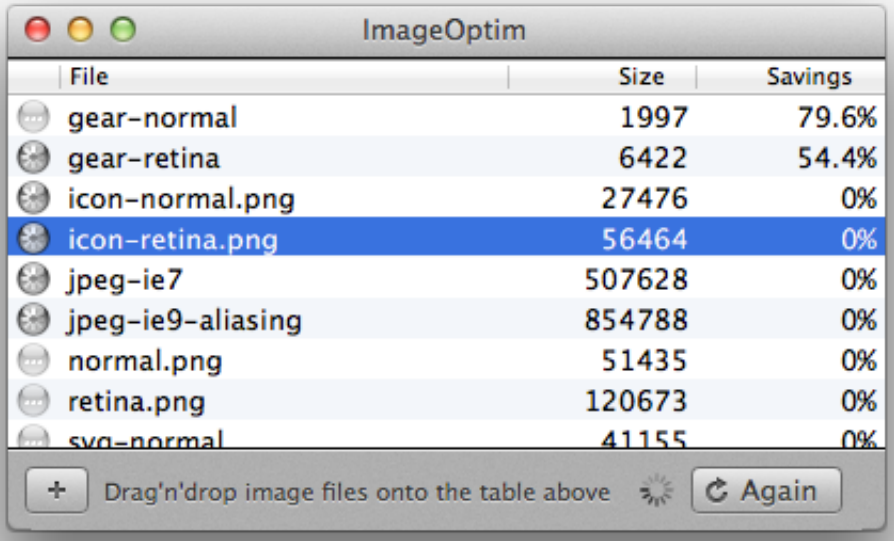
Instead of specifying "src" directly in your image tags, this script assumes you're using the "data-svg" attribute instead, and you provide a both a SVG and a PNG version. On browsers that support SVG, which is tested in the first couple of lines of code in the script tag, the SVG version is loaded, otherwise the PNG version is used.

This little snippet works out of the box without a JavaScript library —it should be a starting point only and I encourage you to adapt it to your specific needs. For example, if you're using Zepto, jQuery or other DOM helper libraries you can probably save a few bytes and implement this in a slightly simpler fashion. Or, you might want to force bitmapped images on mobile devices (it's possible that those render a tad faster!) and only do SVG on desktop.

Optimizing image file sizes

One of the most important steps of retinafying is to optimize image file sizes. While Photoshop (or whatever tool you use to create images) has a "Save for Web" feature, there are dedicated tools to get the file size down further by stripping any meta information as well as trying various compression options and color palette variations. For example, the PNG format allows for 8-bit colors (with a palette) in combination with an 8-bit alpha channel, something browsers show correctly but not supported by Photoshop. Given that your site likely uses a lot of images of all kinds, properly optimizing these can drastically improve loading times—and can often offset the file size increase required by the higher-resolution "retina" images.

A tool I can't recommend enough is ImageOptim, which is freely available at http://imageoptim.com.



It works by seamlessly integrating various command-line optimization tools that you probably never heard of into a dead-simple graphical UI. There are no options—just drop your project folder on it and for each image, it will throw a whole arsenal of optimization tools at it. Note that for larger images this can take a while, but you'll get very impressive improvements across the board, especially if you only use "Save for Web" in Photoshop.

My recommendation is to just drop your project folder onto ImageOptim once a week—just create a reminder for it.

What to do when things go wrong

Bad scaling quality

Unfortunately just using a bigger image and scaling it down on normal displays is not a silver bullet—while it works great in most cases you will eventually stumble upon limitations in browsers. When you experience bad scaling quality, you'll have to resort to provide pre-scaled images at the right size for each screen density. This is harder than it sounds because of zooming on mobile devices, so your mileage may vary.

IMG tag:

DIV with background image:

Normal-resolution CANVAS:

2x CANVAS:

4x CANVAS:

Here's a screenshot of a downscaled image on a iPhone 4 retina screen. Note that you have several options on how to display an image, the most obvious one being an image tag, but you can also use a CSS background-image, as well as a <canvas> tag; and with the Canvas tag there's a CSS resolution and an internal resolution. In this

example, none of these approaches really yield a satisfactory result, so it may be necessary to provide pre-scaled bitmap images.

```
<style>
img, div, canvas {
  display: block;
  margin-left: 20px;
  margin-bottom: 30px;
}

#avatar_background {
  width: 100px;
  height: 100px;
  background: url(thomas.png);
  background-size: 100%;
}
</style>

<img id="avatar_img" src="thomas.png" width="100">
<div id="avatar_background"></div>
<canvas id="avatar_canvas" width="100" height="100"
style="width:100px;height:100px;"></canvas>
<canvas id="avatar_canvas_2" width="200" height="200"
style="width:100px;height:100px;"></canvas>
<canvas id="avatar_canvas_4" width="400" height="400"
style="width:100px;height:100px;"></canvas>

<script>
  var img = new Image()
  img.onload = function(){
    var ctx =
document.getElementById('avatar_canvas').getContext('2d')
    ctx.drawImage(img, 0, 0, 100, 100)

    ctx =
document.getElementById('avatar_canvas_2').getContext('2d')
    ctx.drawImage(img, 0, 0, 200, 200)
```

```
    ctx =
document.getElementById('avatar_canvas_4').getContext('2d')
    ctx.drawImage(img, 0, 0, 400, 400)
  }
  img.src = 'thomas.png'
</script>
```

Jagged edges

A common problem when downscaling images as well as when using CSS transforms can be unwanted jagged edges. This problem haunts WebKit-based browsers like Safari and Chrome. The good thing is that on a retina screen they won't matter as much, but you'll still want to get rid of them.

without opacity fix

with opacity:0.999

If this problem shows up for you when rotating items, you can try

set the opacity of your element to 0.999. This forces the browser to composite the image in a better way.

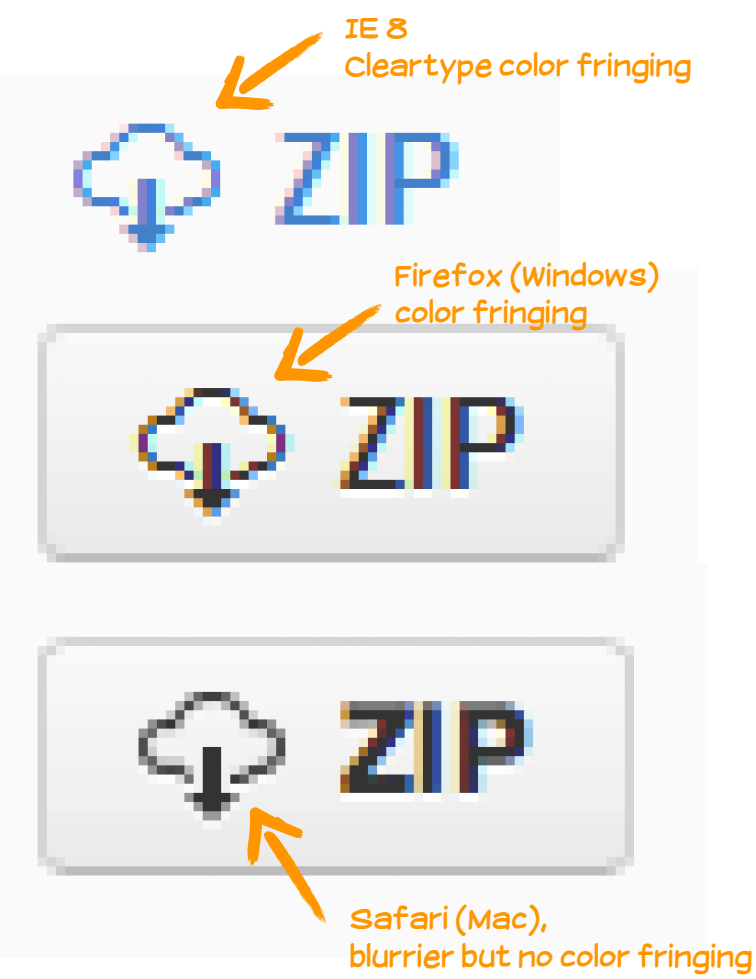
```
<style>
img {
  display: block;
  margin-left: 20px;
  margin-bottom: 30px;
  -webkit-transform: rotate(-5deg);
}

#fix {
  opacity: 0.999;
}
</style>

<img src="thomas.png" width="200">
<img id="fix" src="thomas.png" width="200">
```

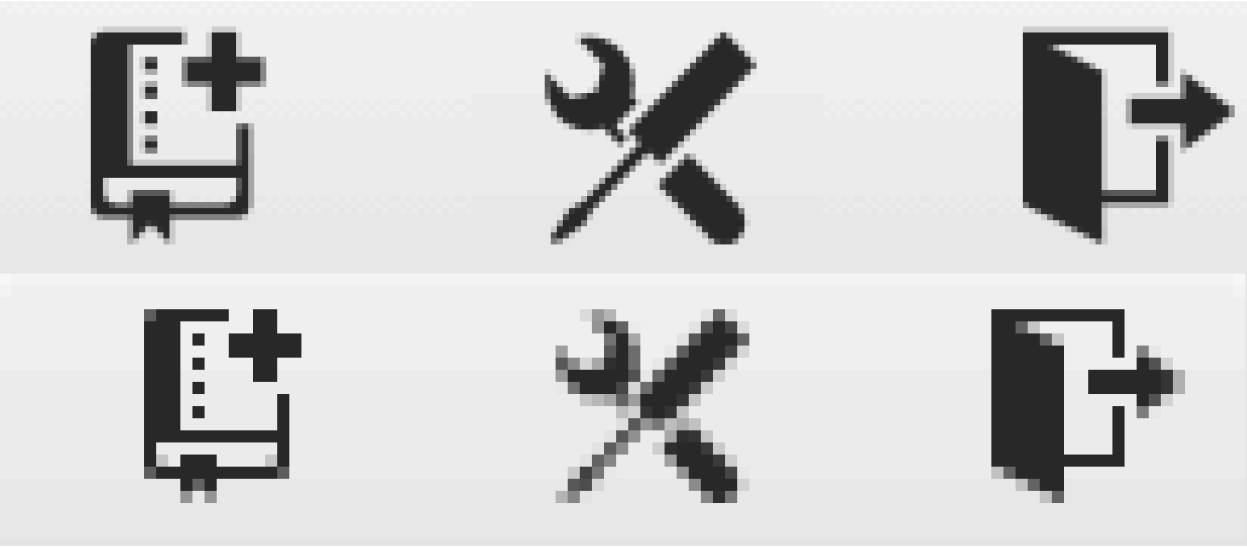
Icon fonts

While icon fonts are a tempting solution to scale icons on retina screens, I'd generally advise to use SVG or bitmapped images simply because these tend to render predictably across browsers and operating systems—font rendering engines are optimized for actual text and not for the display of images.



Next to more obvious restrictions like not being able to support more than one color for each glyph, rendering differences can be a major issue and are practically impossible to optimize. The above example shows these issues with the "ZIP download" button from Github. These issues get more pronounced with additional detail in your icons. With SVG and bitmaps it's easy to deal with this, as you can per-density special casing, and option you don't really have with icon fonts.

Here's a look at some icons on the GitHub site comparing rendering on Safari on a normal screen and a Retina MacBook Pro:



You can see that the "add bookmark" icon is rendered in crisp detail on the normal screen, but is very blurry on the retina screen; whereas the "settings" icon is blurry on the normal screen and crisp on the high-density screen. The last icon looks great on both (tho it is slightly blurry on the retina display.

Given that it's pretty hard to fix things in icon fonts (need specialized software, need to export in various formats), if you want tight control over the appearance of icons and a sane workflow, I'd recommend to completely avoid icon fonts and use SVG or PNG images instead. On the other hand, ready-made icon fonts are quite easy to work with and can be a good choice when prototyping or when you don't care about small misalignments. They'll still look better on retina screens than a scaled up, low-resolution bitmap image.

JavaScript "retina" helper libraries

There's a bunch of JavaScript helper libraries around to make your site "ready for retina", and it's tempting to think that there's an easy solution that only loads in retina images when required. However, as I've laid out it's better to think "retina-first", and assume you're running on a retina screen and make exceptions or adaptions for "normal" screens, as high-density displays will soon be the norm rather than a special thing (and they are the norm on mobile devices already!).

tl;dr—don't use JavaScript "retina" helper libraries

Combined with the extra overhead of loading in a "retina" JavaScript library, parsing and replacing items in the DOM (which causes expensive layout reflows and re-rendering) and of course downloading extra versions of page assets (depending on the library up to 3 total HTTP requests have to be made for each image!) it quickly becomes clear that there's no quick and easy "silver bullet" JavaScript solution.

Just design "retina-first", use SVG and modern CSS whenever possible and for bitmapped images, go high-resolution and let the browser scale down. Make exceptions when you have to.

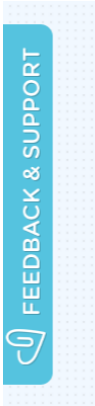
Quick Retina reference

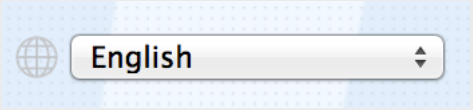
Asset type	What to do
Text	You're good, move along.
PNG	Create 2x version; only use this version except if downscaled is blurry
JPEG	Create 2x version with lower quality; only use this version
Animated GIF	Create 2x version; only use this version except if downscaled is blurry
Background patterns/ texture images	You may need to tweak the "background-size" CSS property
Canvas	Add hi-res support code; adapt use of bitmapped images if necessary
SVG	You're good, move along. (Tho you may need a fallback to PNG for Android 2.x and older IE.)
Favicon	Create a 2x version and a favicon.ico file with both variants.

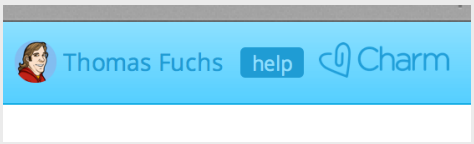
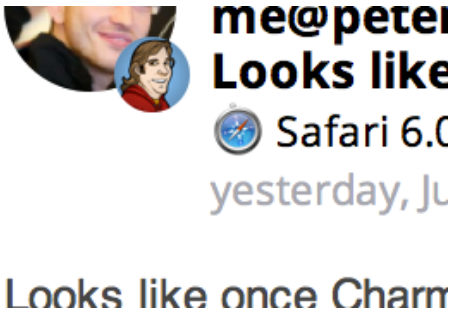
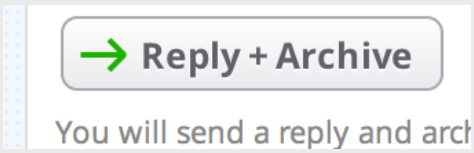
Case Studies

Charm Customer Support

Here's what I did when I upgraded our Charm Customer Support web app to work with Retina screens. Note that the web app is optimized for desktop browsers only, and doesn't support IE.

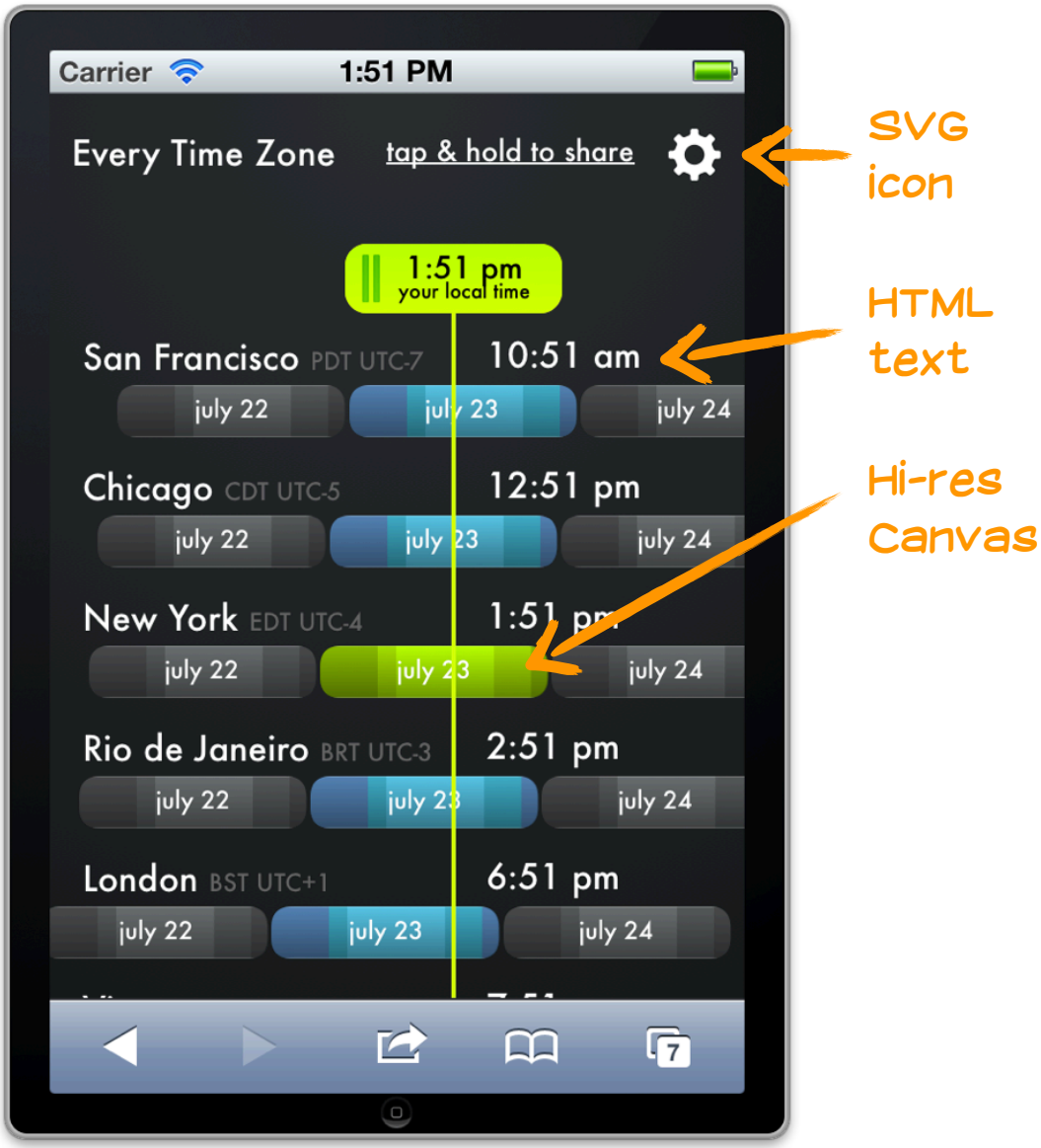
Page element	Notes
	Feedback tab uses a CSS background image (PNG) with two image variants, so the text could be hand-optimized in the smaller version to stay legible at small sizes.
	Logo on login page. Hi-res PNG that is downscaled with width/height CSS attributes.

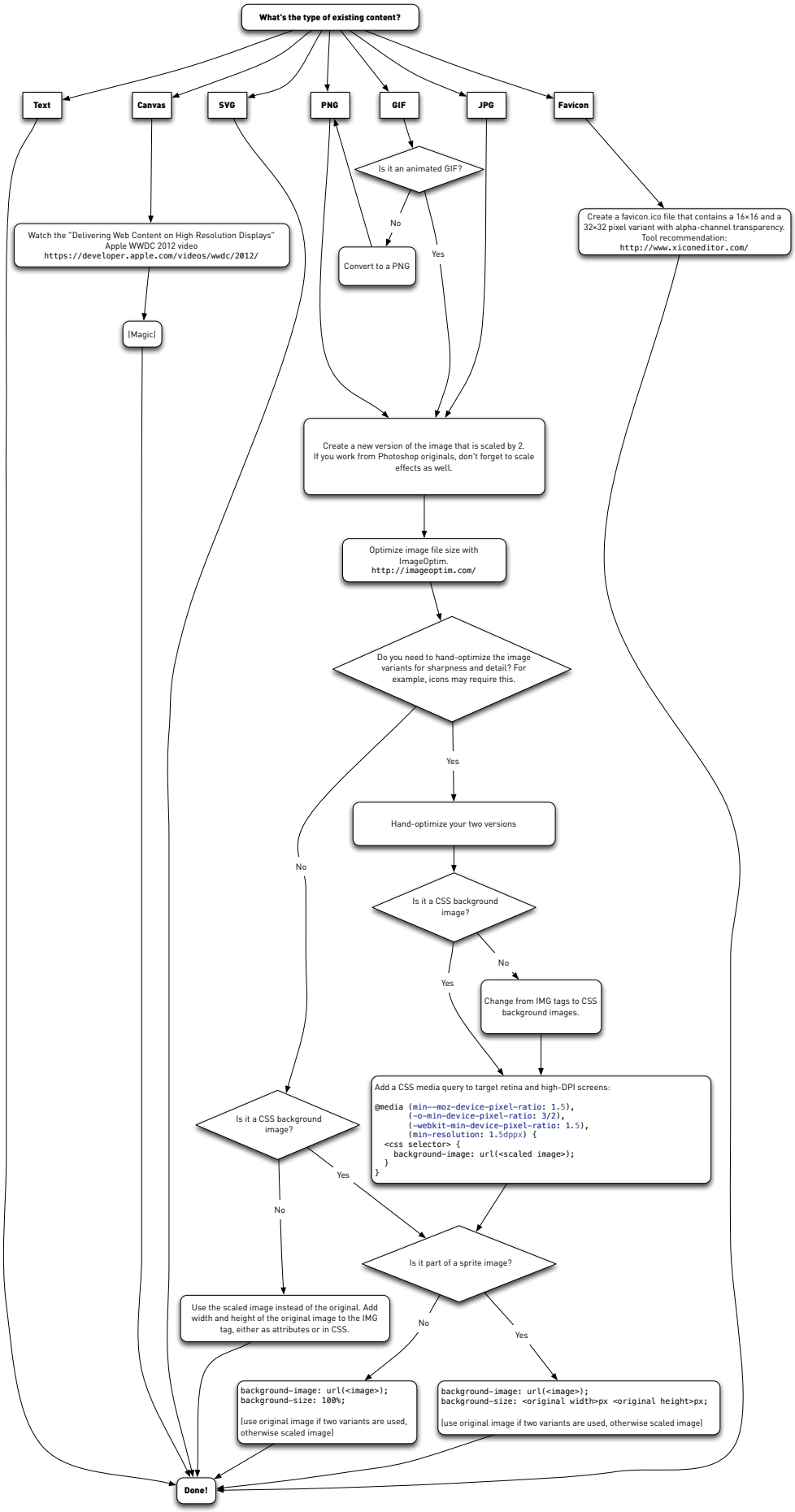
Page element	Notes
	CSS3 gradient, rounded border and shadow on login box. No images where harmed in the making.
	World icon. Hi-res PNG that is downscaled with width/height CSS attributes. The icon doesn't require separate lo/hi-res versions as it scales well.
	"Select" icon—a combination of CSS3 borders and gradients, and a PNG image with the arrows. Doesn't need lo/hi-res versions as it scales well. Background-size is explicitly set in CSS pixels for image-based layer.
	Tab, notification bubble and pointer. Using CSS only. The pointer is drawn with a unicode character ("▾") and uses "text-shadow" to complement the "box-shadow" of the tab.

Page element	Notes
	Avatars are sourced from Gravatar and are scaled JPEG images. The rounding is done via “border-radius” and the vignette is an overlaid div with an inset “box-shadow”. There’s no need to generate avatars in various sizes or preprocess images on the server.
	Charm logo in bar on top. Hi-res PNG that is downscaled with width/ height CSS attributes.
	Text scales automatically, with no change in metrics or other surprises. Browser icons are hi-res PNG images that are downscaled with the “width” and “height” CSS properties.
	Button is CSS3 (border, gradient, text-shadow). The icon is a hi-res PNGs that is used in a CSS background and scaled with “background-size” attributes.

Every Time Zone

Our one-page tool to easily compare the world's time zones was launched back in 2010, when the original iPad was released. I recently did a full rewrite and it fully supports retina screens on phones, tablets and desktop computers.




What's the type of existing content?
Text
Canvas
SVG
PNG
GIF
JPG
Favicon
Is it an animated GIF?
No
Yes
Convert to a PNG
Watch the "Delivering Web Content on High Resolution Displays" Apple WWDC 2012 video
https://developer.apple.com/videos/wwdc/2012/
Create a favicon.ico file that contains a 16×16 and a 32×32 pixel variant with alpha-channel transparency. Tool recommendation:
http://www.xiconeditor.com/
(Magic)
Create a new version of the image that is scaled by 2. If you work from Photoshop originals, don't forget to scale effects as well.
Optimize image file size with ImageOptim.
http://imageoptim.com/
Do you need to hand-optimize the image variants for sharpness and detail? For example, icons may require this.
Yes
No
Hand-optimize your two versions
Is it a CSS background image?
No
Yes
Change from IMG tags to CSS background images.
Add a CSS media query to target retina and high-DPI screens:
@media (min--moz-device-pixel-ratio: 1.5),
(-o-min-device-pixel-ratio: 3/2),
(-webkit-min-device-pixel-ratio: 1.5),
(min-resolution: 1.5dppx) {
<css selector> {
background-image: url(<scaled image>);
}
}
Is it a CSS background image?
Yes
No
Is it part of a sprite image?
Use the scaled image instead of the original. Add width and height of the original image to the IMG tag, either as attributes or in CSS.
No
Yes
background-image: url(<image>);
background-size: 100%;
(use original image if two variants are used, otherwise scaled image)
background-image: url(<image>);
background-size: <original width>px <original height>px;
(use original image if two variants are used, otherwise scaled image)
Done!