1. Developing RAW Images

1.1 RAW Image Conversion

The following are the results of the reconnaissance run of dcraw:

- black=0
- white=16383
- r_scale=1.628906
- g_scale=1.000000
- b_scale=1.386719

I also saved using the TIFF file with the new flags

1.2 Python Initials

I got the information about the numpy arrays for the image:

- Width: 4284
- Height: 2844
- Bits per Pixel: 16

1.3 Linearization

I linearized the image by rescaling between the black and white values in the reconnaissance run, and clipped the result. See Code

1.4 Bayer Pattern Identification

I created a function to identify the bayer pattern.

I found that the easiest way to identify the correct Bayer pattern is to look for similarity across each pixel in the four-pixel neighborhood at the top left corner.

I assume that the same/similar wavelengths of light are spreading across the 2x2 pixel neighborhood due to dispersion/the circle of confusion.

I assume that sensors of different colors have different responses to different wavelengths of light; as such, the diagonal with the two most similar sensor responses would represent the same color, and thus green.

I also assume, based on the spectrum response curves viewed in class, that the camera sensor responses of different colors have some degree of overlap at certain wavelengths. Given that the green and blue response curves overlap on a greater range of wavelengths than green and red, I conclude that the pixel most similar to the mean of the diagonal must be blue, and thus the remaining pixel in the 2x2 neighborhood must be red.

Given this, the code I provided determines the bayer pattern, and we can index into the image at different indices to obtain the separate channels.

Note that I wanted the code to generalize to different bayer patterns, so later functions accept the bayer pattern and index appropriately for any given pattern.

1.5 White Balancing

I implemented the different white balancing algorithms. There were a few questions regarding the implementation that I couldn't clear up in OH, so I just implemented each, decided which to actually use, and commented out the others.

For white world, I found the brightest "pixel" (or rather, neighborhood of four pixels that will combine to form the brightest white pixel in the final image) - this part was a little unclear, but this seemed to be the most logical way of finding the white pixel in the resulting image (as taking the max of each channel would not be selecting a "white" pixel, but could rather be coming from multiple pixels at different locations; I implemented this as well, as I wasn't completely sure, but the results are not pictured)

For the gray world, I rebalanced them so the means are equal (e.g. resulting in a gray). However, one could also interpret this as needing to balance out to be gray with brightness of 0.5, but that ends up the same anyway with the brightness correction.

For the presets, I just used those to scale

Of the three automatic white balancing algorithms, I found that I liked the preset values for the camera the best.

Note: I also implemented a post-demosaic white balancing function because that was the order we discussed in class, and that is much cleaner code and speeds up the pipeline; however that needs to be uncommented to make it run as the assignment does white balancing first.

Results of white world assumption:

Results of gray world assumption:

Results of camera presets:

1.6 Demosaicing

I demosaiced the image according to the bayer pattern; I indexed into the original image based on the bayer pattern, and put each into its own empty array that matched the size of the original image and interpolated. See code.

Note: I couldn't get the interp2d or interpolate to work, so I had to implement bilinear interpolation manually in order to progress; the results seemed to be pretty good, though. If this is an issue, let me know, but it just was extra work on my part.

1.7 Color Space Correction

I performed color correction as specified in the writeup. See Code.

1.8 Brightness Adjustment
I applied the linear brightness adjustment as specified in the writeup, followed by gamma correction.

For the linear scaling, I tested a lot of variations, and 55% looked pretty good to me overall.

1.9 Compression

I tested multiple different compression qualities for the resulting image. For the version specified in the writeup, I couldn't tell the difference; the compression ratio for jpg at quality=95 is 16768/2746, or ~6.1

For all practical purposes, quality of around 50 was almost identical to the original for me (even 40 was almost indistinguishable); perhaps the blush of the baby's face is ever so slightly less sharp, but after swapping back and forth at full screen for several minutes, I can't see any real difference. Compression ratio of 16768/441, or ~38.0

For reference, this has quality=40

1.10 Manual White Balancing

I tested the patches around the following coordinates, selected by displaying the window and viewing the coordinate of the cursor.

- Window Glare=[190, 3279]

- Window Glare=[420, 3658]

- Baby Eye=[1082, 2442]

- Baby Eyebrow (white part)=[937, 2559]

- Baby Eye Glint=[1035, 1833]

1.11 Learn to use DCRAW

I used the following flags:

-4 linearizes
-w white balances based on camera settings
-q 0 sets demosaic algorithm to bilinear
-H 0 sets the highlight behavior to clip, as we use that in our approach
-o 1 converts to sRGB colorspace (including identifying bayer pattern and demosaicing)
-g 2.4 12.9 sets to sRGB gamma slope

Combine for the command:

dcraw -4 -w -q 0 -H 0 -o 1 -g 2.4 12.9 baby.nef

You'll need to rename it, as international characters appear in place of the .ppm extension

magick convert baby.ppm baby.png

It’s darker than I would like, but is about on-par with the baby.jpeg given with the assignment

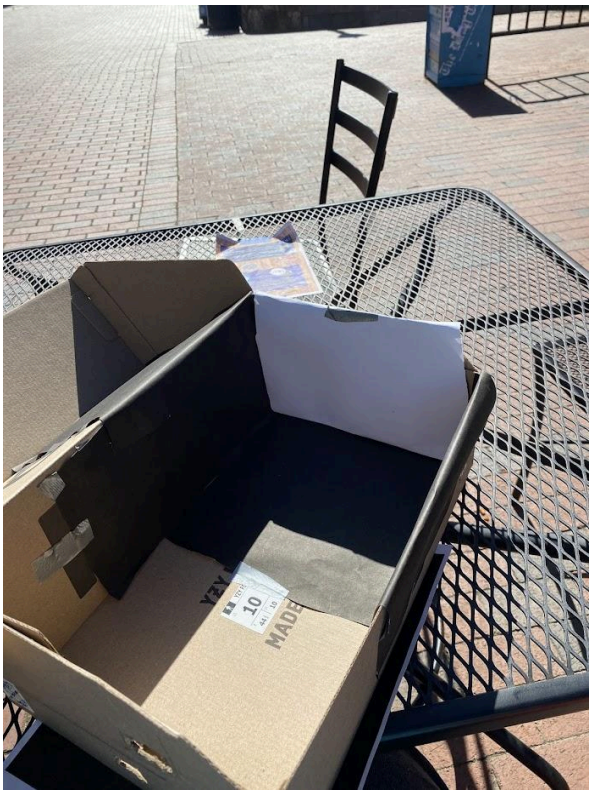
2. Pinhole Camera

2.1 Pinhole Camera Construction

I constructed two variants of the camera obscura using two shoeboxes (one going along the shorter axis, and the other along the longer axis).

I ended up getting better results with the longer axis one, so here are the specs:

Focal Length: 35 cm
Screen size: Full size of the side of the shoebox (21x13 cm)
Pinholes: I tested numerous different sizes, and found that 3 mm, 5 mm, and 10 mm worked best.



2.2 Using the Pinhole Camera

I tried using the formula, however I struggled to get good results, so I just tested a lot of them; eventually I settled on 3mm, 5mm, and 10mm.

I found that 10mm gave a blurry, but colorful and bright image. 5mm gave a sharper, but moderately dimmer image. And 3 mm yielded an image that was almost too dark to see (even

after 30 seconds of exposure time), but was definitely producing an image because I could see shapes moving across the back screen when I moved my hands in front of them.

There are eight example scenes, each of which is ordered 10mm, 5mm, 3mm