CMP2090M

Object-Oriented Programming Assignment Report

Cameron Khan

Student ID: KHA16607446

1. Introduction

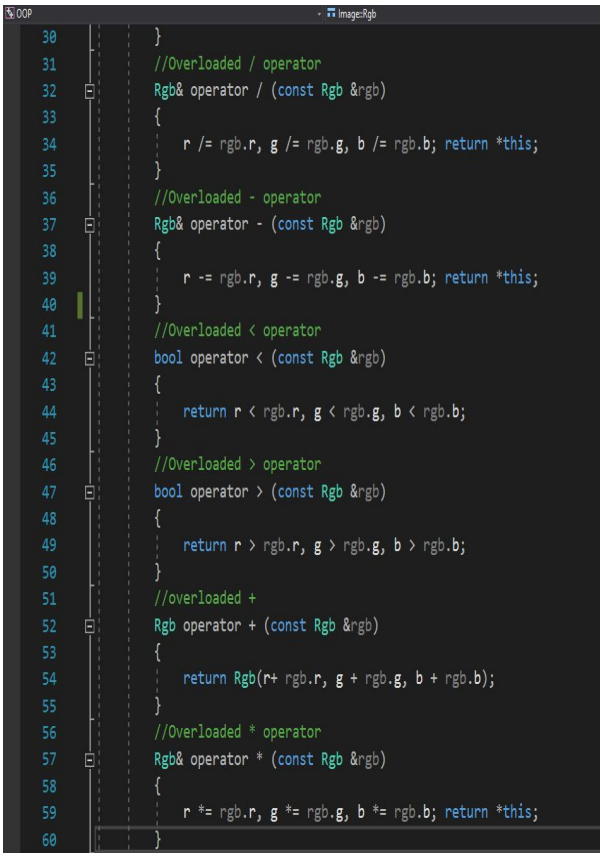
This is the supporting report of the Object Oriented Programming Assignment. The given assignment was to use different means and forms of object orientation, to create image blending algorithms and an algorithm for zooming images via nearest neighbour. The report will go describe, explain and show there outcomes of the program. To see larger versions of the results please visit the imgur links.

2. Programme Structure

The programme has several classes to ensure the code is neat and to show understanding of how to use classes correctly, (I use referencing for my images) these classes are;

. The ReadWrite class, this class is where the methods to read in images and write out images is held, these methods are then used in the main by calling the class. It also holds a method to print out a log file.

. The Image class, this class holds all the operation overloads the rest of my code and classes need to complete arithmetics and are used in several of the algorithms. The class is also used to create a struct named Rgb which allows you to hold 3 values in 1 variable. The overloads I wrote can be seen to the right.



. The Blend class, this holds the several blending algorithms as member functions, this is so they can be used and called globally in any cpp file and so on.

. And finally the Zoom class which holds the method I use to zoom an image using nearest neighbour.

The only thing in my main is the switch menu users will use to access these classes.

3. Algorithms

Mean Blend; I add all the values of each value (r,g,b) and then divide it by the amount of value's you have added). I created a stacker that adds all the individual 3 values of each pixel together. As all the r, g, b values have already been added the only thing that needs to be done in my Mean Algorithm is to divide it by the amount of value's, here that value is 10.

```
void Blend::imgMeanBlend(Image &imgholder)
{
    for (int x = 0; x < imgholder.h*imgholder.w; x++)
    {
        imgholder.pixels[x].r /= 10;
        imgholder.pixels[x].g /= 10;
        imgholder.pixels[x].b /= 10;
    }
}
```

Median Blend; The median blend is a little more complicated in the terms of needing a vector that holds an array fo the image's and slightly more arithmetics need to be applied. This algorithm needs to sort the pixels values into ascending order, this allows us to complete the next bit of the algorithm which is finding the middle value. Because I am working on an even array (10) I need to take the 2 centre values and find there mean (1st center value + 2nd centre value) /2 = median.

```
21  void Blend::imgMedianBlend(std::vector<Image> &imgArray, Image &imgeop)
22  {
23
24      std::vector<float>r;
25      std::vector<float>g;
26      std::vector<float>b;
27
28      r.resize(10);
29      g.resize(10);
30      b.resize(10);
31
32      for (int p = 0; p < imgArray[0].h*imgArray[0].w; p++)
33      {
34          for (int i = 0; i < 10; i++)
35          {
36              r[i] = imgArray[i].pixels[p].r;
37              g[i] = imgArray[i].pixels[p].g;
38              b[i] = imgArray[i].pixels[p].b;
39          }
40
41          std::sort(r.begin(), r.end());
42          std::sort(g.begin(), g.end());
43          std::sort(b.begin(), b.end());
44
45          imgeop[p].r = (r[4] + r[5]) / 2;
46          imgeop[p].g = (g[4] + g[5]) / 2;
47          imgeop[p].b = (b[4] + b[5]) / 2;
48      }
```

Sigma clip and standard deviation;
To complete sigma clipping I first need to find the standard deviation of the array of image's (I find the mean of each pixel in "imgeArray", then for each value I subtract the mean and square its result, I then find the mean of said roots differences). I then perform my sigma clipping algorithm, I define a higher and lower bound and then reject (remove) any values that are outside these bound's the remaining pixels become whats ever left in the "imgArray" I then find the mean(average) of the remaining pixels.

```
void Blend::imgSigmaClip(std::vector<Image> &imgArray, Image &imgeop, Image &imgSigma)
{
    Image::Rgb sigma = 1;
    Image::Rgb sdt2;
    Image::Rgb remainingPixels;
    Image::Rgb remainingMean;
    sdt2 = sdt(imgArray);
    for (int j = 0; j < imgArray.size(); j++)
    {
        float numberOfPixels = 10;
        for (int i = 0; i < imgeop.h*imgeop.w; i++)
        {
            //Initialize the upper and lower bounds
            Image::Rgb Higher(imgeop.pixels[i] + sigma * sdt2);
            Image::Rgb Lower(imgeop.pixels[i] - sigma * sdt2);

                if (imgArray[j].pixels[i] > Higher || imgArray[j].pixels[i] < Lower)
                {
                    //Reject the pixel
                    //Decrement the number of pixels each time a pixel is rejected
                    numberOfPixels--;
                }
                else
                {
                    remainingPixels += imgArray[j].pixels[i];
                }


            Image::Rgb remainingMean = remainingPixels / 2;

            imgSigma.pixels[j] = remainingMean;
        }
    }
```

2X Scale Nearest Neighbour; I create a new empty image which is the x2 the resolution (size) of the image provided for scaling, we then supply 2 scaling constraints x_ratio and y_ratio. We then loop through all the pixels in the output image, using the source pixels to copy from by scaling our control variables by x_ratio and y_ratio

```
void Zoom::x2Scale(Image &toBeZoomed, Image &zoomedImge)
{
    int width = toBeZoomed.w;
    int height = toBeZoomed.h;
    int w2 = zoomedImge.w;
    int h2 = zoomedImge.h;

    double x_ratio = w2 / width;
    double y_ratio = h2 / height;
    double px, py;
    for (int i = 0; i < h2; i++)
    {
        for (int j = 0; j < w2; j++)
        {
            px = (double)((int)(j / x_ratio));
            py = (double)((int)(i / y_ratio));
            zoomedImge.pixels[(i * w2) + j].r = toBeZoomed.pixels[(int)((py * width) + px)].r;
            zoomedImge.pixels[(i * w2) + j].g = toBeZoomed.pixels[(int)((py * width) + px)].g;
            zoomedImge.pixels[(i * w2) + j].b = toBeZoomed.pixels[(int)((py * width) + px)].b;
        }
    }
```

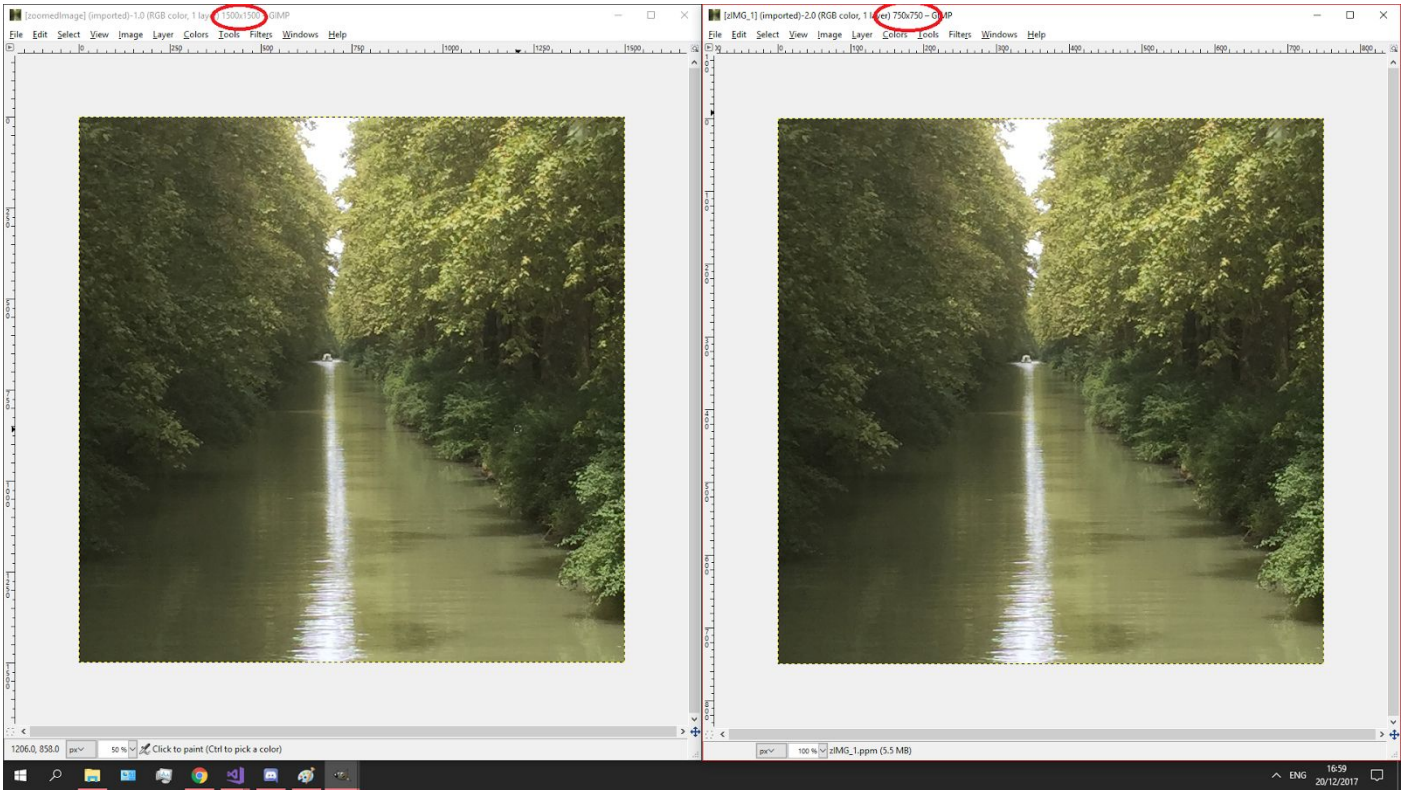
5. Result's;

Mean; The mean blending method as you can see in the image does not manage to remove all of the noise in the set of 10 images evident by the faded truck in the background. In full few of the image you can also notice a blurred and faded human walking across the grass (I cannot fit a full sized image on here). https://imgur.com/DSm7pwl

Median; As you can see compared to the mean image more noise has been reduced as the truck is no longer there, nor are the people walking across the grass . https://imgur.com/bSoemow

Sigma Clap; My sigma clip out come is just a black image, I explain in the discussion below why I believe so.

Zoom; Please check the Imgur link to see the differences between the two image's as I've tried to highlight the new rescaled image is now 1500x1500 (on the left) and the original image is 700x700 (right) https://imgur.com/LdAkqqf



5. Discussion & Conclusion

Although I attempted sigma blending it seems I cannot get it to work its output is just a black image, I believe it is because I am lacking an iterator for my "imgArray" however I did not have time to write this. I also attempted to create a function that will output the image information to a log file however I could not get this working either (see attempt below).

```
/*void ReadWrite::imgInfo()
{
    cout << "Writing LogFile" << endl;
    // assigns where to save the file
    ofstream Output;
    Output.open("LogFile.txt");
    // prints out the header infomation from the .ppm file
    Output << "Image statistics for: " << fileName << "\n";
    Output << "Header: " << h << "\n";
    Output << "Width: " << w << "\n";
    Output << "Height: " << h << "\n";
    Output << "Size: " << size << "\n";
    Output << "Max value: " << maxValue << "\n";

    cout << "File has been written" << endl;
}*/
```

References

Gralak, R. (2008). *Advanced Image Combine Techniques*. [ebook] pp.17-18. Available at: http://www.ccdware.com/support/presentations/advanced_image_combine_gralak20081116.p [Accessed 13 Dec. 2017].