SGN-16006, IMAGE PROCESSING ASSIGNMENT

Jonas Nikula, Vili Saura

240497, 240264
jonas.nikula@student.tut.fi,vili.saura@student.tut.fi

1. INTRODUCTION

In this assignment we implemented three different interpolation methods to create a RGB image from raw Bayer Data. After that, we measured the accuracy of the interpolation methods with Mean Squared Error and Mean Absolute Error, by comparing the image created by our methods to the original, correct image. The interpolation methods used in this assignment are nearest neighbor interpolation, bilinear interpolation and patterned pixel grouping interpolation.

2. BACKGROUND THEORY

Digital (and analog) camera sensors can't capture different colors directly. The way to capture a color picture is to use differently colored filters, which each let through certain wavelengths of light. Usually the wavelengths corresponding to the primary colors red, green and blue are used. The first color pictures were captured by taking three different pictures, each with a different filter. This is of course slower than taking only one picture, so digital cameras use a Bayer Filter [1].

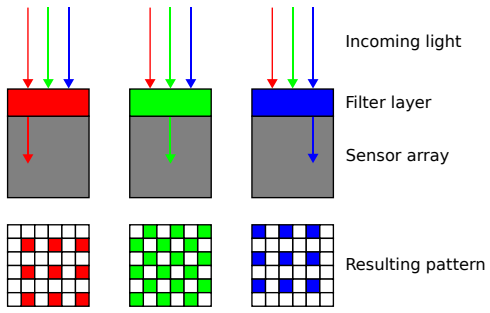

Fig. 1: A concept illustration of the Bayer filter [1]

As seen in figure 1, the problem with the Bayer filter is that every pixel only has one color. Every square of four pixels has 1 red, 1 blue and 2 green pixels. To get a proper picture out this data, the missing two color channels have to be interpolated for each pixel. There are different interpolation methods, three of which are implemented in this lab assignment.

3. METHODS

3.1. Nearest neighbor interpolation

In nearest neighbor interpolation, we process the raw Bayer data in squares of 4 pixels, which contain 1 red, 1 blue and 2 green pixels. For the blue channel, all 4 pixels are set to the color value of the 1 blue pixel. The same is done for the red channel. For the green channel, the pixels without a green color value get their value from the green pixel horizontally next to them.

The main benefits of NN-interpolation are its speed and simplicity.

3.2. Bilinear interpolation

In bilinear interpolation the color value of the pixels is calculated from the color value of neighboring pixels with same color i.e sum the values of neighboring pixels and divide it by the amount of neighboring pixels. However, the pixels at the edges of the picture don't always have neighboring pixels with color value or don't have enough of them. To correct this we can just ignore the border pixels, resulting in smaller image, or as we did in the assignment and pad the color arrays with zeros. The color values of the layers are in different pixels, so every layer needs an unique function to calculate the value of an unknown pixel. Green has the most known color values, and every unknown pixel borders four known pixels, so one function can calculate all values of the green layer.

Only every other row of red and green layers has colorvalues, which has to be taken into account when designing these functions. The rows which have no colorvalues compute the value differently, depending on how many known values border the pixel, and use either four or two values.

Colorvalues of the pixels are calculated from the padded color array and added to the unpadded one. After all values of all colorarrays are calculated, the arrays are combined into RGB image.

3.3. Patterned pixel grouping interpolation

The Patterned Pixel Grouping, or PPG interpolation is a significantly more advanced method of interpolation. We implemented the method using the description found here[2].

This method basically tries to take into account the color gradients found in the data. It also takes advantage of the greater amount of green data and uses it to assess the brightness gradients of the image, which is a feature that the human visual system perceives more accurately than changes in color. Figure 2 has an example of Bayer data, which we'll refer to when explaining the method.

Fig. 2: The example portion of the data used in patterned pixel grouping [2]

First the method calculates a value for all the missing green pixels. It calculates the values by first assessing 4 gradients, in the cardinal directions, using two neighboring green pixels and the red/blue pixel in place of the missing green pixel and the red/blue pixel in the cardinal direction. For example, the "northern" gradient for pixel 13 in 2 is calculated using formula 1

$$\Delta N = |R_{13} - R_3| \times 2 + |G_8 - G_{18}| \quad (1)$$

The pixels used in the calculation of the smallest gradient are then used to calculate the actual green color value for the pixel, as shown in equation 2, which calculates the value in the case of the northern gradient being smallest.

$$G_{13} = (G_8 \times 3 + G_{18} + R_{13} - R_3)/4 \quad (2)$$

After the green values are calculated for every pixel, the missing red and blue values are calculated. The instructions [2] use an auxiliary function called *hue_transit*, defined as:

```
function hue_transit (l1, l2, l3, v1, v3) =
  if (l1 < l2 and l2 < l3) or (l1 > l2 and l2 > l
  then v1 + (v3  v1)  (l2  l1) / (l3  l1)
  else (v1 + v3) / 2 + (l2  2  l1  l3) / 4
```

The red and blue pixels that are situated on top of only green pixels are calculated using only this function. For example, the blue value of pixel 8 in the data 2 is calculated with equation 3

$$B_8 = hue\_transit(G_7, G_8, G_9, B_7, B_9) \quad (3)$$

For pixels where there's already a red or blue value, the method is a bit more complicated. First, we again calculate gradients, this time however only two: A north-east to south-west gradient and a north-west to south-east gradient. For example, the NE to SE gradient for the blue value of pixel 13 is calculated with formula 4

$$\Delta NE = |B_9 - B_{17}| + |R_5 - R_{13}| + |R_{13} - R_{21}| + |G_9 - G_{13}| + |G_{13} - G_{17}| \quad (4)$$

The gradients are then compared again, and for example if the ΔNE gradient is smaller, equation 5 gives the blue value for pixel 13.

$$B_{13} = hue\_transit(G_9, G_{13}, G_{17}, B_9, B_{17}) \quad (5)$$

4. RESULTS

The figs. 3 to 5 contain two images that are obtained by processing the raw Bayer data with our implemented interpolation methods.

(a) image 2 (b) image 5

Fig. 3: A zoomed in area of two NN-interpolation processed images

(a) image 2 (b) image 5

Fig. 4: A zoomed in area of two bilinear interpolation processed images

(a) image 2 (b) image 5

Fig. 5: A zoomed in area of two patterned pixel grouping interpolation processed images

The table 1 contains the Mean Squared Error, Mean Absolute Error and time it took to run the interpolation function. The error rates were obtained by processing the raw data with our implemented functions, and comparing that result to a TIFF encoded file of the same data. The computation times were obtained using Matlab's tic-toc timetaking function. They'll depend on the computer used, but what can be objectively compared are their relative durations.

Table 1: Results from the different methods

	MSE	MAE	Computing time
NN-interpolation	90.403	4.6084	0.0377 s
Bilinear interpolation	46.72	3.1406	0.1347 s
PPG interpolation	27.3351	2.7879	9.8308 s

5. CONCLUSIONS

Comparing the images from Fig 3 to Fig 5, we can see that pixe grouping creates a more natural and less blocky image than NN-interpolation, with bilinear interpolation beign somewhere in between. Studying the results in table 1 we can see, that while PPG interpolation has the lowest error scores of the three interpolation methods, it takes substantially more time to process the image. For contrast, NN-interpolation and bilinear interpolation are faster, but not as accurate. Therefore, processing Bayer filtered pictures using one of these methods is a tradeoff between accuracy and speed. When the amount of pictures to be processed increases, one might choose bilinear interpolation over PPG interpolation if speed is at the essence.

REFERENCES

[1] "Wikipedia: Bayer filter," https://en.wikipedia.org/wiki/Bayer_filter, accessed: 2017-03-05.

[2] C.-K. Lin, "Patterned pixel grouping algorithm," https://sites.google.com/site/chklin/demosaic/, accessed: 2017-03-05.