Image Compression

Our build can help us optimize our images in many different ways, the most obvious of which is for file size. I've gone ahead and created a task that copies images to our distribution directory. We're then piping that output into our optimization plugins. We can compress images with either lossless or lossy compression algorithms. Lossless compression reduces a file in such a way that the original can be recreated from the compressed version. You can think of it as reducing the file size but not throwing away any information.

Imagemin

gulp-imagemin can losslessly compress JPEGs, GIFS, PNGs and SVGs out of the box. Lossless means that even though the file size will end up being smaller, special care is taken to not cause any visual changes whatsoever, meaning that original visual information stays exactly the same.

After you've grabbed the plugin you can simply add a pipe between the new crunch-images task and call `imagemin()` in there. There are a few extra options such as generating progressive images, but even without any configuration this will take all of your images and do any safe optimizations.

Lossy Compression

Lossy compression, on the other hand, can only recreate an approximation of the original. Lossy compression can give you really small file sizes at the expense of image quality. But there are a few lossy optimizations that are truly smart, and PNG quantization is one of them. PNG quantization takes images with or without alpha transparency and converts them to 256 or less colored 8-bit pngs. Now if you do this manually and just convert a 16-bit image to a 8-bit image, you won't like the results. It'll end up...well..like a crappy gif, with unnatural, limited colors.

PNG Quantization

PNG quantization benefits from the fact that there are colors that our vision and brain perceives as very similar, even though they're technically completely different. The quantization algorithm aims to understand which colors actually matter and remaps them to new, optimized colors.

A cool thing about pngquant, the plugin we're going to use, is that it automatically exits and will not save if a certain quality threshold isn't passed.

Let's Try It

1. Download and require the imagemin-pngquant plugin in addition to gulp-imagemin.
2. Create a config object for imagemin. These are the directives that imagemin will use when you pipe images to it. The following snippet instructs imagemin to use progressive rendering for JPEG images and PNG quant for well, PNGs.

```
gulp.task('default', function() {
    return gulp.src('src/images/*')
        .pipe(imagemin({
            progressive: true,
            use: [pngquant()]
        }))
        .pipe(gulp.dest('dist/images'));
});
```

Progressive rendering loads an image in layers where each layer makes the image more detailed. It can make a page feel faster than typical rendering line by line. If you like, you can now configure pngquant as well by adding quality or speed options. Read more about these on the plugin homepage.

Now you've got automatic image crunching in place and working for you but pro-tip, for anything important, take the time to see what will work, even if that means putting in a bit of elbow grease and checking things manually.

Even better compression options

Smaller images can tolerate more aggressive lossy compression. You might want to try other things like converting images to SVG where applicable. SVG stands for Scalable Vector Graphics and uses a XML-based format to describe an image and can in most cases be scaled infinitely without any increase in file size or responsive and fit retina and non-retina screens, or inlining your images into your CSS or into a sprite to save a couple more HTTP requests.

And go check out Sam and Cameron's course!

NEXT

Mentorship
Get support and stay on track