Assignment 1: Computational Photography

Neeraj Panse
Andrew ID: npanse

September 16, 2023

1 Developing RAW Images

1.1 Implement a basic image processing pipeline

Raw Image Conversion:

Using the following command:

`dcraw -4 -d -v -w -T <RAW_filename>`

Black level: 150

White level: 4095

r_scale: 2.394531

g_scale: 1.00000

b_scale: 1.597656

Python Initials:

Bits per pixel: 16

Height: 4016

Width: 6016

Linearization:

Code for linearization:

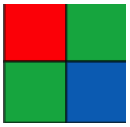
```
data = skimage.io.imread('data/campus.tiff')

data = data.astype('double')
height, width = data.shape

data = (data - black_thresh)/(window_thresh - black_thresh)
data[data < 0] = 0
data[data > 1] = 1
```

Identifying the correct Bayer pattern

Through empirical results, the 'RGGB' Bayer Pattern works perfectly well for this case.



Code snippet for applying bayer pattern

```
def bayer_pattern(image):

    masks = dict((color, np.zeros(image.shape))for color in ['Red', 'Green', 'Blue'])
    masks['Red'][0::2, 0::2] = 1
    masks['Green'][0::2, 1::2] = 1
    masks['Green'][1::2, 0::2] = 1
    masks['Blue'][1::2, 1::2] = 1


    for color in ['Red', 'Green', 'Blue']:
        masks[color] = masks[color].astype(bool)

    R = image * masks['Red']
    G = image * masks['Green']
    B = image * masks['Blue']

    return [np.stack([R, G, B],axis=2), masks]
```

White balancing

Code snippet to apply white balance:

```
def white_balance(data, type):
    """
    0 -> Grey
    1 -> White
    2 -> Custom RGB
    """

    R, G, B = data[:,:,0], data[:,:,1], data[:,:,2]


    if type == 0:
        r_val = R[R != 0].mean()
        g_val = G[G != 0].mean()
        b_val = B[B != 0].mean()
    if type == 1:
        r_val = R.max()
        g_val = G.max()
        b_val = B.max()
    if type == 2:
        R = R * r_scale
        B = B * b_scale
        G = G * g_scale
    else:
        R = R*(g_val/r_val)
        B = B*(g_val/b_val)

    return (R + G + B)
```

Refer to the following image for Grey World Assumption White Balance: ‘data/white_bal_grey.png‘

Refer to the following image for White World Assumption White Balance: ‘data/white_bal_white.png‘

Refer to the following image for Custom RGB Scale White Balance: ‘data/white_bal_custom.png‘

The best white balance algorithm is using the gray world assumption in this case. As shown by the results, the gray world assumption provides accurate details of the scene. Thus, all further results are calculated using the grayscale world assumption algorithm for white balance.

Demosaicing

Code snippet for demosaicing

```
def demosiac(image):

    height, width = image.shape

    x = np.arange(0, width, 2)
    y = np.arange(0, height, 2)
    X,Y = np.meshgrid(x,y)
    R_interp = interp2d(x, y, image[Y,X], kind='linear')

    x = np.arange(1, width, 2)
    y = np.arange(1, height, 2)
    X,Y = np.meshgrid(x,y)
    B_interp = interp2d(x, y, image[Y,X], kind='linear')

    x = np.arange(1, width, 2)
    y = np.arange(0, height, 2)
    X,Y = np.meshgrid(x,y)
    G_interp_1 = interp2d(x, y, image[Y,X], kind='linear')

    x = np.arange(0, width, 2)
    y = np.arange(1, height, 2)
    X,Y = np.meshgrid(x,y)
    G_interp_2 = interp2d(x, y, image[Y,X], kind='linear')

    x = np.arange(0, width)
    y = np.arange(0, height)

    R = R_interp(x, y)
    G = (G_interp_1(x, y)  +  G_interp_2(x, y))/2
    B = B_interp(x, y)

    return np.dstack([R, G, B])
```

Refer to the following image for the Demosiaced output image: ‘data/demos.png‘

Color space correction

Code snippet for color correction

```
# Color Correction
def color_correction(img, ccm):
    return np.dot(img, ccm.T)

M_xyz_cam = np.array([[6988,-1384,-714],[-5631,13410,2447],[-1485,2204,7318]]) /10000.0

M_rgb_xyz = np.array([[0.4124564, 0.3575761, 0.1804375],
[0.2126729, 0.7151522, 0.0721750],
[0.0193339, 0.1191920, 0.9503041]])

M_srgb_xyz = np.matmul(M_xyz_cam, M_rgb_xyz)

M_srgb_xyz = M_srgb_xyz / M_srgb_xyz.sum(axis=1, keepdims=True)

M_srgb_xyz_inv = np.linalg.inv(M_srgb_xyz)

linear_out = color_correction(out, M_srgb_xyz_inv)

linear_out = np.clip(linear_out,0.,1.)
```

Refer to the following image for the color space corrected output image: ‘data/color_correction.png‘

Brightness adjustment and gamma encoding

The scale at which greyscale brightness adjustment works well is **0.25** or **25%**.

Refer to the following image for brightness adjustment and gamma encoding output: 'data/gamma_out.png'

Compression

The main difference between the **.PNG** and **.JPEG** is the file size. The **.PNG** image is significantly larger in size than the **.JPEG**. The compression ratio here is: **5.7617**

I tried the following values for 'quality' while saving the JPEG file: [95, 75, 65, 40, 20, 10, 5]. The output deteriorated only at the quality level 10. Hence, the quality of the image was maintained until 20% compression and stared decreasing at about 10%.

Refer to the following image for compressed JPEG images : 'data/compressed_n.jpeg' where n = [95, 75, 65, 40, 20, 10, 5]

Final Processed Image

As suggested in the assignment, the final processed image is calculated for two types of white balances, RGB scaled white balance and grey world asumption white balance.

Following are the paths for the above:

GreyWorld Assumption output image: 'data/processed_image_grey_world.png'

RGB Scaled White Balance output image: 'data/processed_image_grey_rgb_scale.png'

1.2 Manual White Balance

Code for manual white balance:

```
def white_balance_patch(data, patch_x, patch_y):
    x1, x2 = patch_x
    y1, y2 = patch_y

    patch = data[y1:y2, x1:x2]
    R, G, B = patch[:,:,0], patch[:,:,1], patch[:,:,2]

    print("KK", patch.shape)
    r_val = R[R != 0].mean()
    g_val = G[G != 0].mean()
    b_val = B[B != 0].mean()


    print(r_val, g_val, b_val)
    R = data[:,:,0]/r_val
    G = data[:,:,1]/g_val
    B = data[:,:,2]/b_val

    return (R + G + B)
```

Following are the results for the manual white balance. The selected patch is drawn in purple.. When we select a patch that is completely white, we get a good image. However, when we select a patch that is greyish in color, the brightness of the generated image is high.

Correct patch selection

Wrong patch selection

1.3 Learn to use dcraw

```
// To process the raw image
dcraw -a -r 2 1 1.5 1 -o 1 -q 1 -f data/campus.nef

// To convert to png format
magick data/campus.ppm data/campus.png
```

Refer to the following image for output of the above commands : ‘data/campus.png‘

Amongst the three images, the one using the above dcraw command, the one given to us already and the one processed using python, I consider the one by dcraw to be the best. Although all three images do capture the scene in detail, the specific areas where the brightness of the image is too high, for eg. the sky and the walls on the stairs, the image generated by dcraw is able to capture the most accurate information of the sky and the clouds. Hence, the dcraw based image is the best according to me.

2 Camera Obscura

2.1 Build the pinhole camera

The above images can be found at the following path: ‘data/setup_n .png‘ where n = [1,2,3]

2.2 Use your pinhole camera

Details regarding the captures:

Screen Size: 285 mm x 217 mm

Approximate Focal Lenght(f) = 350 mm

Pinhole diameters : 5 mm, 3 mm, 2 mm

General Observations:

As the diameter of the pinhole decreases the brightness of the captured image increases as more light is able to pass through. As the diameter increases, the sharpness of the image goes on decreasing and images become blurry. Larger diameter results in reduced depth of field. More light is let in, which can be beneficial in low-light situations, but it also reduces the depth of field. This results in objects at different distances not being focused at the same time and thus results in a significant part of the image being blurry.

Scene 1:

Pinhole diameter: 5 mm

Pinhole diameter: 3 mm

Pinhole diameter: 2 mm

Scene 2:

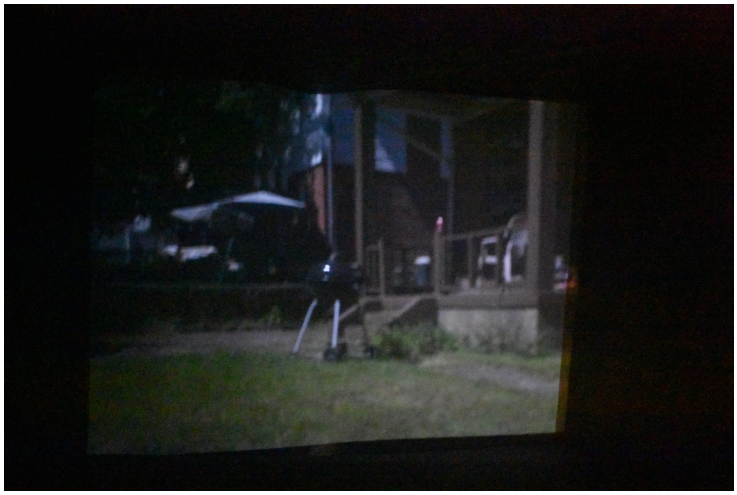
Pinhole diameter: 5 mm

Pinhole diameter: 3 mm

Pinhole diameter: 2 mm

Scene 3:

Pinhole diameter: 5 mm



Pinhole diameter: 3 mm

Pinhole diameter: 2 mm

The above images can be found in their original resolutions at the following path:

Scene 1:

5 mm: 'data/pinhole/1_5mm.jpg‘

3 mm: 'data/pinhole/1_3mm.jpg‘

2 mm: 'data/pinhole/1_2mm.jpg‘

Scene 2:

5 mm: 'data/pinhole/2_5mm.jpg‘

3 mm: 'data/pinhole/2_3mm.jpg‘

2 mm: 'data/pinhole/2_2mm.jpg‘

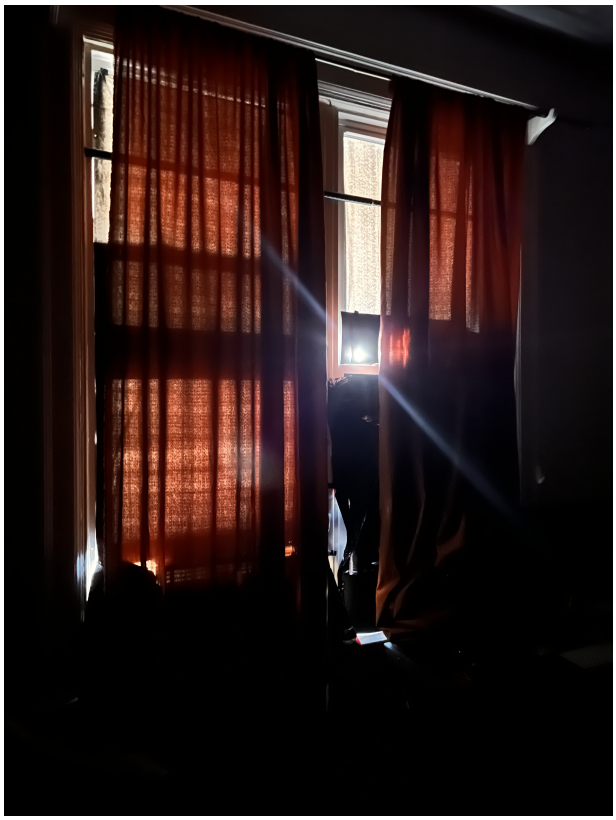
Scene 3:

5 mm: 'data/pinhole/3_5mm.jpg‘

3 mm: 'data/pinhole/3_3mm.jpg‘

2 mm: 'data/pinhole/3_2mm.jpg‘

2.3 Bonus: Camera obscura in your room



Pinhole camera setup through the window

Projected images on the wall