15:13 The Gamma Trick: Two PNGs for the price of one

by Hector Martin 'marcan'

Say you're browsing your favorite hypertext-encoded, bitmap-containing visuo-lingual information distribution medium. You come across an image which—as we do not yet live in an era of infinitely scalable resolution—piques your interest yet is presented as a small thumbnail. *Why are they called thumbnails, anyway?*

Despite the clear instructions not to do so, you resolve to click, tap, press enter, or otherwise engage with the image. After all, you have been conditioned to expect that such an action will yield a higher-quality image through some opaque and clearly incomprehensible process.

Yet the image now appearing before your eyes is not the same image that you clicked on. Curses! What is this sorcery? Have I been fooled? Is this alien technology? *Did someone hack Reddit?*

The first time I came across this technique was a few years ago on a post on 4chan. Despite the fact that the image was not just lewd but downright unsavory to my taste, I have to admit I spent quite some time analysing exactly what was going on in detail. I have since seen this trick used a few times here and there, and indeed I've even used a variant of it myself in a CTF challenge. Thanks go to my friend `@Miluda` for giving me permission to use her art in this article's examples.

So, do tell, what is going on? It all has to do with the PNG format. Like most image formats, PNG images carry metadata. That metadata includes information about how the image, and in particular color information, is itself encoded. The PNG format can specify how RGB values map to how much light comes out of the pixels on your screen in several ways, but one of the simplest is the `'gAMA'` chunk which specifies the gamma value of the image, γ.

Intuitively, you'd think that a pixel with 50% brightness would be encoded as a 0.5 value (or about `0x7f`, in an 8-bit format), but that is not the case. Due to a series of historical circumstances and practical coincidences too long-winded to be worth going into, pixel brightness values are not linear. Instead, they are stored as the brightness value raised to a power γ. The most common default is $\gamma = 0.4545$. When the image is displayed, the pixels are raised to the inverse gamma, 2.2, to obtain the linear brightness value.[64] This is typically done by your monitor. Thus, 50% brightness is actually encoded as 0.73, or `0xba`. PNG images can specify an alternate γ value, and your PNG decoder is responsible for converting it to the correct display gamma.

Like every other optional feature of every other file format, whether this is actually implemented is anyone's guess. As it turns out, most web browsers implement it properly, and most image processing libraries do not. Many websites use these to create thumbnails: Reddit, 4chan, Imgur, Google Docs. We can use this to our advantage.

Take one source image and darken it (map its brightness range to 0%..80%). Take the other source image, and lighten it (map its brightness range to 80%..100%). The two images now occupy distinct portions of the brightness gamut. Now, for every `2x2` group of pixels, take 3 pixels of the darker image and 1 pixel of the lighter image. Finally, encode the result as a PNG and apply the gAMA PNG tag, using an extreme value such as γ=0.0227. (Twenty times lower than the default γ=0.4545.)

[64] Most computers these days use, or at least claim to support, the sRGB colorspace, which doesn't actually use a pure gamma function for a bunch of technical reasons. But it approximates $\gamma = 2.2$, so we're rolling with that.

We can do this easily enough with ImageMagick:

```
$ size=$(convert "$high" -format "%wx%h" info:)
$ convert \( "$low" -alpha off +level 0%,80% \) \
      \( "$high" -alpha off +level 80%,100% \) \
      -size $size pattern:gray25 -composite    \
      -set gamma 0.022727                      \
      -define png:include-chunk=none,gAMA      \
      "$output"
```

When viewed without the specified gamma correction, all of the lighter pixels (25% of the image) approach white and the overall image looks like a washed out version of the darker source image (75% of the image). The 2×2 pixel pattern disappears when the image is downscaled to less than half of its original dimensions (if the scaler is any good anyway). When the gamma correction *is* applied to the original image, however, all the darker pixels are crushed to black, and now the lighter pixels span most of the brightness spectrum, revealing the lighter image as a grid of bright pixels against a black background. If the image is displayed at `1:1` pixel scale, it will look quite clean. Scales between 100% and 50% typically result in moiré artifacts, because most scalers cheat. Scaling down usually darkens the image, because most scalers also don't do gamma-correct scaling.[65]

$\gamma = 0.4545$

$\gamma = 0.0227$

This approach is the one I've seen used so far, and it is easy to achieve using the Levels tool in GIMP, but we can do better. The second image is much too dark: we're mapping the image to a linear brightness range, but then applying a very much non-linear gamma correction. Also, in the first image, we can see a "halo" of the second image, since the information is actually there. We can fix these issues.

Let's use ImageMagick again. First we'll apply a true gamma adjustment to the high source image. The `-gamma` operation in ImageMagick performs an adjustment by the inverse of the supplied value, so to apply an adjustment of $\gamma = 1/20$ we'll pass in 20. We'll also slightly increase its brightness, to ensure that after gamma adjustment the pixels are close enough to white:

```
$ convert "$high" -alpha off +level 3.5%,100% \
                  -gamma 20 high_gamma.png
```

This effectively maps the image range to $0.035^{0.05} = 0.846..1.0$, but with a non-linear gamma curve. Next, because the low image will appear washed out, we'll apply a gamma of 0.8, then darken it to 77% of its original brightness. $0.77^{20} = 0.005$, which is dark enough to not be noticeable. We're keeping this in a variable to chain later.

```
$ low_gamma="-alpha off -gamma 0.8 +level 0%,77%"
```

Now let's compensate for the halo caused by the high image. For every 2x2 output pixels, we'd like an average color of:

$$v = 3/4v_{low} + 1/4$$

That is, as if the high image was completely white. What we actually have is:

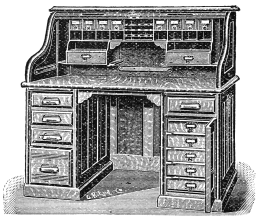
$$v = 3/4v'_{low} + 1/4v_{high}$$

Solving for v'_{low} gives:

$$v'_{low} = v_{low} - 1/3v_{high} + 1/3$$

We can implement this in ImageMagick using `-compose Mathematics`:

```
$ convert \( "$low" $low_gamma \) high_gamma.png \
          -compose Mathematics                   \
          -define compose:args='0,-0.33,1,0.33'  \
          -composite low_adjusted.png
```

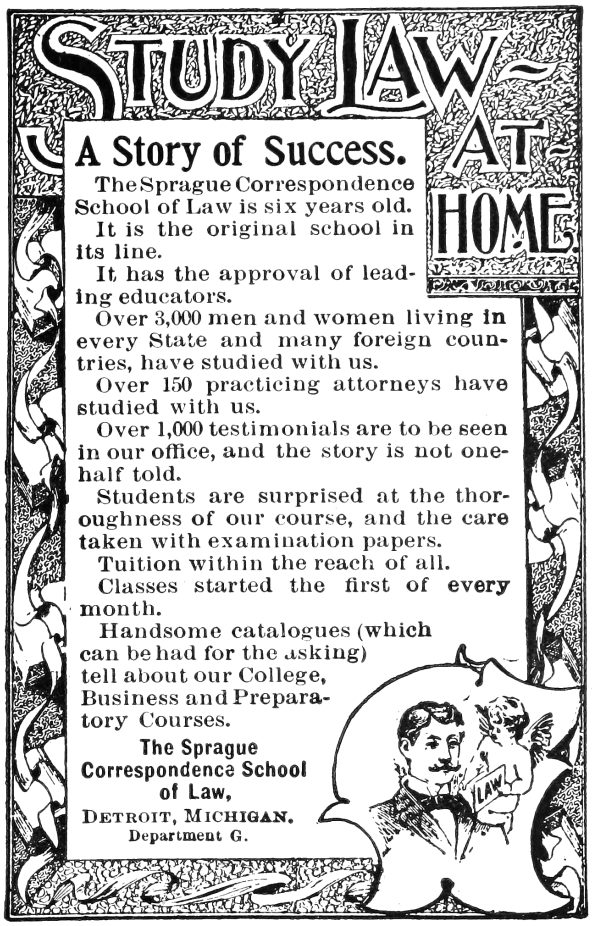


[65] Note that gamma-correct scaling is orthogonal to the gamma trick used here. A simple black-and-white checkerboard *should* be downscaled to a solid 0.73 gray (half the photons, or 50% brightness, at $\gamma = 0.4545$), but most scalers just average it down to 0.5, which is wrong. GIMP is one of the few apps that does gamma-correct scaling these days. Isn't gamma fun?

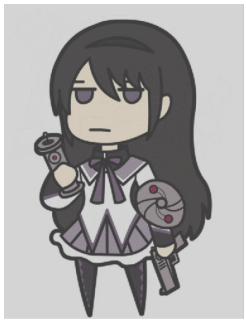
There will be some slight edge effects, due to aliasing issues between the chosen pixels from both images, but this will remove any blatant solid halo areas. This correction assumes that the thumbnail scaler does not perform gamma-correct scaling,[65] which is the common case. This means it is incorrect if the output image is viewed at 1:1 scale (the halo will be visible), but once scaled down it will disappear. In order to cater for gamma-correct scalers (or 1:1 viewing), we'd have to perform the adjustment in a linear colorspace.

Finally, we just compose both images together with a pattern as before:

```
$ convert low_adjusted.png high_gamma.png \
      -size $size pattern:gray25          \
      -composite -set gamma 0.022727      \
      -define png:include-chunk=none,gAMA \
      "$output"
```



The result is much better.



$\gamma = 0.4545$

$\gamma = 0.0227$

The previous images in this article have been filtered (2×2 box blur) to remove the high-frequency pixel pattern, in order to approximate how they would visually appear in a browser context without relying on the specific scaling/resampling behavior of your PDF renderer. In fact, the filtering method varies: gamma-naive for simulating thumbnailing, gamma-aware for simulating the true response at 1:1 scale. For your amusement, here are the raw images. Their appearance will depend on exactly what kind of filtering, scaling, or other processing is applied when the PDF is rasterized. Feel free to play with your zoom setting.

$\gamma = 0.4545$

$\gamma = 0.0227$

Yup, it's 2017 and most software still can't up/downscale images properly. Now don't get me started on the bane that is non-premultiplied alpha, but that's a topic for another day.