

3SK3 Project 1 Lab Report:

Divesh Chudasama – 400115575 – chudasad

Part 1: Mosaic

Generating the mosaic formatted image using the Bayer CFA pattern involved looping through each channel and pulling the samples from where the sensors would for the respective colour.

Part2: Algorithm and Implementation

Using the generated mosaic data, I then go on to process it to interpolate the samples first for the red, blue then green channels. Each takes a relatively long time to compute, since I implemented the process with a naive approach with multiple nested for loops that add to the time complexity of the code.

For the red channel I used bicubic interpolation as my algorithm of choice. Since there are fewer points in the red and blue channels, I thought bicubic would provide a more accurate approximation versus bilinear. Since the spacing of the samples is not like a checkerboard like with green, it was a challenge to find a way to center the 16 points around each interpolation point. Instead for each point I preform 3 interpolations in the shape of a backwards “L”, this helps save some time on computing and spatially made sense.

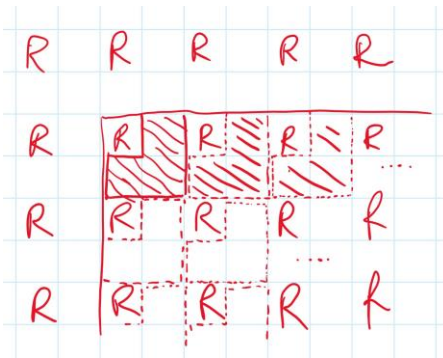


Figure 1 Red channel sampling block

$$f(x, y) = [x^3, x^2, x, 1] * B * F * (B^{-1})^T * [y^3, y^2, y, 1]^T$$

$$F = \begin{bmatrix} f(0,0) & \cdots & f(0,6) \\ \vdots & \ddots & \vdots \\ f(6,0) & \cdots & f(6,6) \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 216 & \cdots & 1 \end{bmatrix}$$

Interpolate:

$$f(2,3) = [8, 4, 2, 1] * B * F * (B^{-1})^T * [27, 9, 3, 1]^T$$

$$f(3,3) = [27, 9, 3, 1] * B * F * (B^{-1})^T * [27, 9, 3, 1]^T$$

$$f(2,3) = [27, 9, 3, 1] * B * F * (B^{-1})^T * [8, 4, 2, 1]^T$$

One of the other issues was the various edge cases, I implemented a series of if/else statements to resolve each one and pass in the nearest available sample as a filler for the missing ones. A similar approach is taken for the blue channel. The structure for interpolation was slightly different, with an upside down “L” unlike in the case of the red channel.

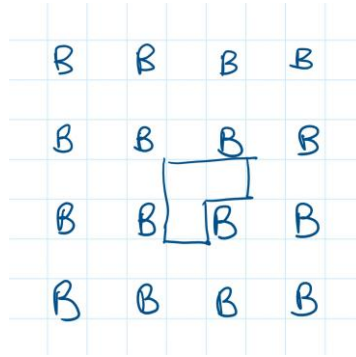


Figure 2 Blue channel sampling block

$$f(x, y) = [x^3, x^2, x, 1] * B * F * (B^{-1})^T * [y^3, y^2, y, 1]^T$$

$$F = \begin{bmatrix} f(0,0) & \cdots & f(0,6) \\ \vdots & \ddots & \vdots \\ f(6,0) & \cdots & f(6,6) \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 216 & \cdots & 1 \end{bmatrix}$$

Interpolate:

$$f(3,3) = [27, 9, 3, 1] * B * F * (B^{-1})^T * [27, 9, 3, 1]^T$$

$$f(4,3) = [64, 16, 4, 1] * B * F * (B^{-1})^T * [27, 9, 3, 1]^T$$

$$f(3,4) = [27, 9, 3, 1] * B * F * (B^{-1})^T * [64, 16, 4, 1]^T$$

As for the green channel, since there are twice as many samples in the green space, I decided to use bilinear interpolation. This would be a little faster in terms of runtime with fewer matrix calculations and also each interpolation point can be centered with 4 sample points.

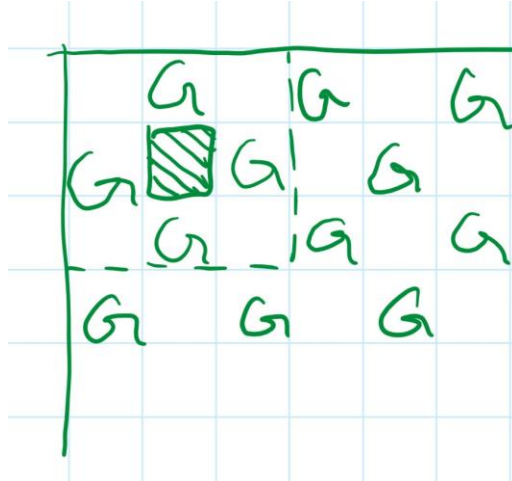


Figure 3 Green channel sampling block

$$f(x, y) = a_3xy + a_2x + a_1y + 1a_0$$

$$F = a * B$$

$$a = [f(0,0), f(0,1), f(1,0), f(1,1)]^T * \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$$

$$f(x, y) = [xy, x, y, 1] * F * B^{-1}$$

$$f(0.5, 0.5) = [0.25, 0.5, 0.5, 1] * F * B^{-1}$$

Implementation

For implementing the coordinate system, I made use of the multidimension arrays that numpy provides, where the height is the x and width is y. This may seem counter intuitive but since the indexing happens like `array[x][y]` It seemed best to stick to this format since the notation for the function also follows $f(x,y)$. For the sake of simplicity I followed this notation.

In order to do the actual processing we loop through each of the spots that need to be filled in. This is done with nested for loops, although not as efficient it is easier to implement. Then, we pull a block of samples from the image data as such:

Red samples: `block = imgdata[i - 2:i + 5, j - 3:j + 3]`

Blue samples: `block = imgdata[i - 3:i + 3, j - 3:j + 3]`

Green samples: `block = imgdata[i - 1:i + 2, j - 1:j + 2]`

In the case of red our first interpolated sample is not exactly in the center, thus resulting in the indexing to be slightly different from the blue samples. Referring to figure 1 containing the “L” shape for the samples we fill in the red channel. This approach will cause issues near edges and corners, I used if statements to resolve them and put the samples we do have into the block and then fill the rest of the block with the first available sample. Although more time consuming than padding the whole image data this gives the missing data some data that more local to that specific block. I created a `fill_missing()` function to do this.

I have 2 files in the project directory, the main one is for running and demosaicing the input image, the second one "utils.py" contains all the helper functions that carry out some of the smaller tasks. For example, the actual matrix calculations like multiplying F, B and other matrices together to compute the interpolation.

Running code:

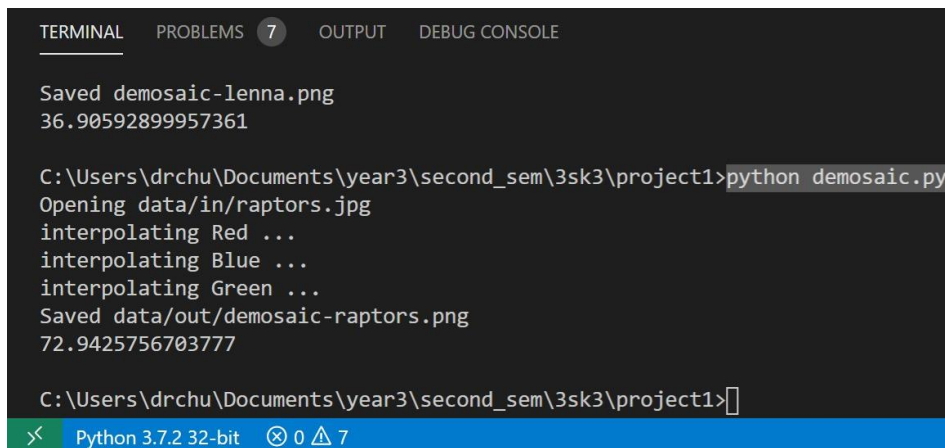
More detailed instructions can also be found in the README file. For the design I chose to use python and some of the supported libraries such as numpy and Pillow. Numpy was mainly used for handling arrays and for processing matrix operations such as multiplication and inverses, Pillow was used to load in the input files and also generate the output files with the interpolated rgb channels. Here is how to run the project inside the main directory:

```
$ python demosaic.py "<input file path>" "<output file path>"
```

Example:

```
$ python demosaic.py "data/in/raptors.jpg" "data/out/demosaic-raptors.png"
```

The command takes a path to the original image as well as the path to where the output will be stored. The console output will look like this as it runs:



```
TERMINAL  PROBLEMS 7  OUTPUT  DEBUG CONSOLE

Saved demosaic-lenna.png
36.90592899957361

C:\Users\drchu\Documents\year3\second_sem\3sk3\project1>python demosaic.py
Opening data/in/raptors.jpg
interpolating Red ...
interpolating Blue ...
interpolating Green ...
Saved data/out/demosaic-raptors.png
72.9425756703777

C:\Users\drchu\Documents\year3\second_sem\3sk3\project1>
Python 3.7.2 32-bit 0 7
```

Figure 4 Terminal output, the MSE is the last line printed (i.e., 72.94...)

Part 3: Results and Conclusion

I then compared the output 3 channels to the original and computed the mean squared error for the following provided images as a benchmark to the values provided on avenue. They ended up being quite comparable and have a relatively high overall error. I also used an additional image of my own as an example which yielded an error of 72. The image was 960 x 540 and is included within the project source code directory. The overall result from the raptors image I tested with showed a lot of distortion in higher detail areas like around the text on the players jersey's etc. There are also some places with artifacts colours, like random spots of red or green, mostly around the edges. The images tended to look "softer" and not as detailed as the original and often a lower MSE meant less artifacts occurring. The lion and lenna images seemed to have better results then the others.



Figure 5 The artifacts are more visible here around the lettering on the jersey.

The artifacts could be a product of the fewer samples in the red and blue channels as well as abrupt changes in colour across the image like for example an edge with 2 different colours.

Image	error
Lenna.png	36.91
kodim05.png	181.28
kodim07.png	64.98
kodim21.png	97.72
kodim23.png	41.99
lion.png	56.86
tree.png	192.14