Preparing Graphics for E-Ink Displays

Created by Phillip Burgess

https://learn.adafruit.com/preparing-graphics-for-e-ink-displays

Last updated on 2024-03-08 03:33:13 PM EST

Table of Contents

Overview 3

- Image Formats

Convert With ImageMagick 5

- Setting Up
- Working the Magic
- Special Cases
- Rotated Images

Convert With Photoshop 9

- Special Cases
- Rotated Images

Overview

What's old is new again.

Back in the early Macintosh computer days, when the Earth's crust was still cooling, getting photos and artwork to look presentable on these computers' 1-bit monochrome displays was something of an art form.

Much of that know-how has fallen by the wayside as screens improved...but with wafer-thin, ultra-low-power **E-Ink displays** appearing everywhere now, a refresher in those techniques is suddenly relevant again.

A couple of software options are shown in this guide. **ImageMagick** (a free command-line tool) and **Photoshop** (not free, but many folks already have it).

The resulting BMP images can be used with **Arduino** or **CircuitPython** libraries for these displays.

For **Arduino**, that would be the **Adafruit_ImageReader** library (https://adafru.it/OfJ), which can be installed with the Arduino Library Manager (Sketch→Include Library→Manage Libraries...). The latest versions of the Arduino IDE will then take care of installing all the dependent libraries (Adafruit_GFX and others). See the **EInk*** examples included with the Adafruit_ImageReader library for usage.

For **CircuitPython**, here's a whole guide to get you started (https://adafru.it/OfK).

Image Formats

The aforementioned libraries — both for Arduino and CircuitPython — read **BMP** image files. But even within BMP, there are a number of different variations...and these libraries, evolving independently, aren't always in step. Knowing what each library can handle is important when converting images.

- **Adafruit_ImageReader for Arduino:** reads 1-bit and 24-bit uncompressed BMPs only.
- **CircuitPython displayio.OnDiskBitmap:** reads uncompressed BMPs of any depth, 1- to 32-bit.

- **CircuitPython adafruit_imageload:** 1-, 4- and 8-bit uncompressed or RLE-compressed BMPs. 16-bit and higher depths are not supported.

Because E-ink displays typically offer just a few shades, 1- or 4-bit output would normally be ideal. But due to the library constraints above, sometimes it's necessary to use 24-bit output regardless, which we'll explain...

Convert With ImageMagick

Setting Up

ImageMagick (https://adafru.it/FCJ) is a command-line tool for image conversion and processing. The software feels a bit anachronistic because it has no GUI...but to its credit it does a fantastic job, is available for Windows, Mac and Linux, and is entirely free so everyone has a shot at using it.

Installing ImageMagic varies by platform and is beyond the scope of this guide, so head over to the ImageMagick download page **(https://adafru.it/FCK)** for guidance on setting up the software.

Regardless of platform or installation procedure, in the end you will be working from the **command line** and typing things in...that's just how it works. This means you'll likely be using the Command Prompt in Windows, Terminal in macOS or Linux.

After ImageMagick is installed, **download these PNG images** to your computer and stash them away for safe keeping. You'll need them later:

Working the Magic

Source images can be most any format. We'll ask ImageMagic to output everything as **.BMP**s, because that's what **CircuitPython** works with...it's a relatively easy image format for microcontrollers to handle.

We'll assume your source image(s) are **already sized for the e-ink display** (check the specific product page for your display's dimensions, there's a variety). ImageMagick does have scale and crop functions, but that's a whole added layer of complexity... you'll probably have an easier time using GUI tools, most operating systems have something basic included. But if you really need it, refer to the ImageMagic documentation (https://adafru.it/FCM) for image resizing.

Put the image(s) you want to convert and the eink-*.png images in the same directory, then "cd" to that directory. (These could be in separate directories, but you'll need to specify relative paths in that case...we're keeping this example simple for command line novices.)

Supposing your source image is called "input.jpg," and you want to create "output.bmp," you'd type:

```
convert input.jpg -dither FloydSteinberg -define dither:diffusion-amount=85% -remap
eink-3color.png BMP3:output.bmp
```

If using a monochrome (black & white) e-ink display, substitute `eink-2color.png` for the `-remap` argument.

For a 4-level grayscale e-ink display, use `eink-4gray.png` instead.

If your target application uses the **Adafruit_ImageReader library for Arduino** (which only handles 1- or 24-bit BMPs), you'll need to insert "-type truecolor" just before the output filename, like so:

```
convert input.jpg -dither FloydSteinberg -define dither:diffusion-amount=85% -remap
eink-3color.png -type truecolor BMP3:output.bmp
```

You do not need or want the truecolor option for **CircuitPython**. But...depending on the route you're taking there...**displayio.OnDiskBitmap** vs. the **adafruit_imageload** library...the latter can handle run-length encoding compression, which saves a modest amount of disk space:

```
convert input.jpg -dither FloydSteinberg -define dither:diffusion-amount=85% -remap
eink-3color.png -compress RLE BMP3:output.bmp
```

If you want something that works with either CircuitPython approach, skip the RLE compression. But I digress...let's focus on the output...

Diffusion dithering used here gives us the most bang-for-buck in most cases.

Try different `dither:diffusion-amount=X%` settings until you find the right compromise between "too contrasty" (lower values) and "too snowy" (higher values). **This is very subjective and the ideal setting will vary from image to image!** Also, this setting is only available in recent releases of ImageMagick...older versions only work at 100%. If you have an old version 6 installation, this might be the time to upgrade.

Let's see how different settings affect 2-color (left), 3-color (center) and 4-level gray (right) images:

Special Cases

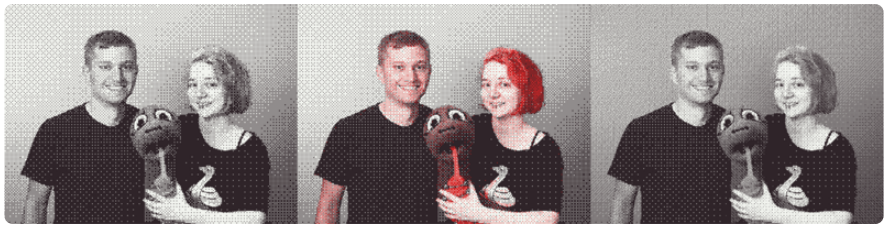
A couple of alternate dither settings might be useful in certain situations...

Ordered dithering uses a structured pattern. It's usually not the best for photos, as it tends to lose edge details, but may provide a "clean" look for flat artwork and diagrams:

```
convert input.jpg -ordered-dither o8x8 -remap eink-3color.png BMP3:output.bmp
```

In place of "o8x8", you can try "o4x4" and "o2x2" for different (sometimes cleaner) results.

Remember to insert the "-type truecolor" or "-compress RLE" settings if (and only if) the situation demands it, as explained earlier.



You can also disable dithering altogether, which may be useful for text, high-contrast line art and 1980s Patrick Nagel prints:

```
convert input.jpg -dither None -remap eink-3color.png BMP3:output.bmp
```

While ImageMagick does have other dithering options, they tend to look best for ultra-high-DPI printed matter and are not well suited for these displays. We'll not cover them here.

Rotated Images

Some displays operate natively in a "portrait" orientation (tall vs wide). If your source image is in the opposite orientation, you have two choices...

1. In your CircuitPython code, instruct the display to use a different rotation setting, e.g.:

```
display.rotation = 1
```

(try 0, 1, 2 or 3)

2. Alternately, leave the code unchanged and rotate the image using ImageMagick, in which case you would insert the following **before** the output filename:

```
-rotate 90
```

Use **90** to rotate the image **clockwise**, or **-90** to rotate **counterclockwise**.

Convert With Photoshop

Adobe Photoshop is not inexpensive. But if you're fortunate enough to already have it, this does make conversion a bit simpler.

To begin, download and unzip these Photoshop color tables:

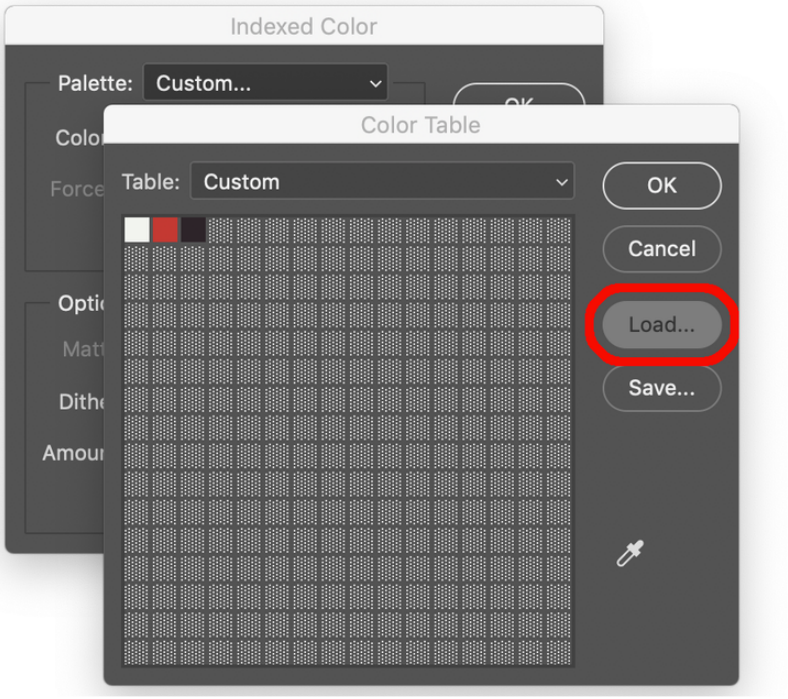
eink-palettes.zip

https://adafru.it/OfL

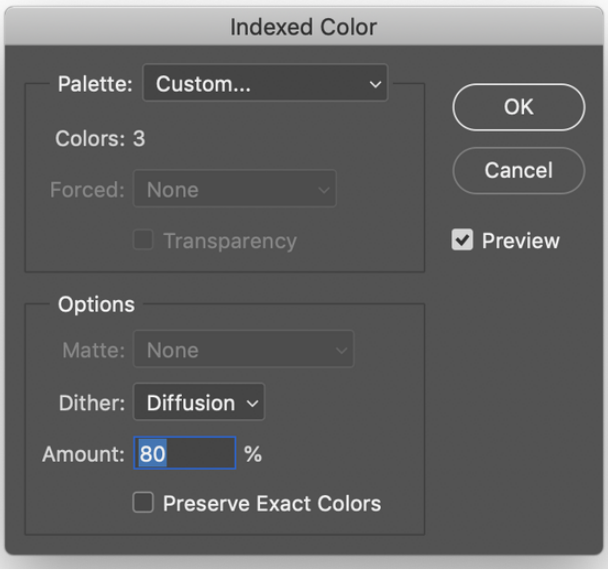
Your source image should be in **RGB mode** (Image→Mode→RGB) if it isn't already. Use the **Image Size** and/or **crop** tools to **match the dimensions of your image(s) to your e-ink display**.

Then we'll do the e-ink conversion using **Image→Mode→Indexed Color...**

From the "**Palette:**" menu, select "**Custom...**" then click the "**Load...**" button to import one of the color tables from the ZIP you downloaded above. Use "eink-3color" to process images for 3-color displays, "eink-2color" for monochrome (black & white) displays, and "eink-4gray" for 4-level grayscale displays.



Then experiment with settings in the **Options** pane to get the effect you desire (keep the "Preview" box checked to see the results interactively).



Diffusion dithering used here gives us the most bang-for-buck in most cases.

Try different "**Amount**" settings until you find the right compromise between "too contrasty" (lower values) and "too snowy" (higher values). **This is very subjective and the ideal setting will vary from image to image!**

Let's see how different settings affect 2-color (left), 3-color (center) and 4-level gray (right) images:

Special Cases

A couple of alternate dither settings might be useful in certain situations...

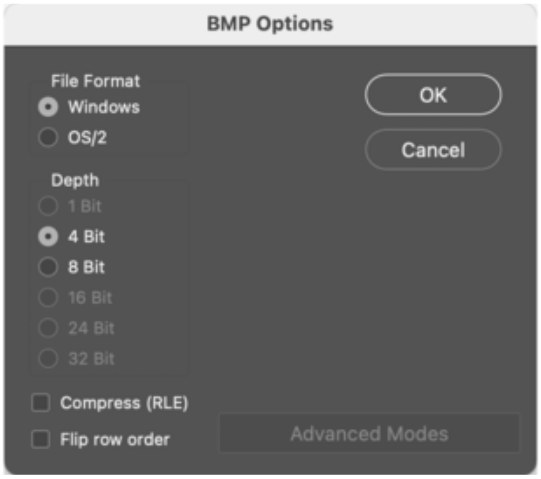
Pattern dithering uses a uniform dot arrangement. It's usually not the best for photos, as it tends to lose edge details, but may provide a "clean" look for flat artwork and diagrams:

Unlike ImageMagick (explained on the prior page), Photoshop just has one pattern dither setting, there are no available tweaks.

The "**None**" dither setting may be useful for text, high-contrast line art and 1980s Patrick Nagel prints:

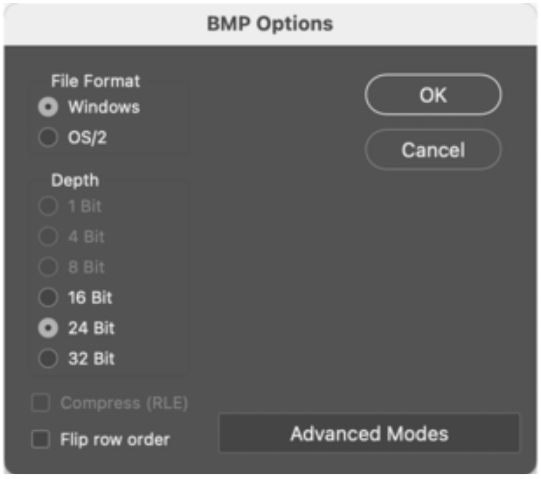
Photoshop also has a “Noise” dither setting — it looks okay for ultra-high-DPI printed matter but is not well suited for these displays.

Once you have a satisfactory conversion, we’ll **export** in **BMP** format…it’s a relatively easy image format for microcontrollers to handle. As explained on the overview page, you have some options here depending on what code you’ll be using to display the image…



If your target system is **CircuitPython**, select **File→Save As…** then choose the “**4 Bit**” option here.

And if your CircuitPython code will be using the **adafruit_imageload** library (rather than **displayio.OnDiskBitmap**), you can check the “**Compress (RLE)**” box to save a little bit of space. If you’re not sure, or might be using both libraries in different situations, leave this **unchecked**.



And for **Arduino** with the **Adafruit_ImageReader** library, first change the image’s color mode back to RGB (**Image→Mode→RGB Color**…don’t simply Undo, we need the dithered version but in RGB mode, not the original image), then select **File→Save As…**

Choose the “**24 Bit**” option and save.

Rotated Images

Some displays operate natively in a “portrait” orientation (tall vs wide). If your source image is in the opposite orientation, you have two choices…

1. In your CircuitPython code, instruct the display to use a different rotation setting, e.g.:

```
display.rotation = 1
```

(try 0, 1, 2 or 3)

2. Alternately, leave the code unchanged and rotate the image using Photoshop:

Image→Image Rotation→90° Clockwise or

Image→Image Rotation→90° Counter Clockwise

Then save as a BMP as described above.