MP0

January 25, 2019

1 MP0 : Image Demosaicing

Welcome to CS 543! This assignment is a warm-up assignment to get you back up working from the winter break! We will try to provide you an iPython Notebook (like this) for all the future assignments! The notebook will provide you some further instructions(implementation related mainly), in addition to the ones provided on class webpage.

1.0.1 Import statements

The following cell is only for import statements. You can use any of the 3 : cv2, matplotlib or skimage for image i/o and other functions. We will provide you the names of the relevant functions for each module. **{For convenience provided at the end of the class assignment webpage}**

```
In [552]: import numpy as np
          import cv2
          import matplotlib.image as mpimg
          import matplotlib.pyplot as plt
          import skimage
          import scipy
          %matplotlib inline
```

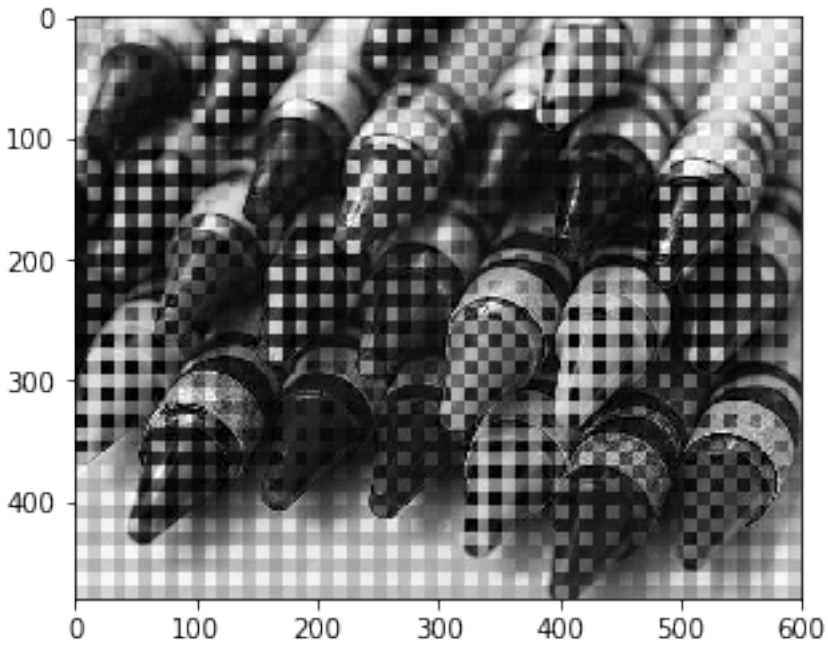
1.0.2 Reading the Mosaic Image

```
In [553]: def read_image(IMG_NAME):
              # YOUR CODE HERE
              img = cv2.imread(IMG_NAME)
              img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
              img = img.astype(np.float32)
              return img
```

```
In [554]: IMG_DIR = 'images/'
          IMG_NAME = 'crayons.bmp'
          mosaic_img = read_image(IMG_DIR+IMG_NAME)# YOUR CODE HERE
```

```
In [555]: # For a sanity check, display your image here
          plt.imshow(mosaic_img.astype(np.uint8))
```

```
Out[555]: <matplotlib.image.AxesImage at 0x1d0e0c710>
```



1.0.3 Linear Interpolation

```
In [556]: ### HINT : You might want to use filters

In [557]: ### HINT : To use filters you might want to write your kernels

In [558]: ### HINT : For writing your kernels you might want to see the RGB Pattern provided o

In [559]: ### HINT : To improve your kernels, you might want to use the squared difference
          ###        between your solution image and the original image

In [560]: def get_solution_image(mosaic_img):
              '''
              This function should return the soln image.
              Feel free to write helper functions in the above cells
              as well as change the parameters of this function.
              '''
              mosaic_shape = np.shape(mosaic_img)
              soln_image = np.zeros((mosaic_shape[0], mosaic_shape[1], 3))
              ### YOUR CODE HERE ###
              kernel_R = np.array([[1,2,1],
                                   [2,4,2],
                                   [1,2,1]])/4
              kernel_G = np.array([[0,1,0],
                                   [1,4,1],
                                   [0,1,0]])/4
```

```
    kernel_B = kernel_R

    Red_filter_unit = np.array([[1.0,0.0],
                                [0.0,0.0]]);
    Red_filter = np.tile(Red_filter_unit,[int(mosaic_shape[0]/2),int(mosaic_shape[1]/
    Red_Channel = mosaic_img[:,:,0]*Red_filter

    Green_filter_unit = np.array([[0.0,1.0],
                                [1.0,0.0]]);
    Green_filter = np.tile(Green_filter_unit,[int(mosaic_shape[0]/2),int(mosaic_shape
    Green_Channel = mosaic_img[:,:,1]*Green_filter

    Blue_filter_unit = np.array([[0.0,0.0],
                                [0.0,1.0]]);
    Blue_filter = np.tile(Blue_filter_unit,[int(mosaic_shape[0]/2),int(mosaic_shape[1
    Blue_Channel = mosaic_img[:,:,2]*Blue_filter

    soln_image[:,:,0] = scipy.ndimage.convolve(Red_Channel,kernel_R,mode='constant',c
    soln_image[:,:,1] = scipy.ndimage.convolve(Green_Channel,kernel_G,mode='constant'
    soln_image[:,:,2] = scipy.ndimage.convolve(Blue_Channel,kernel_B,mode='constant'

    return soln_image
```

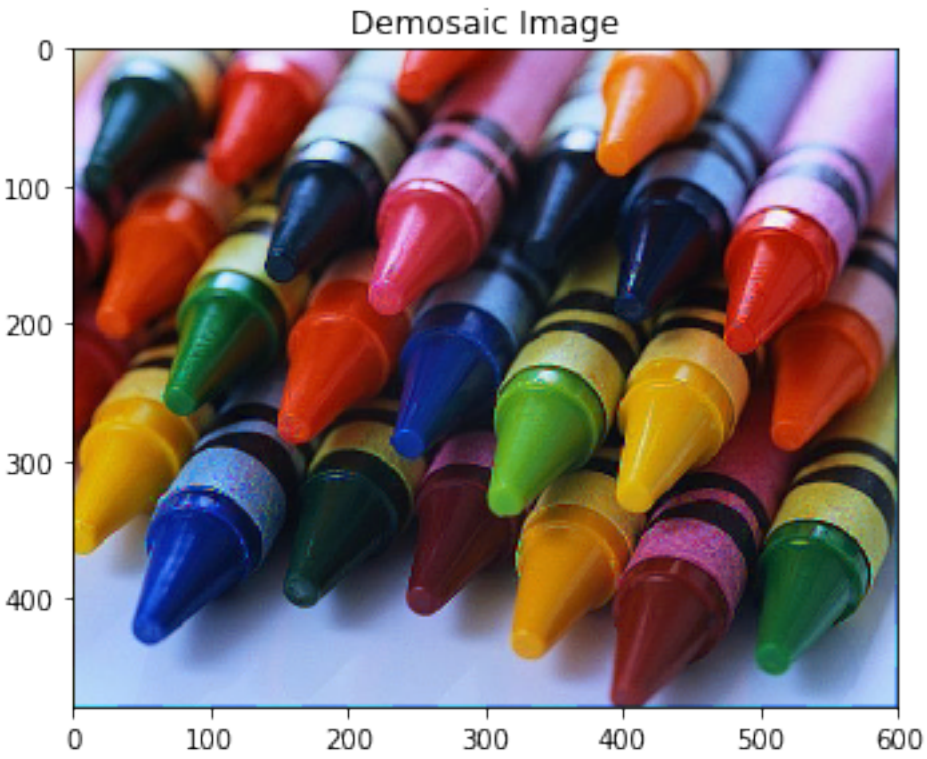
```
In [561]: def compute_errors(soln_image, original_image):

    ssd = np.sum((soln_image[:,:,0:3]-original_image[:,:,0:3])**2, axis = 2)

  #  print(ssd)
    plt.figure
    plt.figure(figsize=(20,10))
    plt.imshow(ssd.astype(np.uint8),cmap = 'gray')
    plt.colorbar()
    plt.title('SSD')
    pp_err = np.average(ssd)
    max_err = np.max(ssd)

    '''
    Compute the Average and Maximum per-pixel error
    for the image.

    Also generate the map of pixel differences
    to visualize where the mistakes are made
    '''
    return pp_err, max_err
```

We provide you with 3 images to test if your solution works. Once it works, you should generate the solution for test image provided to you.

```
In [562]: mosaic_img = read_image(IMG_DIR+'crayons.bmp')
```

```
soln_image = get_solution_image(mosaic_img)
original_image = read_image(IMG_DIR+'crayons.jpg')
# For sanity check display your solution image here
### YOUR CODE
plt.figure(figsize=(20,10))
plt.subplot(211)
plt.imshow(soln_image.astype(np.uint8))
plt.title('Demosaic Image')
plt.subplot(212)
plt.imshow(original_image.astype(np.uint8))
plt.title('Original Image')
```

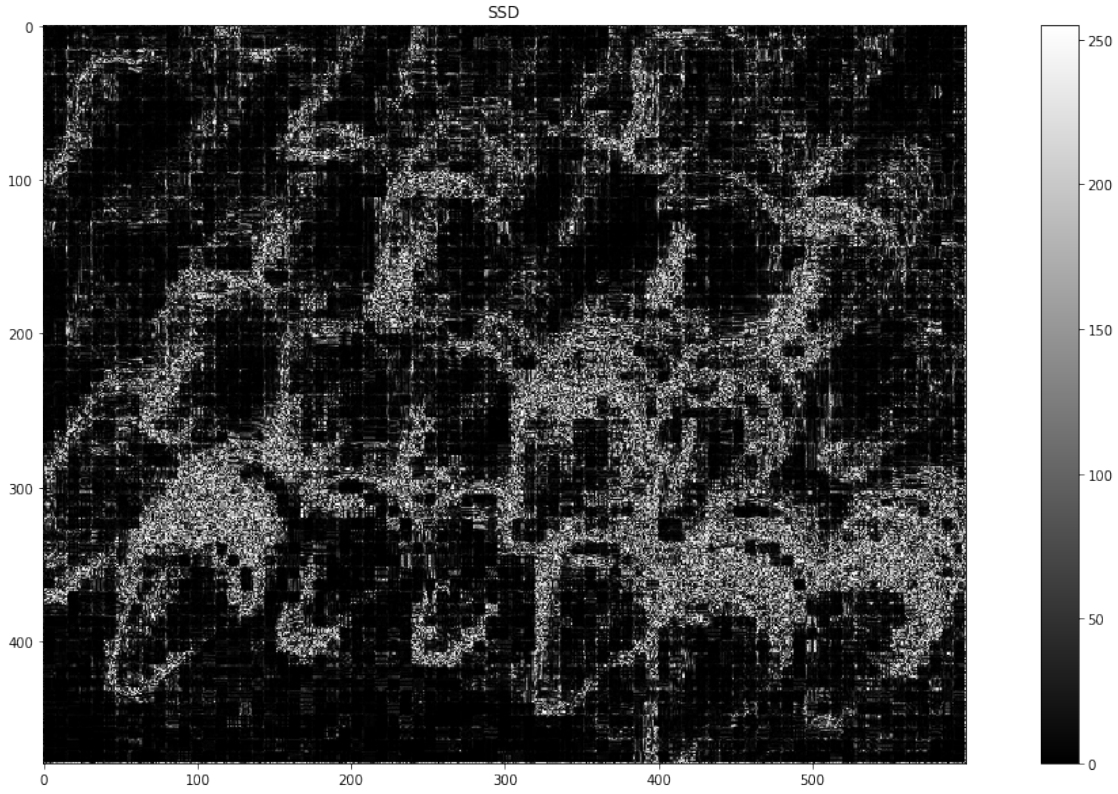
```
Out[562]: Text(0.5, 1.0, 'Original Image')
```



In [563]: pp_err, max_err = compute_errors(soln_image, original_image)

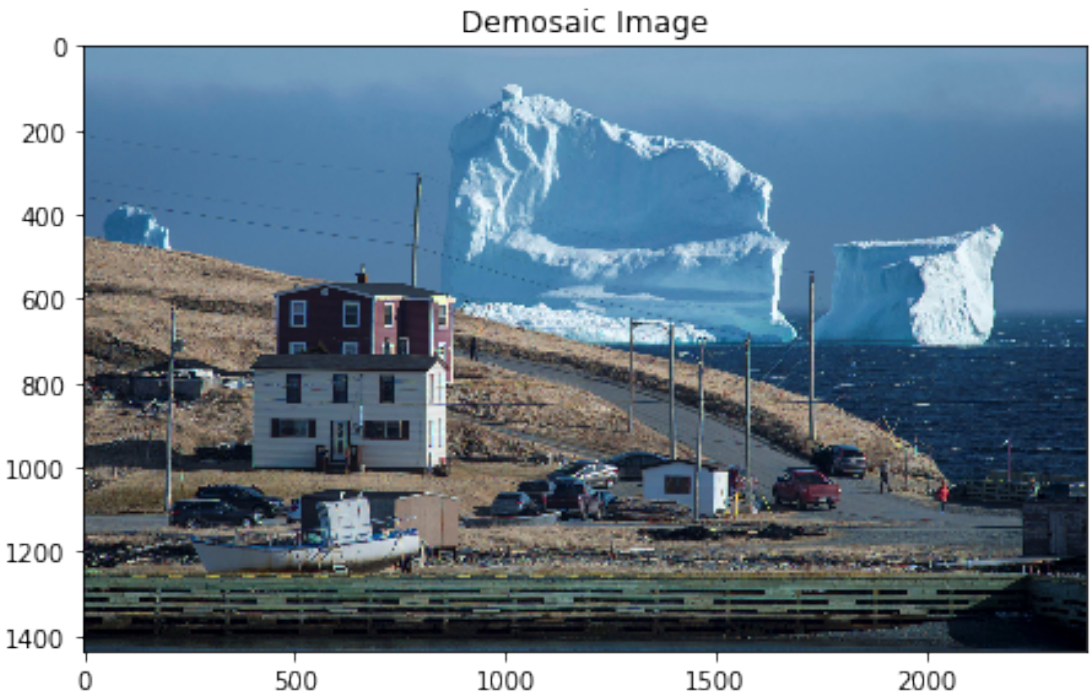
```
        print("The average per-pixel error for crayons is: "+str(pp_err))
        print("The maximum per-pixel error for crayons is: "+str(max_err))
```

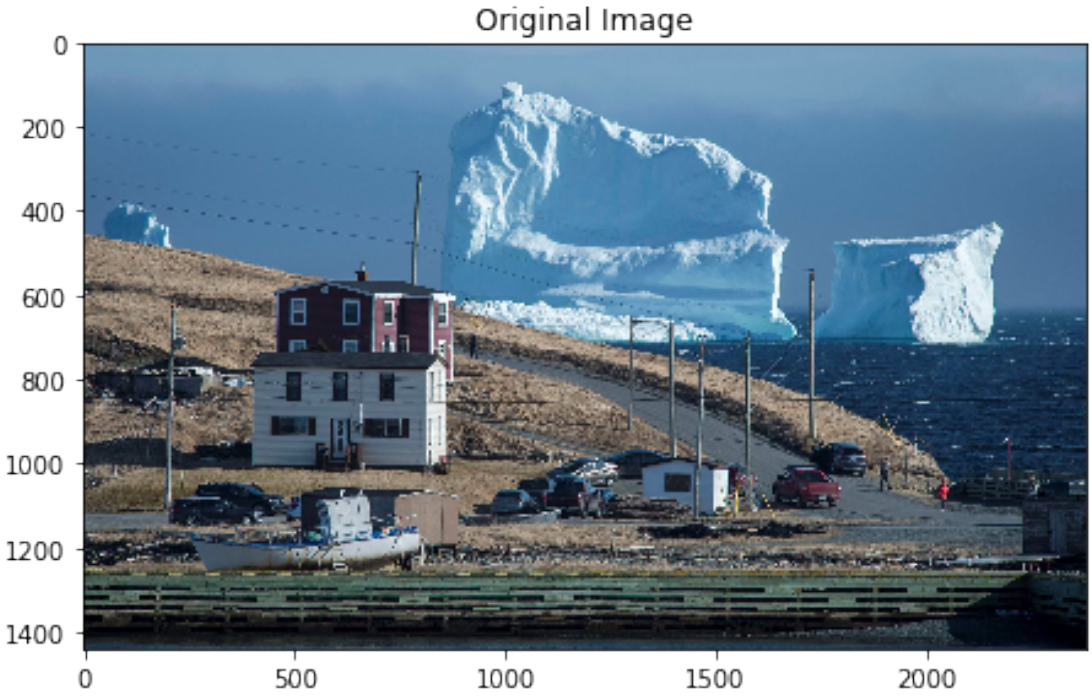
```
The average per-pixel error for crayons is: 211.95443836805555
The maximum per-pixel error for crayons is: 53478.125
```




```
In [564]: mosaic_img = read_image('iceberg.bmp')
          soln_image = get_solution_image(mosaic_img)
          original_image = read_image('iceberg.jpg')
          # For sanity check display your solution image here
          ### YOUR CODE
          plt.figure(figsize=(20,10))
          plt.subplot(211)
          plt.imshow(soln_image.astype(np.uint8))
          plt.title('Demosaic Image')
          plt.subplot(212)
          plt.imshow(original_image.astype(np.uint8))
          plt.title('Original Image')
```

```
Out[564]: Text(0.5, 1.0, 'Original Image')
```



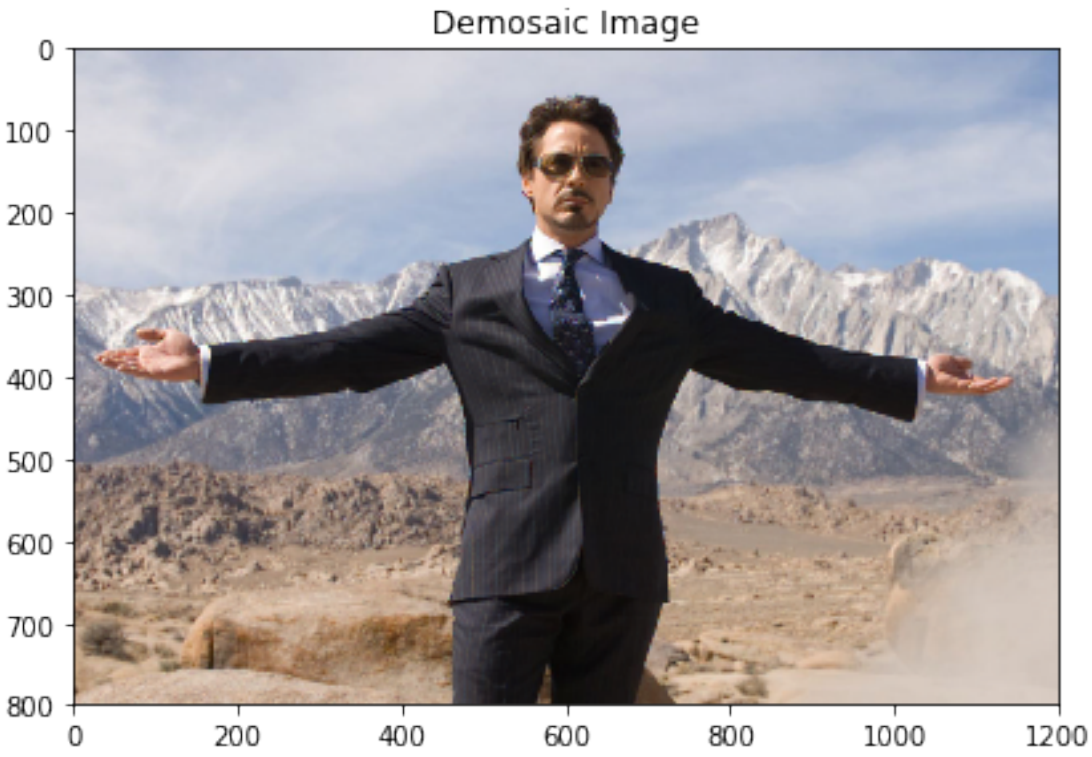


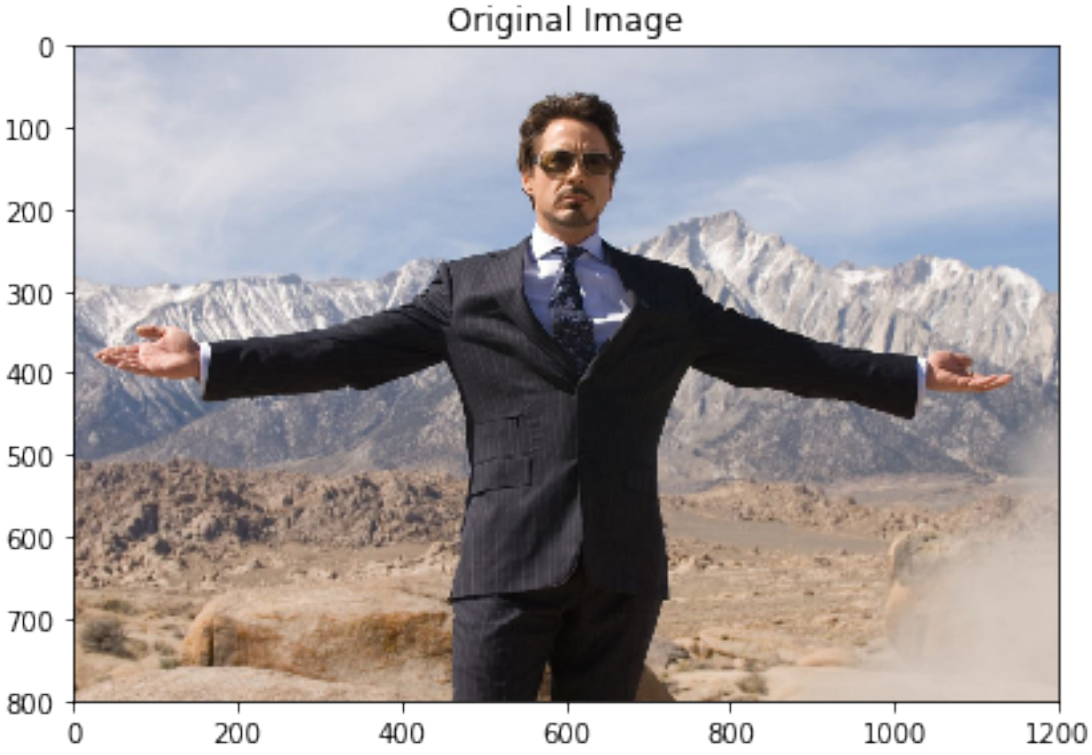

```
In [565]: pp_err, max_err = compute_errors(soln_image, original_image)
          print("The average per-pixel error for iceberg is: "+str(pp_err))
          print("The maximum per-pixel error for iceberg is: "+str(max_err))
```

```
The average per-pixel error for iceberg is: 114.55224648550471
```

```
The maximum per-pixel error for iceberg is: 30331.5625
```


```
In [566]: mosaic_img = read_image('tony.bmp')
          soln_image = get_solution_image(mosaic_img)
          original_image = read_image('tony.jpg')
          # For sanity check display your solution image here
          ### YOUR CODE
          plt.figure(figsize=(20,10))
          plt.subplot(211)
          plt.imshow(soln_image.astype(np.uint8))
          plt.title('Demosaic Image')
          plt.subplot(212)
          plt.imshow(original_image.astype(np.uint8))
          plt.title('Original Image')

Out[566]: Text(0.5, 1.0, 'Original Image')
```





In [567]: pp_err, max_err = compute_errors(soln_image, original_image)

```
        print("The average per-pixel error for tony is: "+str(pp_err))
        print("The maximum per-pixel error for tony is: "+str(max_err))
```

```
The average per-pixel error for tony is: 57.5956853515625
The maximum per-pixel error for tony is: 36891.5625
```


```
In [568]: mosaic_img = read_image('hope.bmp')
          soln_image = get_solution_image(mosaic_img)
          plt.figure(figsize=(20,10))
          plt.imshow(soln_image.astype(np.uint8))
          plt.title('Demosaic Image')
          # Generate your solution image here and show it

Out[568]: Text(0.5, 1.0, 'Demosaic Image')
```

1.0.4 Freeman's Method

For details of the freeman's method refer to the class assignment webpage.

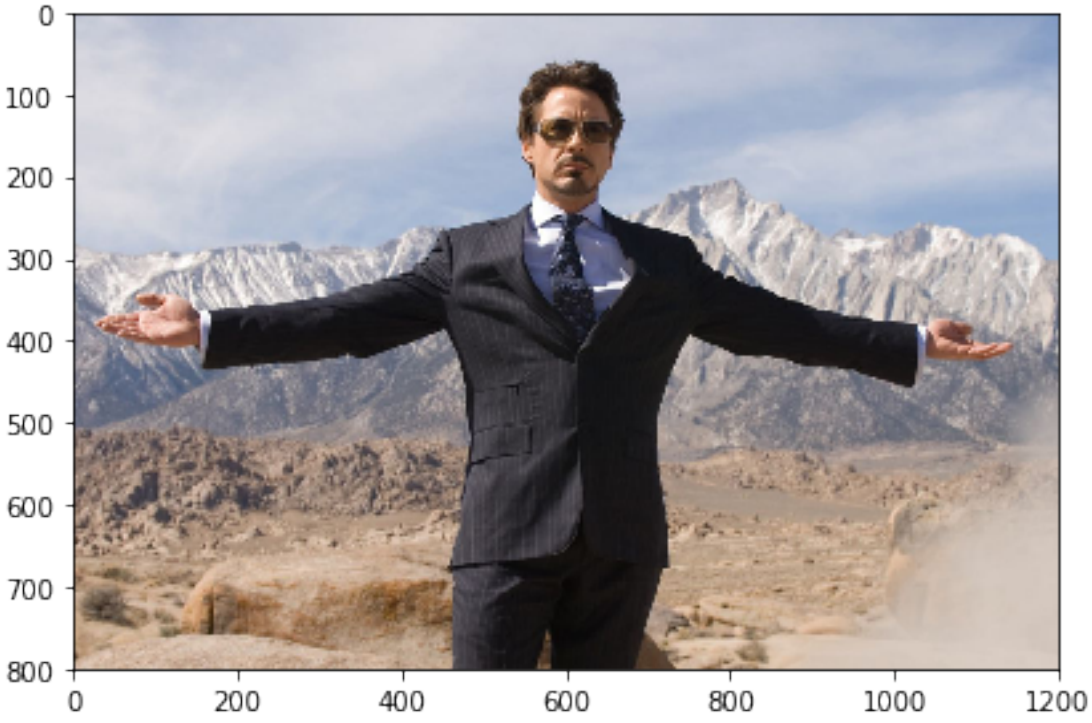
MAKE SURE YOU FINISH LINEAR INTERPOLATION BEFORE STARTING THIS PART!!!

```
In [569]: def get_freeman_solution_image(mosaic_img):
              '''
              This function should return the freeman soln image.
              Feel free to write helper functions in the above cells
              as well as change the parameters of this function.

              HINT : Use the above get_solution_image function.
              '''
              ### YOUR CODE HERE ###
              # Get the linear interpolation image
              soln_image = get_solution_image(mosaic_img)

              soln_image_Red = soln_image[:,:,0]
              soln_image_Green = soln_image[:,:,1]
```

```
    soln_image_Blue = soln_image[:,:,2]

    R_G = soln_image_Red - soln_image_Green
    B_G = soln_image_Blue - soln_image_Green

    R_G = scipy.signal.medfilt2d(R_G)
    B_G = scipy.signal.medfilt2d(B_G)

    R = R_G + soln_image_Green
    B = B_G + soln_image_Green
    G = soln_image_Green

    freeman_soln_image = np.zeros([soln_image.shape[0], soln_image.shape[1],soln_imag


    freeman_soln_image[:,:,0] = R
    freeman_soln_image[:,:,1] = G
    freeman_soln_image[:,:,2] = B
    return freeman_soln_image
```

In [570]:
```
mosaic_img = read_image('tony.bmp')
soln_image = get_freeman_solution_image(mosaic_img)
original_image = read_image('tony.jpg')
# For sanity check display your solution image here
### YOUR CODE
plt.figure(figsize=(20,10))
plt.subplot(211)
plt.imshow(soln_image.astype(np.uint8))
plt.subplot(212)
plt.imshow(original_image.astype(np.uint8))
```

Out[570]:
```
<matplotlib.image.AxesImage at 0x1eda8c780>
```




```
In [571]: pp_err, max_err = compute_errors(soln_image, original_image)
          print("The average per-pixel error for tony is: "+str(pp_err))
```

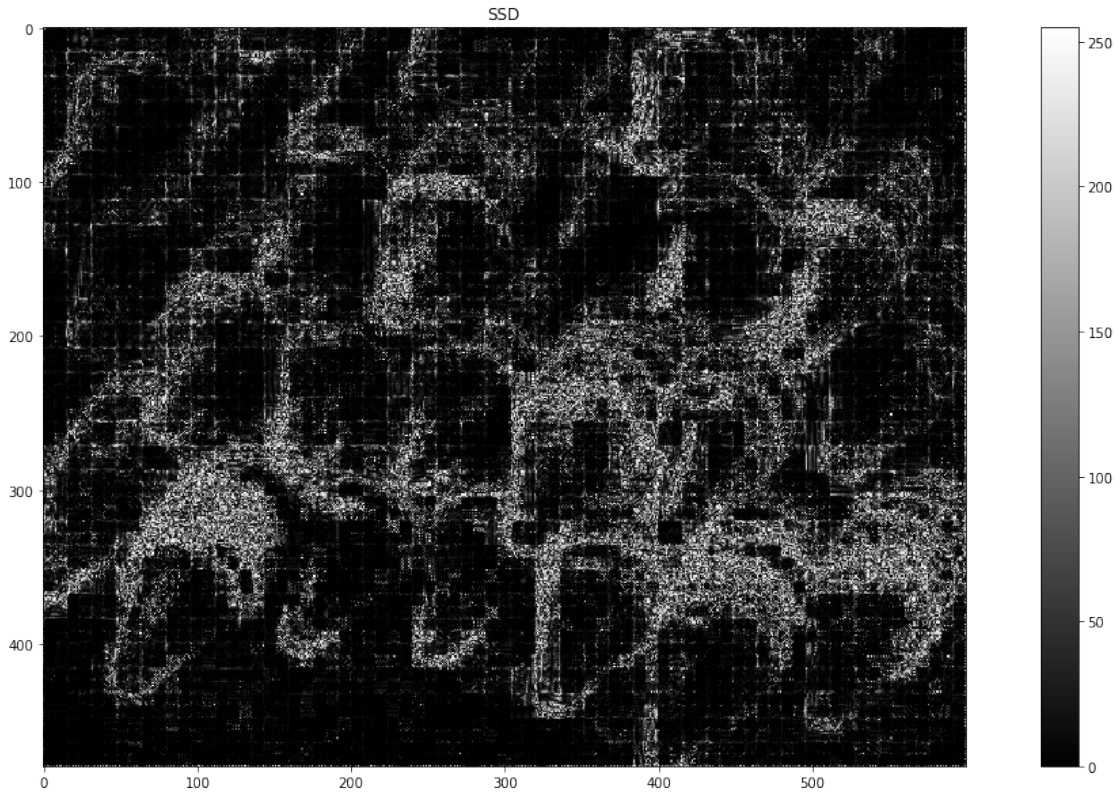
```
        print("The maximum per-pixel error for tony is: "+str(max_err))
```

```
The average per-pixel error for tony is: 28.8272189453125
The maximum per-pixel error for tony is: 32278.0
```

In [572]: *### Feel free to play around with other images for Freeman's method above ###*

In [573]:
```
mosaic_img = read_image('crayons.bmp')
soln_image = get_freeman_solution_image(mosaic_img)
original_image = read_image('crayons.jpg')
pp_err, max_err = compute_errors(soln_image, original_image)
print("The average per-pixel error for crayons is: "+str(pp_err))
print("The maximum per-pixel error for crayons is: "+str(max_err))
```

```
The average per-pixel error for crayons is: 142.63010438368056
The maximum per-pixel error for crayons is: 47857.625
```




```
In [574]: mosaic_img = read_image('iceberg.bmp')
          soln_image = get_freeman_solution_image(mosaic_img)
          original_image = read_image('iceberg.jpg')
          pp_err, max_err = compute_errors(soln_image, original_image)
          print("The average per-pixel error for iceberg is: "+str(pp_err))
          print("The maximum per-pixel error for iceberg is: "+str(max_err))

The average per-pixel error for iceberg is: 72.9752010339409
The maximum per-pixel error for iceberg is: 33839.5625
```


```
In [575]: mosaic_img = read_image('hope.bmp')
          soln_image = get_freeman_solution_image(mosaic_img)
          # Generate your solution image here and show it
          plt.figure(figsize=(20,10))
          plt.imshow(soln_image.astype(np.uint8))

Out[575]: <matplotlib.image.AxesImage at 0x1efeaa2e8>
```

1.0.5 Mosaicing an Image

Now lets take a step backwards and mosaic an image.

```
In [576]: def get_mosaic_image(original_image):
              '''
              Generate the mosaic image using the Bayer Pattern.
              '''
              Red_filter_unit = np.array([[1.0,0.0],
                                          [0.0,0.0]]);

              Red_filter = np.tile(Red_filter_unit,[int(original_image.shape[0]/2),int(original
              Red_Channel = original_image[:,:,0]*Red_filter

              Green_filter_unit = np.array([[0.0,1.0],
                                            [1.0,0.0]]);
              Green_filter = np.tile(Green_filter_unit,[int(original_image.shape[0]/2),int(orig
              Green_Channel = original_image[:,:,1]*Green_filter

              Blue_filter_unit = np.array([[0.0,0.0],
                                           [0.0,1.0]]);
```

```
    Blue_filter = np.tile(Blue_filter_unit,[int(original_image.shape[0]/2),int(origi
    Blue_Channel = original_image[:,:,2]*Blue_filter

    mosaic_img = np.zeros([original_image.shape[0],original_image.shape[1],original_

    mosaic_img[:,:,0] = Red_Channel + Green_Channel + Blue_Channel
    mosaic_img[:,:,1] = mosaic_img[:,:,0]
    mosaic_img[:,:,2] = mosaic_img[:,:,0]

    return mosaic_img
```

In [577]:
```
### YOU CAN USE ANY OF THE PROVIDED IMAGES TO CHECK YOUR get_mosaic_function
original_image = read_image('tony.jpg')
mosaic_img = get_mosaic_image(original_image)
plt.figure(figsize=(20,10))
plt.imshow(mosaic_img.astype(np.uint8))
```

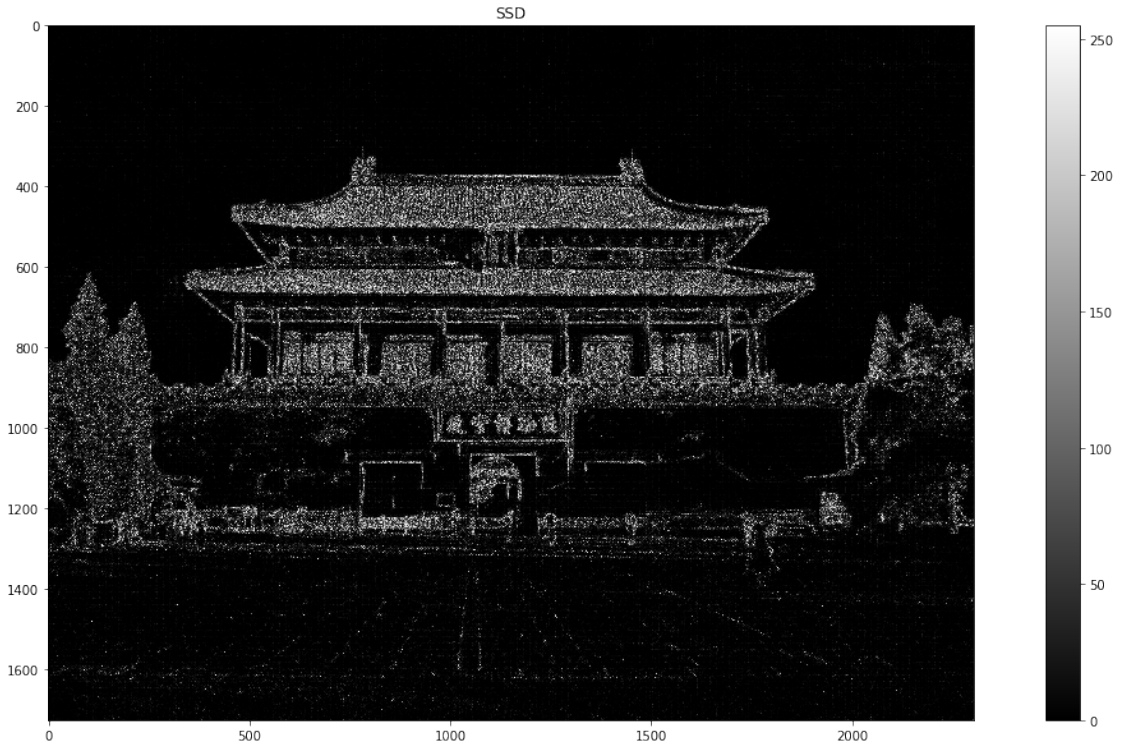
Out[577]: <matplotlib.image.AxesImage at 0x1f1a0e518>

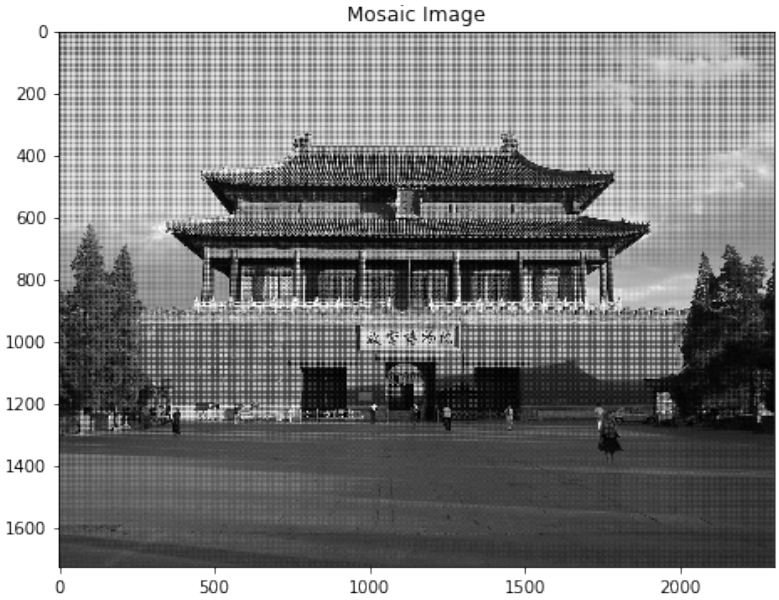
Use any 3 images you find interesting and generate their mosaics as well as their demosaics. Try to find images that break your demosaicing function.

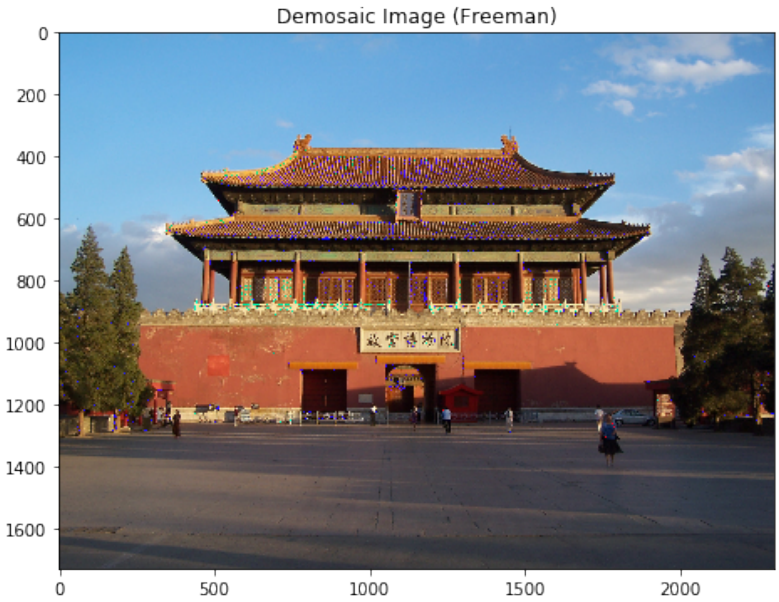
In [580]:
```
### YOUR CODE HERE ###
original_img = read_image('forbidden_city.jpg')
mosaic_img = get_mosaic_image(original_img)
```

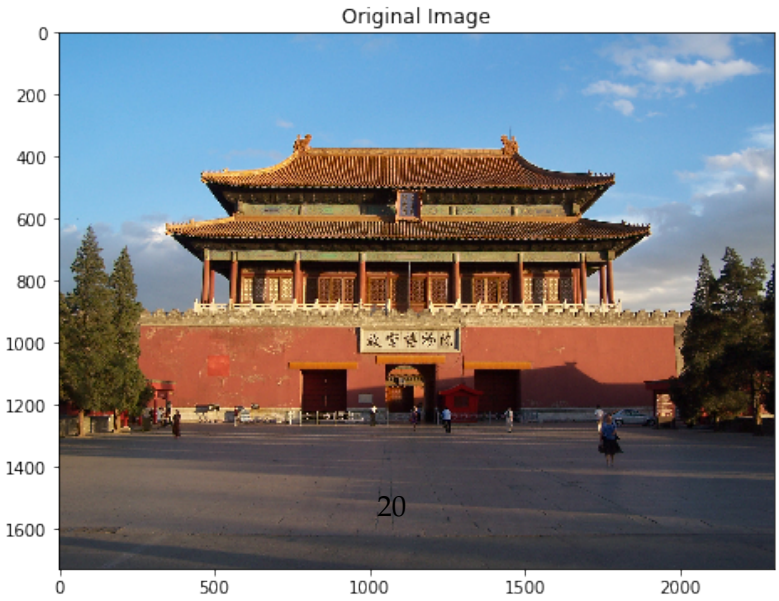
```
demosaic_img = get_freeman_solution_image(mosaic_img)
pp_err, max_err = compute_errors(demosaic_img, original_img)

plt.figure(figsize=(20,20))
plt.subplot(311)
plt.imshow(mosaic_img.astype(np.uint8))
plt.title('Mosaic Image')
plt.subplot(312)
plt.imshow(demosaic_img.astype(np.uint8))
plt.title('Demosaic Image (Freeman)')
plt.subplot(313)
plt.imshow(original_img.astype(np.uint8))
plt.title('Original Image')
```

Out[580]: Text(0.5, 1.0, 'Original Image')



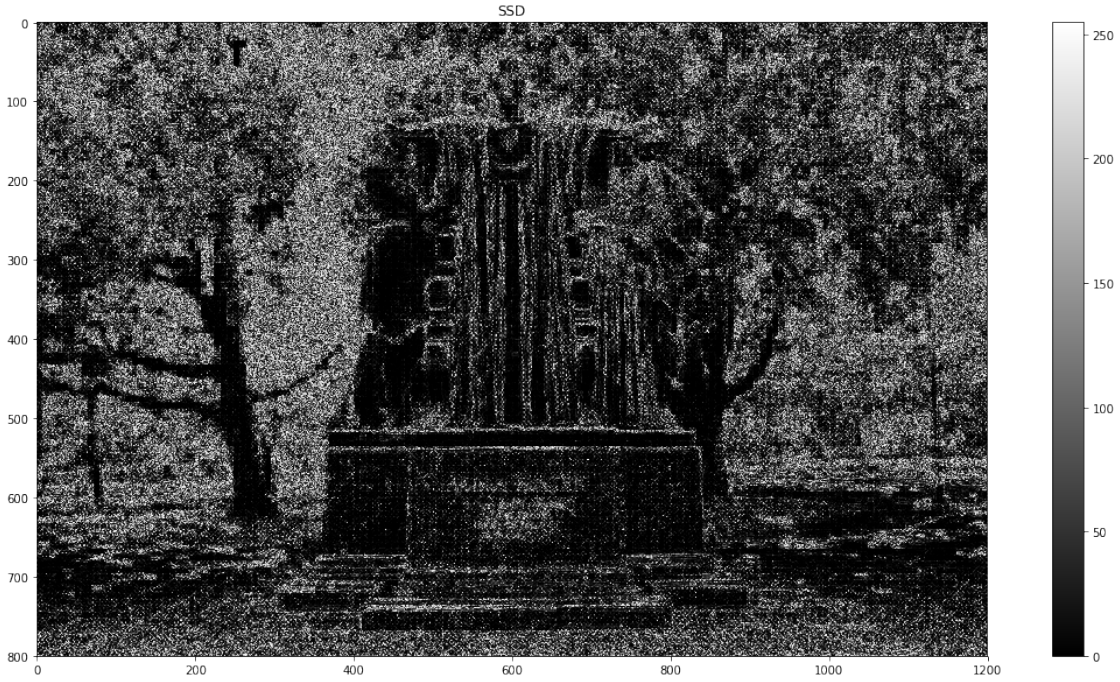

Mosaic Image
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000

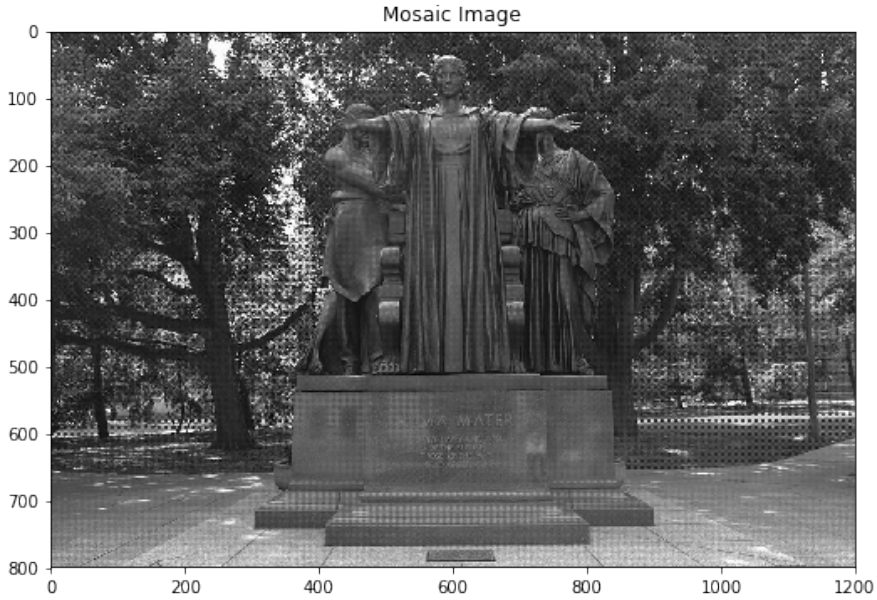

Demosaic Image (Freeman)
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000

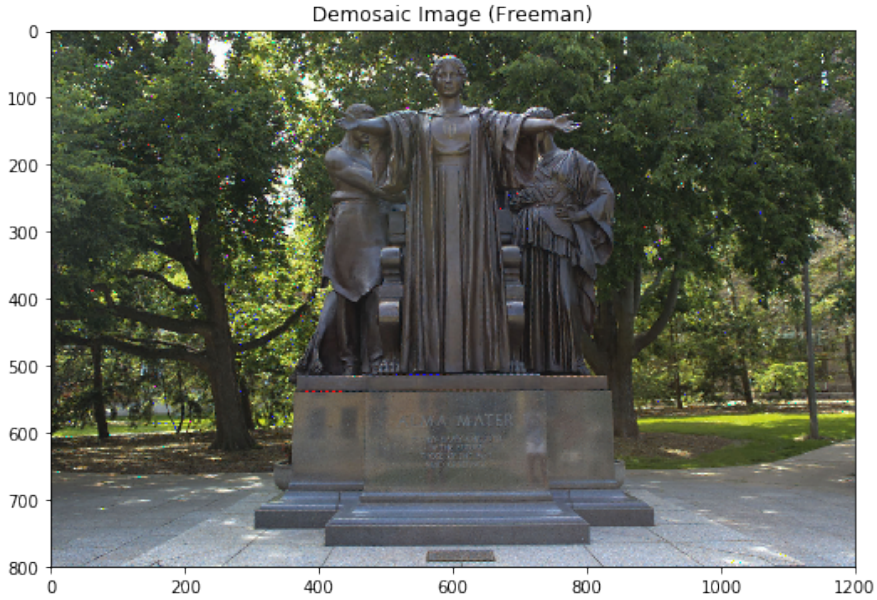

Original Image
0
200
400
600
800
1000
1200
1400
1600
0
500
1000
1500
2000

In [581]:
```
### YOUR CODE HERE ###
original_img = read_image('uiuc.jpg')
mosaic_img = get_mosaic_image(original_img)
demosaic_img = get_freeman_solution_image(mosaic_img)
pp_err, max_err = compute_errors(demosaic_img, original_img)
plt.figure(figsize=(20,20))
plt.subplot(311)
plt.imshow(mosaic_img.astype(np.uint8))
plt.title('Mosaic Image')
plt.subplot(312)
plt.imshow(demosaic_img.astype(np.uint8))
plt.title('Demosaic Image (Freeman)')
plt.subplot(313)
plt.imshow(original_img.astype(np.uint8))
plt.title('Original Image')
```

Out[581]:
```
Text(0.5, 1.0, 'Original Image')
```




Mosaic Image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


Demosaic Image (Freeman)
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


Original Image
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

1.0.6 Bonus Points

In [ ]:

In [4]:
```
### YOUR CODE HERE ###
### YOU ARE ON YOUR OWN :) ####
```