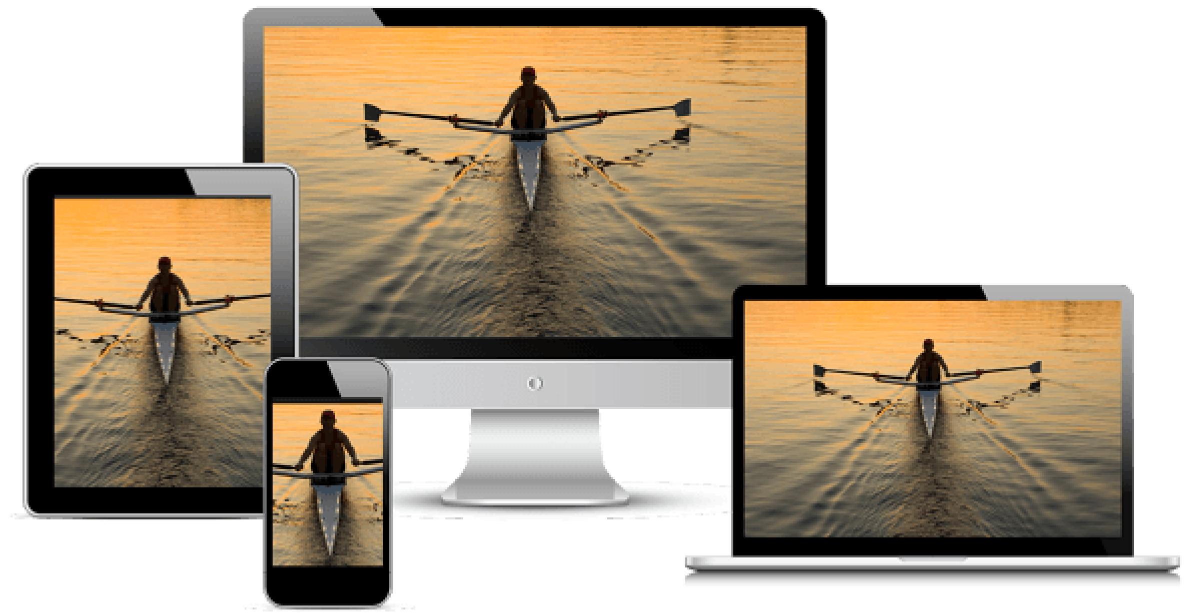
Images

By Pete LePage

Pete is a Developer Advocate

Responsive web design means that not only can our layouts change based on device characteristics, but content can change as well. For example, on high resolution (2x) displays, high resolution graphics ensure sharpness. An image that is 50% width may work just fine when the browser is 800px wide, but uses too much real estate on a narrow phone, and requires the same bandwidth overhead when scaled down to fit a smaller screen.

Art direction



Other times the image may need to be changed more drastically: changing the proportions, cropping, and even replacing the entire image. In this case, changing the image is usually referred to as art direction. See responsiveimages.org/demos/ for more examples.

Responsive Images

Did you know that images account for more than 60% of the bytes on average needed to load a web page?

In this course you will learn how to work with images on the modern web, so that your images look great and load quickly on any device.

Along the way, you will pick up a range of skills and techniques to smoothly integrate responsive images into your development workflow. By the end of the course, you will be developing with images that adapt and respond to different viewport sizes and usage scenarios.

This is a free course offered through Udacity

Take Course

Images in markup

The `img` element is powerful—it downloads, decodes, and renders content—and modern browsers support a range of image formats. Including images that work across devices is no different than for desktop, and only requires a few minor tweaks to create a good experience.

TL;DR

- Use relative sizes for images to prevent them from accidentally overflowing the container.
- Use the `picture` element when you want to specify different images depending on device characteristics (a.k.a. art direction).

- Use `srcset` and the `x` descriptor in the `img` element to give hints to the browser about the best image to use when choosing from different densities.
- If your page only has one or two images and these are not used elsewhere on your site, consider using inline images to reduce file requests.

Use relative sizes for images

Remember to use relative units when specifying widths for images to prevent them from accidentally overflowing the viewport. For example, `width: 50%;` causes the image width to be 50% of the containing element (not 50% of the viewport or 50% of actual pixel size).

Because CSS allows content to overflow its container, you may need to use max- width: 100% to prevent images and other content from overflowing. For example:

```
img, embed, object, video {
  max-width: 100%;
}
```

Be sure to provide meaningful descriptions via the `alt` attribute on `img` elements; these help make your site more accessible by giving context to screen readers and other assistive technologies.

Enhance `imgs` with `srcset` for high DPI devices

The `srcset` attribute enhances the behavior of the `img` element, making it easy to provide multiple image files for different device characteristics. Similar to the `image-set` CSS function native to CSS, `srcset` allows the browser to choose the best image depending on the characteristics of the device, for example using a 2x image on a 2x display, and potentially in the future, a 1x image on a 2x device when on a limited bandwidth network.

```
<img src="photo.png" srcset="photo@2x.png 2x" ...>
```

On browsers that don't support `srcset`, the browser simply uses the default image file specified by the `src` attribute. This is why it is important to always include a 1x image that can be displayed on any device, regardless of capabilities. When `srcset` is supported, the comma-separated list of image/conditions is parsed prior to making any requests, and only the most appropriate image is downloaded and displayed.

While the conditions can include everything from pixel density to width and height, only pixel density is well-supported today. To balance current behavior with future features, stick with simply providing the 2x image in the attribute.

Art direction in responsive images with `picture`

To change images based on device characteristics, also known as art direction, use the `picture` element. The `picture` element defines a declarative solution for providing multiple versions of an image based on different characteristics, like device size, device resolution, orientation, and more.

Dogfood: The `picture` element is beginning to land in browsers. Although it's not available in every browser yet, we recommend its use because of the strong backward compatibility and potential use of the Picturefill polyfill. See the ResponsiveImages.org site for further details.

Use the `picture` element when an image source exists in multiple densities, or when a responsive design dictates a somewhat different image on some types of screens. Similar to the **video** element, multiple **source** elements can be included, making it possible to specify different image files depending on media queries or image format.

```
<picture>
  <source media="(min-width: 800px)" srcset="head.jpg, head-2x.jpg 2x">
  <source media="(min-width: 450px)" srcset="head-small.jpg, head-small-2x.jpg 2x
  <img src="head-fb.jpg" srcset="head-fb-2x.jpg 2x" alt="a head carved out of woo
</picture>
```

Try it

In the above example, if the browser width is at least 800px then either **head.jpg** or **head-2x.jpg** is used, depending on the device resolution. If the browser is between 450px and 800px, then either **head-small.jpg** or **head-small- 2x.jpg** is used, again, depending on the device resolution. For screen widths less than 450px and backward compatibility where the **picture** element isn't supported, the browser renders the **img** element instead, and should always be included.

Relative sized images

When the final size of the image isn't known, it can be difficult to specify a density descriptor for the image sources. This is especially true for images that span a proportional width of the browser and are fluid, depending on the size of the browser.

Instead of supplying fixed image sizes and densities, you can specify the size of each supplied image by adding a width descriptor along with the size of the image element, allowing the browser to automatically calculate the effective pixel density and choose the best image to download.

```
<img src="lighthouse-200.jpg" sizes="50vw"
     srcset="lighthouse-100.jpg 100w, lighthouse-200.jpg 200w,
             lighthouse-400.jpg 400w, lighthouse-800.jpg 800w,
             lighthouse-1000.jpg 1000w, lighthouse-1400.jpg 1400w,
             lighthouse-1800.jpg 1800w" alt="a lighthouse">
```

Try it

The above example renders an image that is half the viewport width (**sizes="50vw"**), and depending on the width of the browser and its device pixel ratio, allows the browser to choose the correct image regardless of how large the browser window is. For example, the table below shows which image the browser would choose:

Browser width	Device pixel ratio	Image used	Effective resolution
400px	1	`200.png`	1x
400px	2	`400.png`	2x
320px	2	`400.png`	2.5x
600px	2	`800.png`	2.67x
640px	3	`1000.png`	3.125x
1100px	1	`1400.png`	1.27x

Account for breakpoints in responsive images

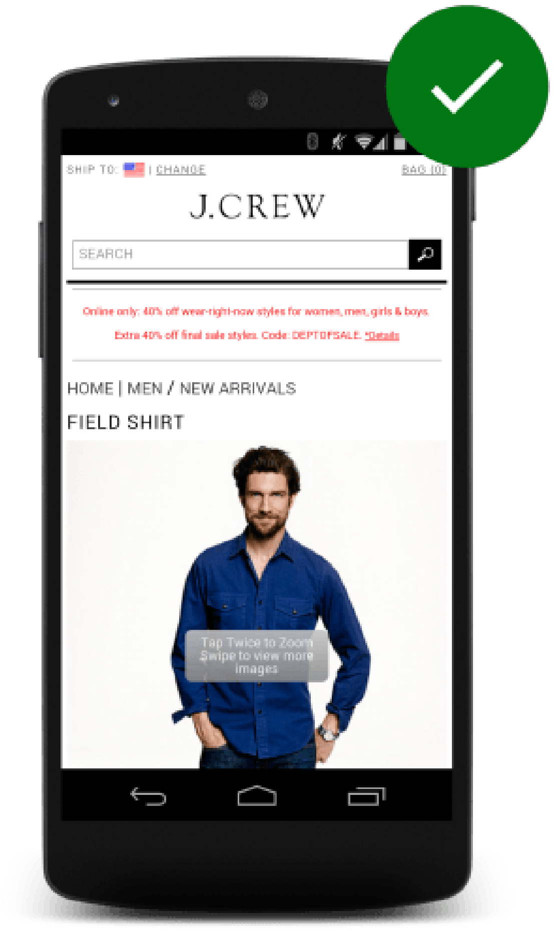
In many cases, the image size may change depending on the site's layout breakpoints. For example, on a small screen, you might want the image to span the full width of the viewport, while on larger screens, it should only take a small proportion.

```
<img src="400.png"
     sizes="(min-width: 600px) 25vw, (min-width: 500px) 50vw, 100vw"
     srcset="100.png 100w, 200.png 200w, 400.png 400w,
             800.png 800w, 1600.png 1600w, 2000.png 2000w" alt="an example image"
```

Try it

The `sizes` attribute, in the above example, uses several media queries to specify the size of the image. When the browser width is greater than 600px, the image is 25% of the viewport width; when it is between 500px and 600px, the image is 50% of the viewport width; and below 500px, it is full width.

Make product images expandable


SHIP TO: | CHANGE
BAG (0)
J.CREW
SEARCH
Online only: 40% off wear-right-now styles for women, men, girls & boys.
Extra 40% off final sale styles. Code: DEPTOFSALE. *Details
HOME | MEN / NEW ARRIVALS
FIELD SHIRT
Tap Twice to Zoom
Swipe to view more images

J. Crew's website with expandable product image.

Customers want to see what they're buying. On retail sites, users expect to be able to view high resolution closeups of products to get a better look at details, and study participants got frustrated if they weren't able to.

A good example of tappable, expandable images is provided by the J. Crew site. A disappearing overlay indicates that an image is tappable, providing a zoomed in image with fine detail visible.

Other image techniques

Compressive images

The compressive image technique serves a highly compressed 2x image to all devices, no matter the actual capabilities of the device. Depending on the type of image and level of compression, image quality may not appear to change, but the file size drops significantly.

Try it

Caution: Use caution with the compressive technique because of the increased memory and decoding costs it requires. Resizing large images to fit on smaller screens is expensive and can be particularly painful on low-end devices where both memory and processing is limited.

JavaScript image replacement

JavaScript image replacement checks the capabilities of the device and "does the right thing." You can determine device pixel ratio via `window.devicePixelRatio`, get screen width and height, and even potentially do some network connection sniffing via `navigator.connection` or issuing a fake request. When you've collected all of this information, you can decide which image to load.

One big drawback to this approach is that using JavaScript means that you will delay image loading until at least the look-ahead parser has finished. This means that images won't even start downloading until after the `pageload` event fires. In addition, the browser will most likely download both the 1x and 2x images, resulting in increased page weight.

Inlining images: raster and vector

There are two fundamentally different ways to create and store images—and this affects how you deploy images responsively.

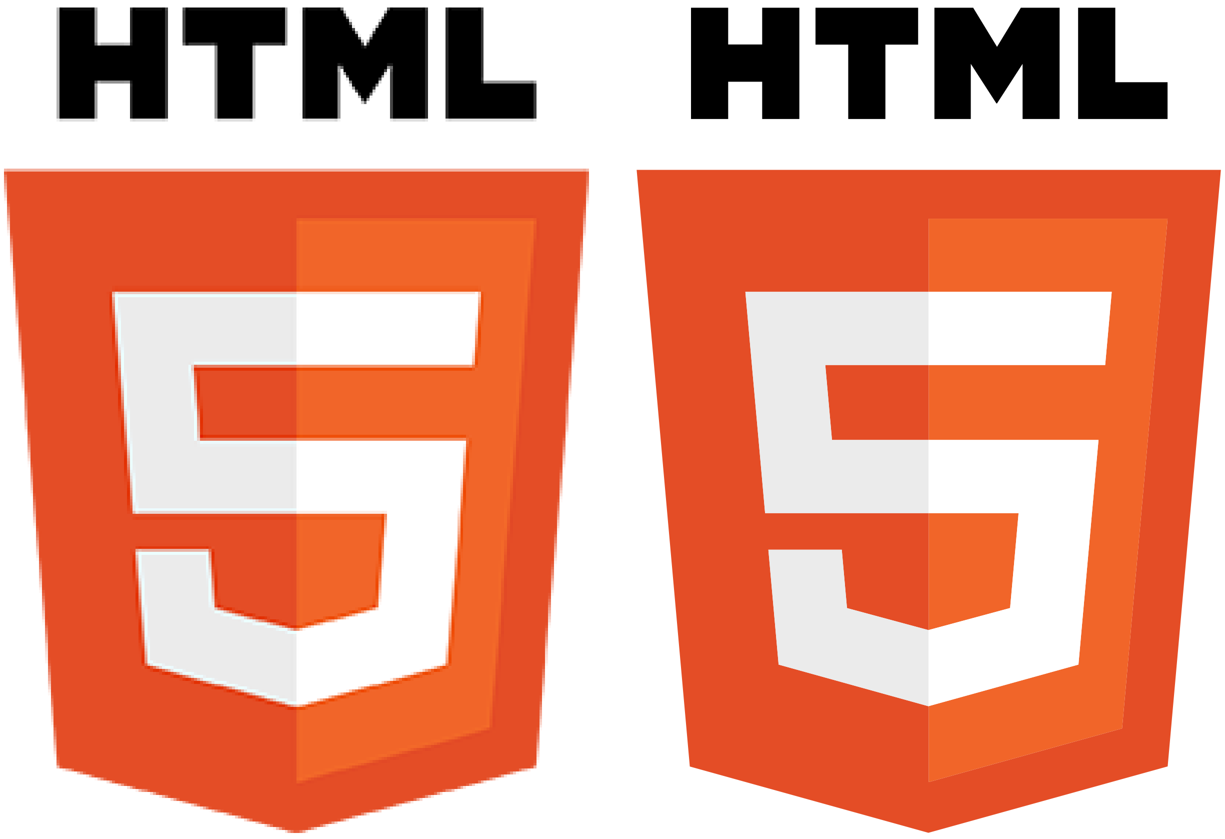
Raster images — such as photographs and other images, are represented as a grid of individual dots of color. Raster images might come from a camera or scanner, or be created with the HTML canvas element. Formats like PNG, JPEG, and WebP are used to store raster images.

Vector images such as logos and line art are defined as a set of curves, lines, shapes, fill colors and gradients. Vector images can be created with programs like Adobe Illustrator or Inkscape, or handwritten in code using a vector format such as SVG.

SVG

SVG makes it possible to include responsive vector graphics in a web page. The advantage of vector file formats over raster file formats is that the browser can render a vector image at any size. Vector formats describe the geometry of the image—how it's constructed from lines, curves, and colors and so on. Raster formats, on the other hand, only have information about individual dots of color, so the browser has to guess how to fill in the blanks when scaling.

Below are two versions of the same image: a PNG image on the left and an SVG on the right. The SVG looks great at any size, whereas the PNG next to it starts to look blurry at larger display sizes.



If you want to reduce the number of file requests your page makes, you can code images inline using SVG or Data URI format. If you view the source of this page, you'll see that both logos below are declared inline: a Data URI and an SVG.

SVG has great support on mobile and desktop, and optimization tools can significantly reduce SVG size. The following two inline SVG logos look identical, but one is around 3KB and the other only 2KB:

Data URI

Data URIs provide a way to include a file, such as an image, inline by setting the src of an `img` element as a Base64 encoded string using the following format:

```
<img src="data:image/svg+xml;base64,[data]">
```

The start of the code for the HTML5 logo above looks like this:

```
<img src="data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0
BZG9iZSBJbGx1c3RyYXRvciAxNi4wLjAsIFNWRyBFeHBvcnQgUGx1Zy1JbiAuIFNWRyBWZXJzaW ...">
```

(The full version is over 5000 characters in length!)

Drag 'n' drop tool such as jpillora.com/base64-encoder are available to convert binary files such as images to Data URIs. Just like SVGs, Data URIs are well supported on mobile and desktop browsers.

Inlining in CSS

Data URIs and SVGs can also be inlined in CSS—and this is supported on both mobile and desktop. Here are two identical-looking images implemented as background images in CSS; one Data URI, one SVG:

Inlining pros & cons

Inline code for images can be verbose—especially Data URIs—so why would you want to use it? To reduce HTTP requests! SVGs and Data URIs can enable an entire web page, including images, CSS and JavaScript, to be retrieved with one single request.

On the downside:

- On mobile, Data URIs can be significantly slower to display on mobile than images from an external `src`.
- Data URIs can considerably increase the size of an HTML request.

- They add complexity to your markup and your workflow.
- The Data URI format is considerably bigger than binary (up to 30%) and therefore doesn't reduce total download size.
- Data URIs cannot be cached, so must be downloaded for every page they're used on.
- They're not supported in IE 6 and 7, incomplete support in IE8.
- With HTTP/2, reducing the number of asset requests will become less of a priority.

As with all things responsive, you need to test what works best. Use developer tools to measure download file size, the number of requests, and the total latency. Data URIs can sometimes be useful for raster images—for example, on a homepage that only has one or two photos that aren't used elsewhere. If you need to inline vector images, SVG is a much better option.

Images in CSS

The CSS `background` property is a powerful tool for adding complex images to elements, making it easy to add multiple images, and causing them to repeat, and more. When combined with media queries, the background property becomes even more powerful, enabling conditional image loading based on screen resolution, viewport size, and more.

TL;DR

- Use the best image for the characteristics of the display, consider screen size, device resolution, and page layout.
- Change the `background-image` property in CSS for high DPI displays using media queries with `min-resolution` and `-webkit-min-device-pixel-ratio`.
- Use srcset to provide high resolution images in addition to the 1x image in markup.
- Consider the performance costs when using JavaScript image replacement techniques or when serving highly compressed high resolution images to lower resolution devices.

Use media queries for conditional image loading or art direction

Media queries not only affect the page layout; you can also use them to conditionally load images or to provide art direction depending on the viewport width.

For example, in the sample below, on smaller screens only `small.png` is downloaded and applied to the content `div`, while on larger screens `background-image: url(body.png)` is applied to the body and **`background-image: url(large.png)`** is applied to the content **`div`**.

```
.example {
  height: 400px;
  background-image: url(small.png);
  background-repeat: no-repeat;
  background-size: contain;
  background-position-x: center;
}

@media (min-width: 500px) {
  body {
    background-image: url(body.png);
  }
  .example {
    background-image: url(large.png);
  }
}
```

Try it

Use image-set to provide high res images

The **`image-set()`** function in CSS enhances the behavior **`background`** property, making it easy to provide multiple image files for different device characteristics. This allows the browser to choose the best image depending on the characteristics of the device, for example using a 2x image on a 2x display, or a 1x image on a 2x device when on a limited bandwidth network.

```
background-image: image-set(
  url(icon1x.jpg) 1x,
  url(icon2x.jpg) 2x
);
```

In addition to loading the correct image, the browser also scales it accordingly. In other words, the browser assumes that 2x images are twice as large as 1x images, and so scales the 2x image down by a factor of 2, so that the image appears to be the same size on the page.

Support for **`image-set()`** is still new and is only supported in Chrome and Safari with the **`-webkit`** vendor prefix. Take care to include a fallback image for when **`image-set()`** is not supported; for example:

```
.sample {
  width: 128px;
  height: 128px;
  background-image: url(icon1x.png);
  background-image: -webkit-image-set(
    url(icon1x.png) 1x,
    url(icon2x.png) 2x
  );
  background-image: image-set(
    url(icon1x.png) 1x,
    url(icon2x.png) 2x
  );
}
```

Try it

The above loads the appropriate asset in browsers that support image-set; otherwise it falls back to the 1x asset. The obvious caveat is that while `image-set()` browser support is low, most browsers get the 1x asset.

Use media queries to provide high res images or art direction

Media queries can create rules based on the device pixel ratio, making it possible to specify different images for 2x versus 1x displays.

```
@media (min-resolution: 2dppx),
(-webkit-min-device-pixel-ratio: 2)
{
  /* High dpi styles & resources here */
}
```

Chrome, Firefox, and Opera all support the standard `(min-resolution: 2dppx)`, while the Safari and Android browsers both require the older vendor prefixed syntax without the `dppx` unit. Remember, these styles are only loaded if the device matches the media query, and you must specify styles for the base case. This also provides the benefit of ensuring something is rendered if the browser doesn't support resolution-specific media queries.

```
.sample {
  width: 128px;
  height: 128px;
  background-image: url(icon1x.png);
}

@media (min-resolution: 2dppx), /* Standard syntax */
```

```
(-webkit-min-device-pixel-ratio: 2)  /* Safari & Android Browser */
{
  .sample {
    background-size: contain;
    background-image: url(icon2x.png);
  }
}
```

Try it

You can also use the min-width syntax to display alternative images depending on the viewport size. This technique has the advantage that the image is not downloaded if the media query doesn't match. For example, `bg.png` is only downloaded and applied to the **body** if the browser width is 500px or greater:

```
@media (min-width: 500px) {
  body {
    background-image: url(bg.png);
  }
}
```

Use SVG for icons

When adding icons to your page, use SVG icons where possible or in some cases, unicode characters.

TL;DR

- Use SVG or unicode for icons instead of raster images.

Replace simple icons with unicode

Many fonts include support for the myriad of unicode glyphs, which can be used instead of images. Unlike images, unicode fonts scale well and look good no matter how small or large they appear on screen.

Beyond the normal character set, unicode may include symbols for arrows (←), math operators (√), geometric shapes (★), control pictures (▶), music notation (♬), Greek letters (Ω), even chess pieces (♞).

Including a unicode character is done in the same way named entities are: &#XXXX, where XXXX represents the unicode character number. For example:

```
You're a super &#9733;
```

You're a super ★

Replace complex icons with SVG

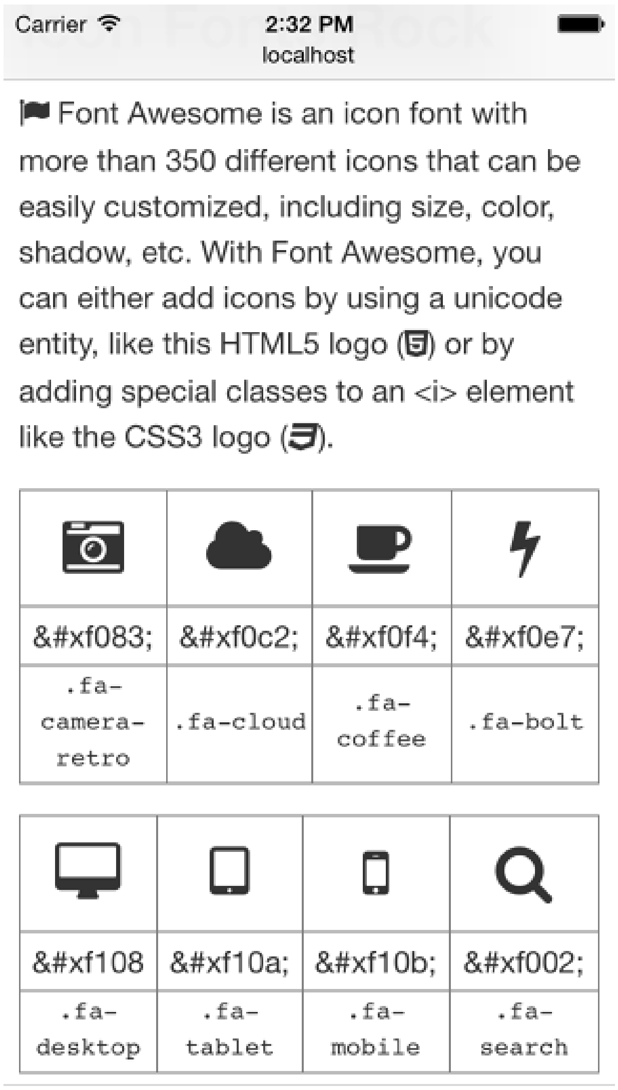
For more complex icon requirements, SVG icons are generally lightweight, easy to use, and can be styled with CSS. SVG have a number of advantages over raster images:

- They're vector graphics that can be infinitely scaled.
- CSS effects such as color, shadowing, transparency, and animations are straightforward.
- SVG images can be inlined right in the document.
- They are semantic.
- They provide better accessibility with the appropriate attributes.

```
With SVG icons, you can either add icons using inline SVG, like
this checkmark:
  <svg version="1.1" xmlns="http://www.w3.org/2000/svg"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       width="32" height="32" viewBox="0 0 32 32">
    <path d="M27 4l-15 15-7-7-5 5 12 12 20-20z" fill="#000000"></path>
  </svg>
or by using an image tag, like this credit card icon:
<img src="credit.svg">.
```

Try it

Use icon fonts with caution



Carrier 2:32 PM localhost

Font Awesome is an icon font with more than 350 different icons that can be easily customized, including size, color, shadow, etc. With Font Awesome, you can either add icons by using a unicode entity, like this HTML5 logo () or by adding special classes to an <i> element like the CSS3 logo ().

📷	☁	☕	⚡
&#xf083;	&#xf0c2;	&#xf0f4;	&#xf0e7;
`.fa-camera-retro`	`.fa-cloud`	`.fa-coffee`	`.fa-bolt`

🖥	▯	📱	🔍
&#xf108	&#xf10a;	&#xf10b;	&#xf002;
`.fa-desktop`	`.fa-tablet`	`.fa-mobile`	`.fa-search`

Example of a page that uses FontAwesome for its font icons.

Icon fonts are popular, and can be easy to use, but have some drawbacks compared to SVG icons:

- They're vector graphics that can be infinitely scaled, but may be anti-aliased resulting in icons that aren't as sharp as expected.
- Limited styling with CSS.
- Pixel perfect positioning can be difficult, depending on line-height, letter spacing, etc.
- They aren't semantic, and can be difficult to use with screen readers or other assistive technology.
- Unless properly scoped, they can result in a large file size for only using a small subset of the icons available.

```
With Font Awesome, you can either add icons by using a unicode
entity, like this HTML5 logo (<span class="awesome">&#xf13b;</span>)
```

```
or by adding special classes to an &lt;i&gt; element like the CSS3
logo (<i class="fa fa-css3"></i>).
```

Try it

There are hundreds of free and paid icon fonts available including Font Awesome, Pictos, and Glyphicons.

Be sure to balance the weight of the additional HTTP request and file size with the need for the icons. For example, if you only need a handful of icons, it may be better to use an image or an image sprite.

Optimize images for performance

Images often account for most of the downloaded bytes and also often occupy a significant amount of the visual space on the page. As a result, optimizing images can often yield some of the largest byte savings and performance improvements for your website: the fewer bytes the browser has to download, the less competition there is for client's bandwidth and the faster the browser can download and display all the assets.

TL;DR

- Don't just randomly choose an image format—understand the different formats available and use the format best suited.
- Include image optimization and compression tools into your workflow to reduce file sizes.
- Reduce the number of http requests by placing frequently used images into image sprites.
- To improve the initial page load time and reduce the initial page weight, consider loading images only after they’ve scrolled into view.

Choose the right format

There are two types of images to consider: vector images and raster images. For raster images, you also need to choose the right compression format, for example: `GIF`, `PNG`, `JPG`.

Raster images, like photographs and other images, are represented as a grid of individual dots or pixels. Raster images typically come from a camera or scanner, or can be created in the browser with the `canvas` element. As the image size gets larger, so does the file size.

When scaled larger than their original size, raster images become blurry because the browser needs to guess how to fill in the missing pixels.

Vector images, such as logos and line art, are defined by a set of curves, lines, shapes, and fill colors. Vector images are created with programs like Adobe Illustrator or Inkscape and saved to a vector format like `SVG`. Because vector images are built on simple primitives, they can be scaled without any loss in quality or change in file size.

When choosing the appropriate format, it is important to consider both the origin of the image (raster or vector), and the content (colors, animation, text, etc). No one format fits all image types, and each has its own strengths and weaknesses.

Start with these guidelines when choosing the appropriate format:

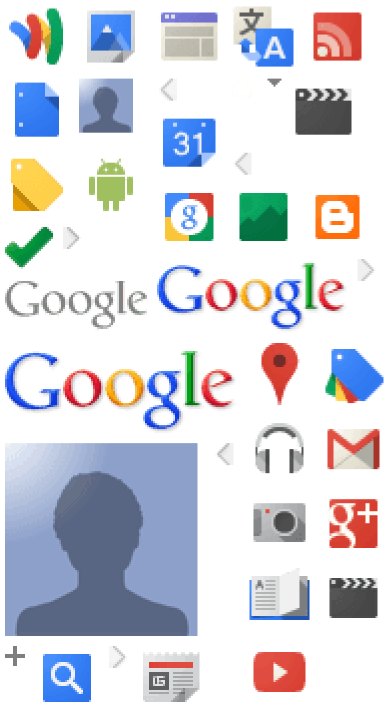
- Use `JPG` for photographic images.
- Use `SVG` for vector art and solid color graphics such as logos and line art. If vector art is unavailable, try `WebP` or `PNG`.
- Use `PNG` rather than `GIF` as it allows for more colors and offers better compression ratios.
- For longer animations consider using `<video>`, which provides better image quality and gives the user control over playback.

Reduce the file size

You can reduce image file size considerably by "post-processing" the images after saving. There are a number of tools for image compression—lossy and lossless, online, GUI, command line. Where possible, it's best to try automating image optimization so that it's a first-class citizen in your workflow.

Several tools are available that perform further, lossless compression on `JPG` and `PNG` files with no effect on image quality. For `JPG`, try jpegtran or jpegoptim (available on Linux only; run with the –strip-all option). For `PNG`, try OptiPNG or PNGOUT.

Use image sprites



CSS spriting is a technique whereby a number of images are combined into a single "sprite sheet" image. You can then use individual images by specifying the background image for an element (the sprite sheet) plus an offset to display the correct part.

```
.sprite-sheet {
  background-image: url(sprite-sheet.png);
  width: 40px;
  height: 25px;
}

.google-logo {
  width: 125px;
  height: 45px;
  background-position: -190px -170px;
}

.gmail {
  background-position: -150px -210px;
}

.maps {
  height: 40px;
  background-position: -120px -165px;
}
```

Try it

Spriting has the advantage of reducing the number of downloads required to get multiple images, while still enabling caching.

Consider lazy loading

Lazy loading can significantly speed up loading on long pages that include many images below the fold by loading them either as needed or when the primary content has finished loading and rendering. In addition to performance improvements, using lazy loading can create infinite scrolling experiences.

Be careful when creating infinite scrolling pages—because content is loaded as it becomes visible, search engines may never see that content. In addition, users who are looking for information they expect to see in the footer, never see the footer because new content is always loaded.

Avoid images completely

Sometimes the best image isn't actually an image at all. Whenever possible, use the native capabilities of the browser to provide the same or similar functionality. Browsers generate visuals that would have previously required images. This means that browsers no longer need to download separate image files thus preventing awkwardly scaled images. You can use unicode or special icon fonts to render icons.

Place text in markup instead of embedded in images

Wherever possible, text should be text and not embedded into images. For example, using images for headlines or placing contact information—like phone numbers or addresses—directly into images prevents users from copying and pasting the information; it makes the information inaccessible for screen readers, and it isn't responsive. Instead, place the text in your markup and if necessary use webfonts to achieve the style you need.

Use CSS to replace images

Modern browsers can use CSS features to create styles that would previously have required images. For example: complex gradients can be created using the `background` property, shadows can be created using `box-shadow`, and rounded corners can be added with the `border-radius` property.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque sit amet augue eu magna scelerisque porta ut ut dolor. Nullam placerat egestas nisl sed sollicitudin. Fusce placerat, ipsum ac vestibulum porta, purus dolor mollis nunc, pharetra vehicula nulla nunc quis elit. Duis ornare fringilla dui non vehicula. In hac habitasse platea dictumst. Donec ipsum lectus, hendrerit malesuada sapien eget, venenatis tempus purus.

```
<style>
  div#noImage {
    color: white;
    border-radius: 5px;
    box-shadow: 5px 5px 4px 0 rgba(9,130,154,0.2);
    background: linear-gradient(rgba(9, 130, 154, 1), rgba(9, 130, 154, 0.5));
  }
</style>
```

Keep in mind that using these techniques does require rendering cycles, which can be significant on mobile. If over-used, you'll lose any benefit you may have gained and it may hinder performance.

Last updated July 2, 2018.