Image Correction for RGB LED Matrices

Created by Phillip Burgess

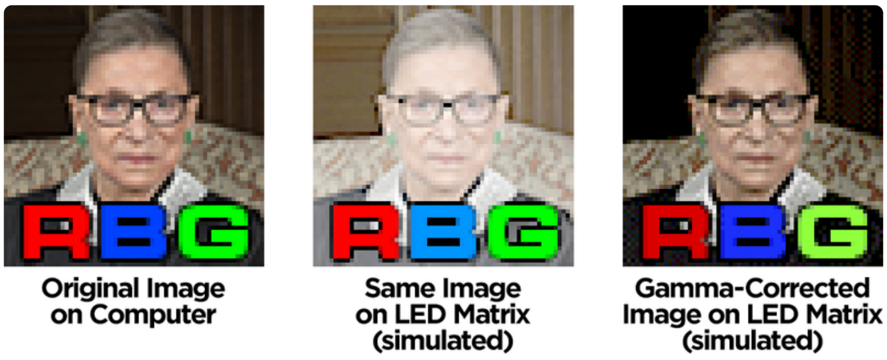
https://learn.adafruit.com/image-correction-for-rgb-led-matrices

Last updated on 2024-06-03 03:14:34 PM EDT

Table of Contents

Overview

Working with **RGB LED matrices** in CircuitPython (or Arduino with the Adafruit_Protomatter library), you might notice that certain graphics appear **washed out**. High-contrast elements such as text or "LED sand" look fine, but something appears off when you try loading a more subtly-shaded BMP image.


RBG
Original Image on Computer
Same Image on LED Matrix (simulated)
Gamma-Corrected Image on LED Matrix (simulated)

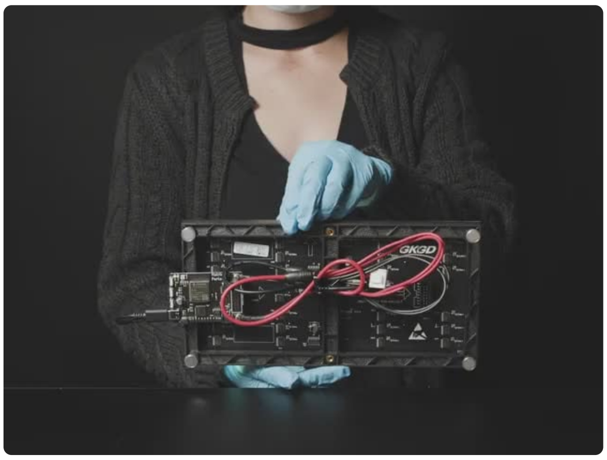
It's not an error in your graphics or the LED matrix, but an artifact of how our eyes have a non-linear response to light. We've written a whole guide about the issue and the fix — **gamma correction** (https://adafru.it/w2B) — and you'll see the topic appear again and again in several other LED-related guides.

We'd like to have gamma correction baked right into our LED matrix driver code. Work continues on that but isn't ready yet…and once there, it will demand extra RAM and CPU cycles that not all projects can afford.

In the interim, we've come up with a couple techniques for preprocessing images to look better on these matrices, as seen in the graphically-intensive Moon Clock (https://adafru.it/NB7) and Matrix Portal Eyes (https://adafru.it/NVc) projects. Not as desirable as true gamma correction in the matrix driver, but often a good compromise!

Both approaches require using a “regular” desktop or laptop computer — one uses a **Python** script, the other relies on the free **ImageMagick** graphics utility.

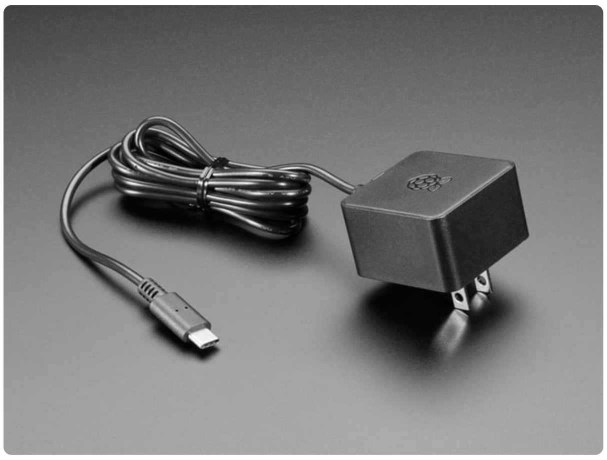
Parts



Adafruit Matrix Portal - CircuitPython Powered Internet Display

Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...

https://www.adafruit.com/product/4745



Official Raspberry Pi Power Supply 5.1V 3A with USB C

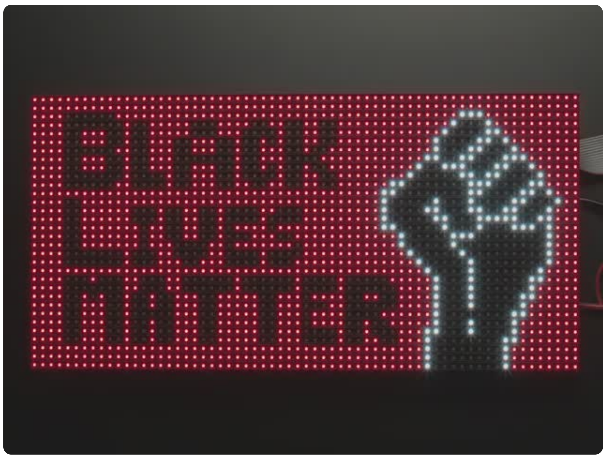
The official Raspberry Pi USB-C power supply is here! And of course, we have 'em in classic Adafruit black! Superfast with just the right amount of cable length to get your Pi 4...

https://www.adafruit.com/product/4298

64x32 RGB LED Matrix - 3mm pitch

Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...

https://www.adafruit.com/product/2279



64x32 RGB LED Matrix - 4mm pitch
Bring a little bit of Times Square into your home with this sweet 64 x 32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them...
https://www.adafruit.com/product/2278

64x32 RGB LED Matrix - 5mm pitch
Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...
https://www.adafruit.com/product/2277

64x32 RGB LED Matrix - 6mm pitch
Bring a little bit of Times Square into your home with this sweet 64x32 square RGB LED matrix panel. These panels are normally used to make video walls, here in New York we see them on...
https://www.adafruit.com/product/2276

64x32 Flexible RGB LED Matrix - 4mm Pitch
If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB, so it's flexible?" and the...
https://www.adafruit.com/product/3826

[64x32 Flexible RGB LED Matrix - 5mm Pitch](https://www.adafruit.com/product/3803)
If you've played with multiplexed RGB matrices, you may have wondered "hey, could we possibly manufacture these on a thin enough PCB so it's flexible?" and the answer...
https://www.adafruit.com/product/3803

Still Images Using Python

Our first method handles **still images** in most popular formats — PNG, JPG and so forth — and outputs a **BMP** image suitable for CircuitPython use. **BMP is frequently used in microcontroller environments because it's easy to decode.**

This approach requires:

- A desktop or laptop computer
- Some familiarity with command line usage
- Python 2 or 3 and the PIL or Pillow library

This method does not crop or resize images for the matrix! The source material must match the final size you intend to use. Sometimes you want images smaller or larger than the matrix, to scroll and move around. That's fine, and this method will handle it.

Installation

From the command line, make sure you have the Python Imaging Library (PIL) or Pillow installed. For existing Python users, you'd type:

```
pip install pillow
```

This will either download and install the imaging library, or you might get a "Requirement already satisfied" message, which is fine.

Then download our image conversion script... you can use wget or curl from the command line if one of those is installed on your system. Either way, this should be typed (or copied-and-pasted) as a single unbroken line, regardless how it may appear formatted in your web browser right now):

```
wget https://raw.githubusercontent.com/adafruit/Adafruit_Media_Converters/master/
protomatter_dither.py
```

or

```
curl https://raw.githubusercontent.com/adafruit/Adafruit_Media_Converters/master/
protomatter_dither.py &gt; protomatter_dither.py
```

or you can download it from your browser here:

Download protomatter_dither.py from Github

https://adafru.it/Oa4

Use

It's easiest if the conversion script and your images are in the **same directory**. If not... you should "cd" to the **directory containing the images** and type the relative path to the script, not the other way around. The script isn't terribly robust with handling relative paths.

But first: **what BMP variant** will your project require? In CircuitPython, `displayio.OnDiskBitmap` prefers full-color **24-bit** RGB images, while `adafruit_imageload.load` works with **8-bit** "palettized" images. Look at your project's **source code** for insights. The Matrix Creature Eyes (https://adafru.it/NVc) project, for example, uses 8-bit palette images.

The **protomatter_dither.py** script accepts a list of input file names. BUT...if you want 8-bit palette BMP output, the list should be preceded by an "**8**".

Let's suppose you have three images to convert, a mix of PNG and JPG files called foo.png, bar.jpg and baz.png.

If you want 24-bit full color output, invoke the script like this:

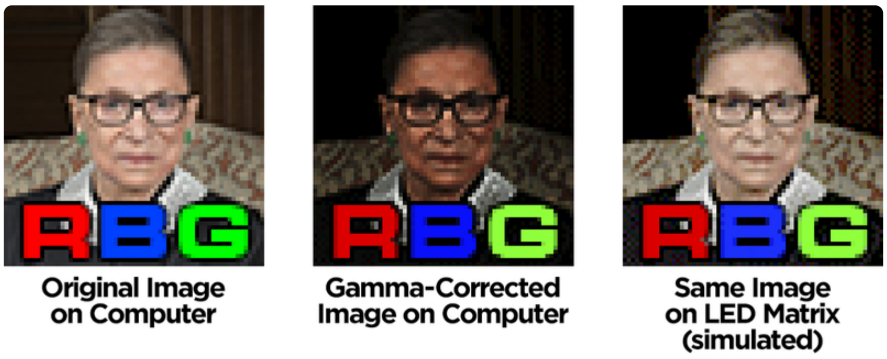
```
python protomatter_dither.py foo.png bar.jpg baz.png
```

For 8-bit palette output, insert an "8" before the file list, like so:

```
python protomatter_dither.py 8 foo.png bar.jpg baz.png
```

The script will output a **BMP** file for each input file, with "-processed" added to the filename — for example, "**foo.png**" will result in an output file called "**foo-processed.bmp**". The original file remains there, unharmed. You can then rename the output file to whatever you want and copy it to the **CIRCUITPY** drive of your microcontroller board loaded with CircuitPython firmware..

The BMPs will look very dark on your computer...but they'll appear normal on the matrix! The script has to compensate for the lack of gamma correction there.


RBG
Original Image on Computer
RBG
Gamma-Corrected Image on Computer
RBG
Same Image on LED Matrix (simulated)

Occasionally these BMPs will need some manual touch-ups. The **Matrix Eyes** project, for example, relies on **one very specific color** being transparent...but, because the Python script applies dithering, occasionally it leaves a few dots in what should be the transparent areas.

You'll need an **image editor** that can load and save **BMP** images (such as Photoshop, but there are many others, some free) ... and, importantly, you must save the BMP as the exact same variant that you loaded, 8- or 24-bit, uncompressed. Some programs might change these details, so make sure you're getting the right thing before committing to a big project.

Animated GIFs using ImageMagick

One of the example projects for the **Adafruit_Protomatter** library (**Arduino** environment) is an **animated GIF player**. The previously-mentioned Python script doesn't handle animated GIFs, but there's another tool — **ImageMagick** — that may be able to help.

ImageMagick is a free command line tool for image conversion, available for most popular systems. The download page for ImageMagick (https://adafru.it/FCK) explains a few different ways to access the software. We'll assume here you have ImageMagick installed and working on your system.

Sometimes the different ports of ImageMagick aren't in perfect sync, and commands that work in one environment aren't available in another. The steps we'll go through here worked on a Mac but might not work the same on Windows. If you encounter incompatibility, check that you're running the latest available for your platform, or consider a nearby Linux system (e.g. Raspberry Pi) if you have access to one.

If Image Already Matches Matrix Pixel Dimensions...

...it's fairly straightforward, ImageMagick's "gamma" command will pull down the brightness:

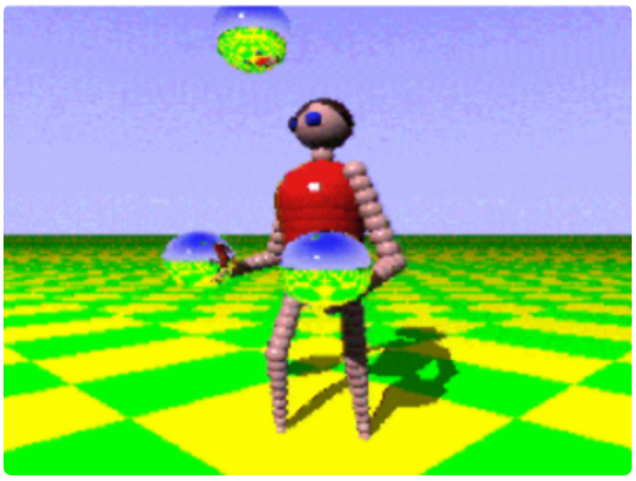
```
convert in.gif -coalesce -gamma 0.4 -dispose None -interlace None -ordered-dither
o4x4,32,64,32 -layers OptimizeFrame out.gif
```

You can adjust the `0.4` down (darker) or up (lighter) to get things dialed in just right for the matrix. Usually 0.4 works fine though.

The "-ordered-dither" part is optional. This adds dithering to recover some of the detail otherwise lost during the gamma adjustment.

The coalesce/dispose/interlace settings are to ensure better compatibility with the AnimatedGIF library. It doesn't support interlaced GIFs, a reasonable tradeoff to allow handling arbitrarily long or complex animations.

If Image Does Not Match Matrix Size...



Let's suppose we had an animated GIF of the classic Amiga juggler demo, and we want to prepare this for the RGB matrix.

The GIF, at 320x200 pixels, is much larger than the matrix. And the 16:10 aspect ratio doesn't match.

One option is to **scale the whole image** to fit the matrix, disregarding the pixel aspect ratio. If there's a significant change to the aspect ratio, distortion will be apparent, but the full frame will be used.

For a 64x32 pixel matrix:

```
convert juggler.gif -coalesce -gamma 0.4 -resize 64x32\! -dispose None -interlace
None -ordered-dither o4x4,32,64,32 -layers OptimizeFrame out.gif
```

This should be entered as a single unbroken line, regardless how it may be formatted in the web browser. Change the input and output filenames to suit your needs.

The reference to `64x32` is the matrix pixel dimensions…change this if using a larger or smaller matrix. The `\!` tells ImageMagick to scale/stretch to this size absolutely, regardless of distortion it may incur. The `o4x4,32,64,32` is not related to the matrix size…keep that part intact!

To maintain the original **pixel aspect ratio** instead, cropping the image as necessary:

```
convert juggler.gif -coalesce -gamma 0.4 -resize 64x32^ -gravity center -extent
64x32 -dispose None -interlace None -ordered-dither o4x4,32,64,32 -layers
OptimizeFrame out.gif
```

Now there's two `64x32` references to the matrix pixel dimensions…change **both** for different matrix sizes. The `^` on the first one means we'll be **cropping** to this size, not distorting. Do not change the `o4x4,32,64,32`, regardless of matrix size.

The resulting GIFs will look dark on your computer (left). But played back on the LED matrix (simulated on the right), it will more closely match the original's colors and brightness: