MP0 : Image Demosaicing

Welcome to CS 180! This assignment is a warm up assignment to get you back up working from the winter break. We will try to provide you with some of the basic functions for image processing. Implement what you need, you may further develop upon provided functions/implementations marked partially or skeleton in the code or provided on new webpage.

Import statements

The following cells are for import statements. You can use any of the C, C++, numpy or opencv or opencv-contrib-python-headless packages (or the rest of the relevant functions for each module). **Use conversions provided in the rest of the this assignment notebook**

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
import cv2
import os
import numpy as np
from google.colab.patches import cv2_imshow
```

Reading the Mosaic Image

```
cap_image_path = 'dataset/'
cap_image = cv2.imread(cap_image_path + 'bridge.png', cv2.IMREAD_GRAYSCALE)
```

```
# If this is a mosaiced image, generate your image here
print(mosaiced_image.shape)
```

```
(1024, 1024, 1)
```

```
# If this is a mosaiced image, show it here.
plt.imshow(mosaiced_image, cmap = 'gray')
```

Linear Interpolation

```
# TODO: Implement bilinear interpolation
# (10 points)
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def bilinear_interpolation(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
bilinear_output = bilinear_interpolation(im)
print("Output shape: ", bilinear_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(bilinear_output.astype(np.uint8))
```

Freeman's Method

```
# TODO: Implement Freeman's method
# (10 points)
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def freeman(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
freeman_output = freeman(im)
print("Output shape: ", freeman_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(freeman_output.astype(np.uint8))
```

Mosaicing an Image

```
# TODO: Implement mosaicing an image
# (5 points)
# INPUT: 3D array of RAW values of size (H, W, 3)
# OUTPUT: 2D array of mosaiced values of size (H, W)

def mosaicing(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W))

    # TODO: 1 point

    return im_output
```

```
# Mosaicing
im = cap_image[:, :, :]
print("Image shape: ", im.shape)
mosaiced_output = mosaicing(im)
print("Output shape: ", mosaiced_output.shape)
```

```
Image shape:  (1024, 1024, 3)
Output shape:  (1024, 1024)
```

```
# Show the output
plt.imshow(mosaiced_output, cmap = 'gray')
```

Linear Interpolation

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def bilinear_interpolation(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
bilinear_output = bilinear_interpolation(im)
print("Output shape: ", bilinear_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(bilinear_output.astype(np.uint8))
```

Freeman's Method

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def freeman(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
freeman_output = freeman(im)
print("Output shape: ", freeman_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(freeman_output.astype(np.uint8))
```

Mosaicing an Image

```
# TODO: 1 point
# INPUT: 3D array of RAW values of size (H, W, 3)
# OUTPUT: 2D array of mosaiced values of size (H, W)

def mosaicing(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W))

    # TODO: 1 point

    return im_output
```

```
# Mosaicing
im = cap_image[:, :, :]
print("Image shape: ", im.shape)
mosaiced_output = mosaicing(im)
print("Output shape: ", mosaiced_output.shape)
```

```
Image shape:  (1024, 1024, 3)
Output shape:  (1024, 1024)
```

```
# Show the output
plt.imshow(mosaiced_output, cmap = 'gray')
```

Linear Interpolation

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def bilinear_interpolation(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
bilinear_output = bilinear_interpolation(im)
print("Output shape: ", bilinear_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(bilinear_output.astype(np.uint8))
```

Freeman's Method

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def freeman(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
freeman_output = freeman(im)
print("Output shape: ", freeman_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(freeman_output.astype(np.uint8))
```

Mosaicing an Image

```
# TODO: 1 point
# INPUT: 3D array of RAW values of size (H, W, 3)
# OUTPUT: 2D array of mosaiced values of size (H, W)

def mosaicing(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W))

    # TODO: 1 point

    return im_output
```

```
# Mosaicing
im = cap_image[:, :, :]
print("Image shape: ", im.shape)
mosaiced_output = mosaicing(im)
print("Output shape: ", mosaiced_output.shape)
```

```
Image shape:  (1024, 1024, 3)
Output shape:  (1024, 1024)
```

```
# Show the output
plt.imshow(mosaiced_output, cmap = 'gray')
```


Linear Interpolation

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def bilinear_interpolation(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
bilinear_output = bilinear_interpolation(im)
print("Output shape: ", bilinear_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(bilinear_output.astype(np.uint8))
```


Freeman's Method

```
# TODO: 2 points
# INPUT: 2D array of RAW values of size (H, W)
# OUTPUT: 3D array of interpolated values of size (H, W, 3)

def freeman(im):
    '''
    raw_shape = im.shape
    H = raw_shape[0]
    W = raw_shape[1]

    im_output = np.zeros((H, W, 3))

    # TODO: 1 point

    return im_output
```

```
# Interpolate
im = mosaiced_image[:, :, 0]
print("Mosaiced image shape: ", im.shape)
freeman_output = freeman(im)
print("Output shape: ", freeman_output.shape)
```

```
Mosaiced image shape:  (1024, 1024)
Output shape:  (1024, 1024, 3)
```

```
# Show the output
plt.imshow(freeman_output.astype(np.uint8))
```


Mosaicing an Image

Now we use a cute bulldozer as our mosaic an image.

```
C:\Users\ZIYUPENG\AI_Tools\Python\Python312\Lib\site-packages\librosa\core\spectrum.py:224:UserWarning:n_fft=2048 is too small for input signal of length=1
  warnings.warn(f"n_fft={n_fft} is too small for input signal of length={n_samples}")
C:\Users\ZIYUPENG\AI_Tools\Python\Python312\Lib\site-packages\librosa\core\spectrum.py:224:UserWarning:n_fft=2048 is too small for input signal of length=1
  warnings.warn(f"n_fft={n_fft} is too small for input signal of length={n_samples}")
```

[0/15]

```
DEV FOOD LOSS:NAN NAN
EVAL FOOD LOSS:NAN NAN | ACC:0.
```