Download the files here!

All the files you need for this tutorial can be found at *netm.ag/responsive-255*

ABOUT THE AUTHOR

ANDI SMITH

w: *andismith.com*

t: @andismith

areas of expertise: Frontend development

q: When was the last time you cried?

a: xxx xxx x x xxxx xxxx xxx xxxxxxx x xx xxx xx xxxx x xxxx xx x xxxx

RWD

AUTOMATING RESPONSIVE IMAGES

Andi Smith explores techniques for integrating responsive images into your site and reveals how to use Grunt to generate them automatically

Developers are obsessed with file size. When we look for a plug-in, we hunt for hours for the one with the fewest kilobytes. We rewrite our entire site so we don't have to include the 'bloated' jQuery library. We minify, concatenate and uglify our JavaScript until it becomes deliberately unreadable. And then we put our tiny, tiny JavaScript file on the same page as a 400kb image.

The overall file size of a page, known as the 'page weight', is a major issue for all devices, as the success of your site may be determined by its load time, but it is of particular concern when there is a poor connection speed. If you are sending an image with a resolution higher than that of the device's screen, a proportion of the data you are sending is completely unused: if the dimensions of the image are twice that of the screen, 75 per cent of the data goes to waste. It also applies additional pressure to the device's processors, responsible for scaling the image to fit the screen. When you start adding rich animations to your page, scaling becomes especially problematic.

Responsive design techniques have solved many of the layout issues plaguing mobile users, but there are still challenges to overcome when it comes to images:

- Ensuring that the most appropriate version of an image is downloaded for the current device
- Avoiding downloading multiple versions of an image
- Determining the right time during page load to choose and load a version of an image
- Keeping markup concise
- Browser support

VIDEO

Watch an exclusive screencast of this tutorial created by the author: *netm.ag/tut5-255*

Over the last few years, three different techniques have been proposed for foreground responsive images: `srcset`, `srcN` and the `<picture>` element. Recently representatives from each of the browsers have settled on using the `<picture>` element.

Currently `<picture>` can only be emulated with a polyfill, but browser support is expected to follow in the coming months. A recent Indiegogo campaign (*netm.ag/blink-255*) has ensured the syntax will be implemented in the Blink renderer used in Chrome and Opera, and a stretch goal will cover WebKit and Safari; while the Firefox team is also actively developing its own implementation.

As detailed in Mat Marquis' article on the previous page, the `<picture>` element uses standard media queries expressions for fine control over which image to display. Based on the matching media query, it uses a simplified version of `srcset` to determine the image source to load. You can serve different images for screens with different pixel densities by adding a parameter after the filename:

```
<picture>
    <source media="(min-width: 40em)" srcset="dog-large.
    jpg 1x, dog-large-hd.jpg 2x" />
    <source srcset="dog-small.jpg 1x, dog-small-hd.jpg 2x" />
    <img src="dog-small.jpg" alt="Harry Beagle" />
</picture>
```

Additionally, we can stop our images from overflowing from their containing box by setting a `max-width` value in our CSS:

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

If the image is not part of the page content or it is not critical to interaction but is instead part of the design, the simplest option is to include it with the `background` or `background-image` CSS property and then use media queries to load in the correct image size. We can supplement this technique with the `background-size` property to ensure our image fits the element it is contained within:

```
.container {
    background: url(/img/bg-container-small.jpg) no-repeat
    top left;
    background-size: cover;
}
@media screen and (min-width: 768px) {
    .container {
        background-image: url(/img/bg-container-large.jpg);
    }
}
```

GENERATING RESPONSIVE IMAGES

Generating a set of responsive images for your entire site manually would make it difficult to maintain. Thankfully, we can automate this process using Grunt and a task called Grunt Responsive Images (*netm.ag/grs-255*). This matches a set of files and folders and resizes the images to the dimensions you require.

Generating responsive images manually would make sites difficult to maintain

If you're unfamiliar with Grunt, it is a task runner that automates the tiresome parts of frontend development such as generating Sass files, linting JavaScript and minification. Chris Coyier has written a good article on getting started: *netm.ag/grunt-255*.

Once you have Grunt up and running in your project, adding the Grunt Responsive Images task is straightforward. Navigate to your project's directory in Terminal or Command Prompt, then type:

```
npm install grunt-responsive-images --save-dev
```

This will download the task and save it to your `packages.json` file. You also need to download an image-processing client to do the heavy

▶

FOCUS ON

THE RETINA REVOLUTION TECHNIQUE

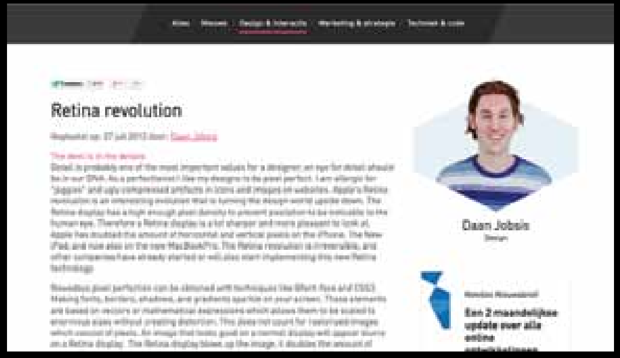
The 'Retina revolution' technique was first documented by Daan Jobsis (*netm.ag/retina-255*). Through a number of tests, he discovered that heavy compression doesn't significantly affect the quality of images with a large number of pixels, even when the file size of a high-resolution image is reduced below that of the base version. The image is then scaled within the browser, which results in a sharper image with few noticeable artifacts.

Moving forward with this discovery, one of the optimum strategies found was to save the images at 2.2 times the resolution required with a very low image quality.

To do this technique with Grunt Responsive Images, you need to enable upscaling and set a `quality` value to 25%. For example:

```
options: {
    sizes: [{ height: 440, name: 'small', quality: 25, upscale:
    true, width: 880 },
    { height: 880, name: 'medium', quality: 25, upscale:
    true, width: 1760 },
    { height: 1320, name: 'large', quality: 25, upscale:
    true, width: 2200 }]
},
```

Be careful with scaling images: they require the device to run extra processing, so if you have to support older devices or you have rich animations on your site, you may experience lower frame rates.



Retina revolution

[illegible]

Daan Jobsis

Een 2 maandelijkse update over alle online

More details Daan Jobsis's original article on the Retina revolution technique processing. Grunt Responsive Images can use either GraphicsMagick (*graphicsmagick.org*) or ImageMagick (*imagemagick.org*). GraphicsMagick is the faster of the two libraries, and the default library used. If you're using a Mac and Homebrew you can install it with:

```
brew install GraphicsMagick
```

Alternatively, install it from the official website. Once GraphicsMagick is installed, open up your `Gruntfile.js` and add the following line to load the task:

```
grunt.loadNpmTasks('grunt-responsive-images');
```

Next, add a section named `responsive_images` to the data object passed into `grunt.initConfig()`:

```
  grunt.initConfig({
      ...
      responsive_images: {
      }
  });
```

Within this section, we can specify multiple tasks, which is useful if we want to apply different rules to different sets of images. For example, you may wish to resize your blog's images to one size and the images in your homepage carousel to another. Let's add a task for the homepage carousel:

```
  responsive_images: {
      carousel: {
          options: {
              sizes: [{ height: 200, width: 400 },
                  { height: 400, width: 800 },
                  { height: 600, width: 1000 }]
          },
          files: []
      }
  }
```

Automated resize Using Grunt Responsive Images, you can resize and crop your images to match your site's requirements

Our carousel task specifies three sizes for our images: 400 x 200, 800 x 400 and 1,000 x 600. By default these sizes are treated as maximum value boundaries, so the image will be resized until the first of these values has been matched. We do not have to specify both dimensions: the task can resize with just one and maintain the aspect ratio of the image.

Our task also specifies the path where we can find the source images to resize, using four parameters:

- `expand` allows us to build the files list dynamically
- `cwd` specifies the original path where we will find the files specified in `src`
- `dest` specifies the destination path to save our new images to
- `src` specifies the image file paths themselves, relative to `cwd`. Any paths we include in here will be copied to the destination structure. We can use `**/*` to specify any images that sit in this directory or any directories below the path we have specified so far; and we can use `.{gif,jpg,png}` to specifically target files with the extensions .gif, .jpg or .png.

```
  files: [{
      expand: true,
      src: ['images/carousel/**/*.{jpg,gif,png}'],
      cwd: './src/',
      dest: './dest/'
  }]
```

It's important the `src` and `dest` paths are different, or when the task is run a second time, it will process the images generated by the previous run.

Now run the task by going to Terminal or Command Prompt, navigating to the project directory and typing:

```
grunt responsive_images
```

The task will scan the source path and resize any images it finds, saving the output to the destination path we specified. By default, the filenames will be suffixed with `-400x200`, `-800x400` and `-1000x600`. To change this, we can add a `name` property to our size options. This has the advantage that if you change the dimensions at a later date, you do not need to change the HTML where you include these images.

```
  options: {
      sizes: [{ height: 200, name: 'small', width: 400 },
          { height: 400, name: 'medium', width: 800 },
          { height: 600, name: 'large', width: 1000 }]
  },
```

GraphicsMagick Image Processing System

Power tool The GraphicsMagick library powers Grunt Responsive Images

The images generated will have kept their aspect ratio by default, and consequently will have used the width and height parameters specified as maximum value boundaries. This is useful for images within articles, but may not be as useful for a carousel where we would expect our images to have a fixed height.

Grunt Responsive Images generates responsive images for your site automatically

We can tell the Grunt Responsive Images task to resize and crop our image in order to maintain a fixed size by adding an `aspectRatio` property and setting it to `false`. We can control the crop by specifying a `gravity` value. By default, `gravity` is set to `'Center'`, but it can be set to `'North'`, `'South'`, `'East'`, `'West'`, or a combination of these, like `'NorthWest'`:

```
  options: {
      sizes: [{ aspectRatio: false, gravity: 'NorthWest',
      height: 200, name: 'small', width: 400 },
          { aspectRatio: false, gravity: 'Center', height: 400,
          name: 'medium', width: 800 },
          { aspectRatio: false, gravity: 'Center', height: 600,
          name: 'large', width: 1000 }]
  },
```

Now we have our images correctly sized and ready for our responsive site, we can add them to our carousel using one of the responsive imaging techniques mentioned previously.

You can read more about Grunt Responsive Images on the task's GitHub page (*netm.ag/grs-255*). Many more options are available and new options are being added all the time.

RESOURCES

FURTHER INFORMATION

If you would like to know more about responsive images, the following links discuss the techniques covered in this article in more detail:

More information:

- The Grunt Responsive Images website: *andismith.com/grunt-responsive-images*
- The `srcset` specification: *dev.w3.org/html5/srcset*
- The `<picture>` specification: *picture.responsiveimages.org*
- The Responsive Images Community Group, a group dedicated to finding responsive image solutions: *responsiveimages.org*
- Ethan Marcotte's article on fluid images: *netm.ag/fluidimages-255*

More responsive image solutions:

- The adaptive images technique, a server-side responsive image technique: *adaptive-images.com*
- ImagerJS, a temporary solution for displaying responsive images created by BBC News: *github.com/BBC-News/Imager.js*
- HiSRC, a jQuery adaptive image plug-in: *github.com/1Marc/hisrc*
- A up-to-date comparison chart showing the pros and cons of different responsive image techniques: *netm.ag/chart-255*

Image-processing libraries:

Action group The Responsive Images Community Group (RICG) is a group of developers dedicated to finding markup-based responsive image solutions