Ankur Kejriwal
Kejria1
400113206

Project 2

Instructions for running the program

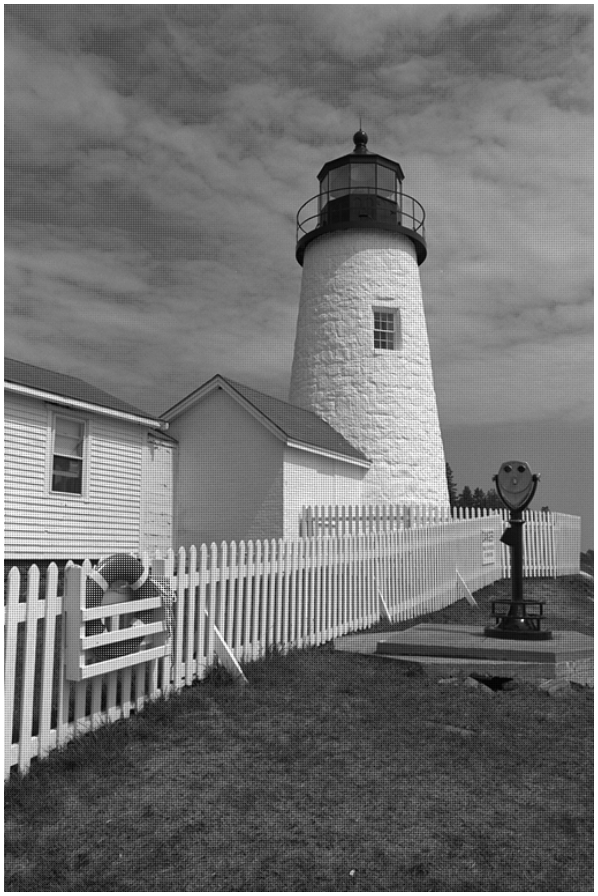
The least-square algorithm is implemented in Matlab with help from the image processing toolkit. The program consists of two files coeff.m(training portion on a separate image) and project2.m (using the trained coefficients and demosaicing) which produce a 3 channel demosaicked image and outputs the mse to the terminal window. Run coeff.m first and then run the project2.m file to ensure the coefficients are generated and saved in the workspace before

The main file is project2.m which contains all the relevant functions and code required to run the project and the image provided is the lighthouse.png image which was used in project 1 as well. To change the input image change the filename on line 4. The file outputs three images one of which is the mosaicked image in 3 channel format(mosaic3Channel.png), the other is a 1 channel representation of the same image(mosaick1Channel.png), and the last being the interpolated image (CompleteImage.png).

The Mean squared error is outputted to the console for both my interpolation implementation and using Matlab's demosaic function and are labeled accordingly.

CFA Image Generation:

Simulation of the camera sensor was performed by downsampling the input image in a similar fashion to the CFA filter on top of conventional cameras. To mosaic the input image I created 2x2 matrices for each channel with a 1 in the position of the sample we want to retain and scaled it up to the size of the input image and multiplied by each channel of the image independently and compressed them to form the Bayer input image(which follows the RGGB pattern) for my interpolation algorithm.



Downsampled Bayer Image

Algorithm Design:

The algorithm that is implemented is based on the least-squares regression approach explored in class where 8 sets of coefficients are produced from a training image which based on the pattern of the patch is used to approximate the missing 2 color channels in the patch. For example in a 5x5 RGGB patch, the middle pixel is a Red pixel with a missing green and blue channel value. Using the coefficients associated with an RGGB patch you can multiply and add the 25x1 coefficients for the green and blue channel pixel with the patch to produce the missing two values. This process is repeated for the 3 different patches and the type of patch is decided by using the location of the red pixel in the original downsampled image and based on the type of patch the appropriate pixels are used. In the preprocessing stage, the image is symmetrically padded to ensure that we don't index outside of the range of the picture and is trimmed to ensure the Bayer pattern remains. In the training phase, the image is padded to ensure that when training we have exactly Height*width columns as an output of the im2col function which is used to store the 25 pixels (5x5) needed for each patch.

$$A = (X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}R = X^{+}R$$

Figure 1 General Equation used to find coefficients for a patch

The 3 2D matrices which hold the interpolated red, green, and blue values are concatenated into one 3D matrix and passed to the image write function to produce the output image. Output images can be seen in the zip folder accompanying this submission.

Implementation of the Algorithm

The sample images for training were chosen carefully as they contained high frequency information and contained a wide range of colours which would assist in the accuracy of the coefficients generated in the training stage. Training is done in the coeff.m file in order to reduce the run time everytime you change the interpolation image. The image is padded 2 pixels in each direction to ensure we have the exact amount of columns from the im2col function as the pixels in the image as each column is the 5x5 window we use to find the coefficients. Each individual channel of the image was converted to a single column using the im2col function in order to simplify calculations.

Coefficients of each patch were split into function which are defined by the pattern of the patch in question. RGGB would mean the 2x2 window in the top left of the 5x5 window followed an RGGB pattern. Each of these functions generates 3 5x5 matrices using repmat on a 2x2 matrix with a 1 in place of the pixel sample we want to retain and trims the end to have a 5x5 matrix which is the size of the window in question. It multiplies the output from repmat with the window which produces the downsample channel and each channel is concatenated to produce the mosaic which is then used by the equation in figure 1 to generate the coefficients.

After running the coeff.m file, the interpolation is done in the project1.m file where it uses the coefficients and iterates over the whole image and uses the location of the red pixel in the downsampled channel to decide what type of patch it is currently on. A 2x2 window is pulled from the top left of the 5x5 window and multiplied by a mask with 0s in the location of where the red pixels should be and is compared to a matrix of zeros of the same size which would indicate where the red pixel is located which helps decide what pattern the patch follows. A 5x5 window

is created around the pixel we are currently on and multiplies it by the coefficients for that patch and interpolates the two missing pixels and writes that to the output channel for that colour

Error:

The mean squared error for my implementation was calculated using the immse function in Matlab and was found to be **22.5711** and I compared it to the Matlab demosaic function which produced an image with a mse of **27.5216**. Comparing the image produced by Matlab's demosaic function, it is evident that the least squares approach reduced the zipper effect near the fence region and on the rocks which shows that the algorithm that I implemented reduced artefacts created by interpolation.

Zipper pattern as seen in Least squares demosaicing

Zipper pattern seen in Matlab's demosaicing