CPE 428: Computer Vision
Homework 1: RAW Processing
Instructor: Jonathan Ventura

In this homework you will implement a basic pipeline to convert the raw sensor output of a camera into a processed JPEG image.

The raw sensor output comes in the form of a 16-bit array of values, stored in a DNG file. These values represent the per-pixel brightness readouts from a sensor with a Bayer camera filter array. The Bayer pattern is the standard pattern:

RGRG
GBGB
RGRG
GBGB

The template code in process.py reads in the DNG file and extracts the raw image data in 16-bit integer format. You need to complete process.py and the utility functions in raw.py to convert the raw image into a JPEG. Implement the following steps in the pipeline:

1. Convert the raw image data to 32-bit floating point in [0 1] range. (Note: the normalization should be done by dividing by 2^16-1 rather than the maximum value in the image.)
2. Demosaic the red, green, and blue channels using bilinear interpolation. (This should be implemented from scratch, not using imported python functions from other libraries like OpenCV or similar. Using NumPy functions is okay, of course.)
3. Apply white balancing by scaling each channel so that its mean is 0.5.
4. Apply an inverse gamma curve x' = x^(1/gamma) where 1/gamma = 0.85.
5. Clip and quantize: clip to [0 1] range, scale by 255, and convert to 8-bit unsigned integer.
6. Save the image both as a PNG and a JPEG.

Your code must fill in the functions demosaic(), white_balance(), and curve_and_quantize() provided in the template python file raw.py. These functions will be tested in an automated script during grading, so be sure not to change their names, arguments, or return types. Also fill in the command line app process.py to read in the raw image data, call the processing functions and output the JPEG.

Your implementations of white_balance() and curve_and_quantize() should not use for loops to iterate over the pixels in the image. Numpy arrays and functions automatically apply operations over all pixels, allowing you to avoid slow for loops in your code. It is okay to use for loops in demosaic() since an implementation without for loops would be pretty challenging (although it would be much faster)!

Note: The example DNG files are quite large, and your algorithm might be slow (for loops are notoriously slow in Python). To facilitate debugging, you might want to create a small RAW array by hand and test it with your methods.

Required libraries:

- rawpy
- numpy
- imageio

Useful functions:
- np.clip() to clip values
- np.mean() to compute the mean
- np.pad() for padding
- np.stack() to combine separate arrays into a multi-channel image
- imageio.imwrite() to save the PNG and JPEG

Links to example DNG files and processed JPEGs:
- L1004220.DNG
- L1004220.JPG
- L1004235.DNG
- L1004235.JPG
- L1004432.DNG
- L1004432.JPG

Deliverables:
- raw.py
- process.py
- Text document (plain text, Word doc, or PDF) explaining your solution

Please upload these files individually, not as a ZIP – this makes grading easier.

Deadline: Sunday, Oct 2, 11:59pm