Preoptimizing Your Images

Gatsby ships with excellent image optimization capabilities (see the image tutorial for more info). However, this image optimization can come with a cost. It can be fairly CPU intensive, and in some cases may lead to long build times. As a means of debugging and perhaps improving your overall build performance, it *may* be helpful to pre-optimize your (extremely large) images.

First, some context. `gatsby-plugin-sharp` ships with a `fluid` option which will attempt to create four images intended to map to various screen resolutions. Producing multiple images ensures that your images are ready and optimized for phone displays, desktop displays, and everything in between. This plugin receives a non-optimized image and produces optimized images for *all* of your users and devices. Magic!

If you find yourself running into build performance issues, it could be helpful to consider applying some image optimizations. Images in your setup *could* be overly large, especially compared to the requested image sizes in your layout (e.g. if your layout has a max width of 600 pixels).

For instance, if your layout is 600 pixels wide, then the highest resolution image you will need is 1200 pixels to account for 2x pixel density. If you have images that are 3000 or 4000 pixels wide then you could resize your image to 1800 pixels, which may improve overall build performance.

How to pre-optimize your images

Here's an example script to pre-optimize your image dimensions and slightly compress. This optimization may serve as a helpful technique to possibly improve build time if your local repository has many, very large images.

```
const sharp = require(`sharp`)
const glob = require(`glob`)
const fs = require(`fs-extra`)

const matches = glob.sync(`src/images/**/*.{png,jpg,jpeg}`)
const MAX_WIDTH = 1800
const QUALITY = 70

Promise.all(
```

```
  matches.map(async match => {
    const stream = sharp(match)
    const info = await stream.metadata()

    if (info.width < MAX_WIDTH) {
      return
    }

    const optimizedName = match.replace(
      /(\..+)$/,
      (match, ext) => `-optimized${ext}`
    )

    await stream
      .resize(MAX_WIDTH)
      .jpeg({ quality: QUALITY })
      .toFile(optimizedName)

    return fs.rename(optimizedName, match)
  })
)
```

The variables `MAX_WIDTH` and `QUALITY` should be configured in regards to your particular website's layout. Quality can be lowered further for perhaps more gains in performance, but be aware that it can introduce visual differences between the original and optimized images, **so be sure to compare before over optimizing**.