COMPENG 3SK3 Project 3 Report

Raeed Hassan
hassam41

April 10, 2023

Linear Interpolation

A linear regression based demosaicing algorithm was not implemented for this project, instead this report will demonstrate a linear interpolation algorithm that will use the surrounding pixels to interpolate the red, green, and blue colour components of each pixel based on the surrounding pixels. Depending on the Bayer Pattern applied, the interpolation algorithm will determine which of the surrounding pixels can be used to calculate the three colour components on the pixel. For example, when the Bayer Pattern applied is 'RGGB' and the selected pixel is red, the algorithm will use the pixel's to determine the red component of the image, the average of the pixels on each cardinal direction to determine the green component of the image, and the average of the pixels on each diagonal to determine the blue component of the image. The algorithm does not handle the edges of the image to simplify the coding of the problem.

MATLAB Implementation

The code for the MATLAB implementation of this linear interpolation algorithim is implemented in `project3.m`. The functions implementing linear interpolation are found in `demosaic_interpolation.m`. The interpolation algorithm for a 'RGGB' pattern is shown in Listing 1.

Listing 1: MATLAB Implementation of RGGB Interpolation Algorithm

```
function out_img = demosaic_rggb(mosaic_img)
    % Get the size of the input image
    [height, width] = size(mosaic_img);

    % Create an output RGB image
    out_img = zeros(height, width, 3);

    mosaic_img = double(mosaic_img);

    % Loop over each pixel in the mosaic image
    for i = 1:height-1
        for j = 1:width-1
            % Determine the color of the current pixel based on
                its position in the pattern
            c1 = mod(i, 2);  % 1 = Red, 2 = Green, 3 = Blue
            c2 = mod(j, 2);

            if (i == 1) || (j == 1)
                continue
            end

            % Interpolate the missing color channels based on
               surrounding pixels
```

```
            if c1 == 1 && c2 == 1  % Red
                out_img(i, j, 1) = mosaic_img(i, j);
                out_img(i, j, 2) = (mosaic_img(i, j-1) +
                   mosaic_img(i, j+1) + mosaic_img(i-1, j) +
                   mosaic_img(i+1, j))/4;
                out_img(i, j, 3) = (mosaic_img(i-1, j-1) +
                   mosaic_img(i+1, j-1) + mosaic_img(i-1, j+1)
                   + mosaic_img(i+1, j+1))/4;
            elseif c1 == 0 && c2 == 1  % Green 1
                out_img(i, j, 1) = (mosaic_img(i+1, j) +
                   mosaic_img(i-1, j))/2;
                out_img(i, j, 2) = mosaic_img(i, j);
                out_img(i, j, 3) = (mosaic_img(i, j-1) +
                   mosaic_img(i, j+1))/2;
            elseif c1 == 1 && c2 == 0  % Green 2
                out_img(i, j, 1) = (mosaic_img(i, j-1) +
                   mosaic_img(i, j+1))/2;
                out_img(i, j, 2) = mosaic_img(i, j);
                out_img(i, j, 3) = (mosaic_img(i-1, j) +
                   mosaic_img(i+1, j))/2;
            else  % Blue
                out_img(i, j, 1) = (mosaic_img(i-1, j-1) +
                   mosaic_img(i+1, j-1) + mosaic_img(i-1, j+1)
                   + mosaic_img(i+1, j+1))/4;
                out_img(i, j, 2) = (mosaic_img(i, j-1) +
                   mosaic_img(i, j+1) + mosaic_img(i-1, j) +
                   mosaic_img(i+1, j))/4;
                out_img(i, j, 3) = mosaic_img(i, j);
            end
        end
    end

    out_img = uint8(out_img);
end
```

Visual Comparison

A visual comparison of `test1.png` can be seen in Figure 1. We can compare the MATLAB `demosaic` algorithm in Figure 1b and the linear interpolation algorithm in Figure 1c. We can observe that these two algorithms have similar performance, displaying similar colours and showing the same artifacts around edges where there are large changes in colour in the image.

A comparison of `test2.png` can also be seen in Figure 2. We similarly observe a similar

performance between the MATLAB implementation of `demosaic` and the linear interpolation algorithm.

Drawbacks of Linear Interpolation

The linear interpolation demosaicing algorithm results in Figures 1c and 2c demonstrate the flaws and drawbacks of such an interpolation algorithm. Due to the lack of information available for making an interpolation for any given pixel, we can observe that when there should be a large shift in colour between pixels it will often resulting in artifacting around edges of objects in the scene. This artifacting can also result in the image appearing more blocky than it should given its resolution.

One potential way to tackle the drawbacks of such an approach is to simply increase the amount of information available to the algorithm. For example, if each pixel accounted for the next several pixels around it when performing the interpolation, it would likely handle edges of objects better. Similarly, a linear regression algorithm that solves the linear least square problem could appropriately assign weights to each surrounding pixels contribution to more accurately calculate a missing colour component past simply taking the average.

(a) Test Image

(b) MATLAB `demosaic` Output

(c) Linear Interpolation Demosaicing Output

Figure 1: Test Image `test1.png`

(a) Test Image

(b) MATLAB `demosaic` Output

(c) Linear Interpolation Demosaicing Output

Figure 2: Test Image `test2.png`