BME Basics of Programming II

FINAL PROJECT REPORT

BITMAP IMAGE TREATMENT USING C++

By Laporte Nathan Claude

SUMMARY :

I) Introduction

The BMP (Bitmap) file format was created as a unified open standard for image files developed by Microsoft and IBM. It is one of the easiest image format to use and develop, and its structure is well described on the internet. There are different types of format, depending on the color depth (1bit – 8 colors -, 4bits – 16 colors -, 8 bits – 256 colors - , 16 bits – 65536 colors -, and 24 bits – 16.8M colors-) The latest tend to be the most used nowadays, since the size of and image is not so relevant. However, BMP files are usually heavy, because they rarely use compression, thus do not alter the image. It is as such not common on the internet, where compressed formats are favored.

It is composed of two headers that always have the same structures, followed by a pixel matrix (bitmap), as such it is really easy to understand how it works.

I chose to focus on the 24bits BMP format because it is very intuitive, and offers a lot of different interactions with the image. It is also very easy to convert an image to BMP using paint, making the software able to treat almost any image.

We can also note that image treatment is now one of the great opportunity, since almost all people use digital photography, and good image treatment software are scarce, and expensive (EG: Photoshop).

II) Problems faced and solutions.

When developing the program the first goal was to be able to read and copy an image. The very first version did not use any OOP and relied on fopen, fwrite… Which is not very optimized. I started to think about a way to add the Object Oriented paradigm, and created a class. An image was now treated as an object, having its own methods, like writing a new file, or changing the image. I reorganized the program to use strict c++ with the "file" type and the fstream library.

One of the problem faced was that the bitmap was stored in a vector, which is one dimension, making some filters trickier to implement, especially the mosaic filter. The K-mean filter did not make it to the final version because of this. There was concerns because of the type used to store the matrix: in order to reduce the size, each pixel was stored in a char (8bits), leading to conversions from char to int for some operations. I had several cases of "wrong colors" and had to used unsigned char to have the (0 -255) possibilities.

In order to add inheritance, I chose to add the support for other bitmap files, with different color depths, since for these files, we need to add the color palette, which acts like an index of all colors used in the image. But the filters had to be redefined, and almost all of them produce odd results, because the palette works in a way that I still don't understand. It was also problematic that when an image object had to be created, (the file was read) in order to determine the color depth, and thus to know if the child or the mother class should be used, which sounds inefficient.

Finally, I was asked to implement the filters in a parent children inheritance pattern, which I tried to do since, but it is more complicated than I thought. My attempt to implement such pattern can be found in the annexes.

III) Conclusion.

This program made me understand the importance of the design, because unfortunately when you follow one path, it becomes really hard to change, for instance, in my case implementing the parent – child inheritance was really tricky and required me to rethink the whole program. It was also very instructive, I had to search the internet for the small problems I could have. Overall I am really happy with the result, as the filters work well for 24bits BMP files. I might use this program afterwards for myself.

IV) Sources

https://en.wikipedia.org/wiki/BMP_file_format

https://web.archive.org/web/20080912171714/http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html

http://stackoverflow.com/questions/8836017/reading-a-bmp-file-in-c

http://stackoverflow.com/questions/9448478/what-is-wrong-with-this-sepia-tone-conversion-algorithm/9448635#9448635

http://www.cplusplus.com/reference/fstream/ifstream/

http://www.cplusplus.com/reference/vector/vector/?kw=vector

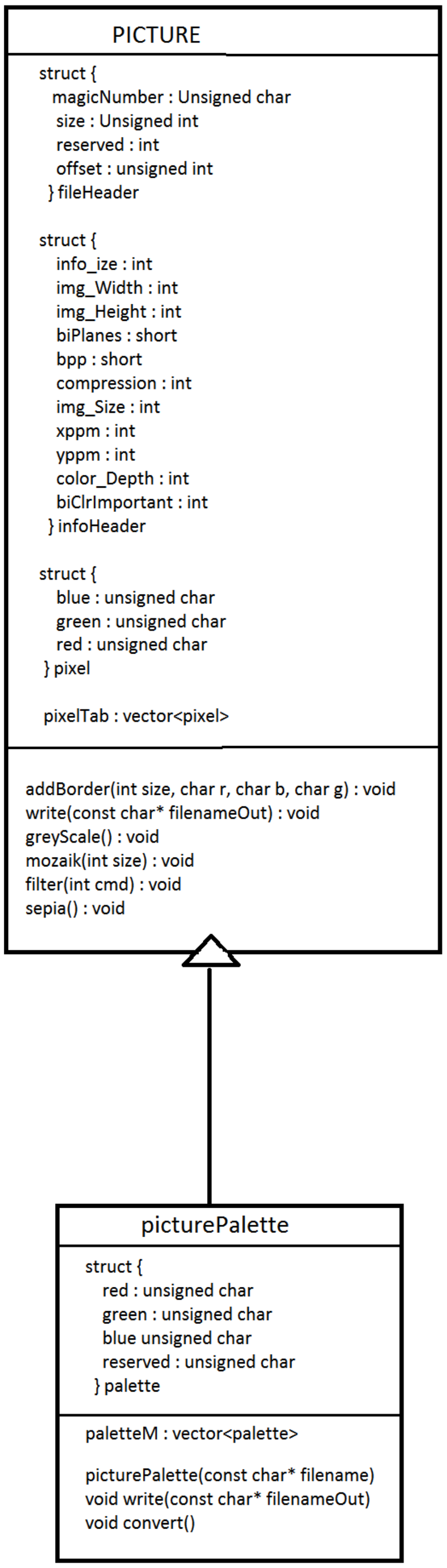
Link to the github with the program :

https://github.com/Sysmetryx/PROJETCPP

LAPORTE Nathan Claude
Neptun Code : IBIS2E

Simple inheritance for different types of BMP formats : With and without the palete.

UML Class diagram :



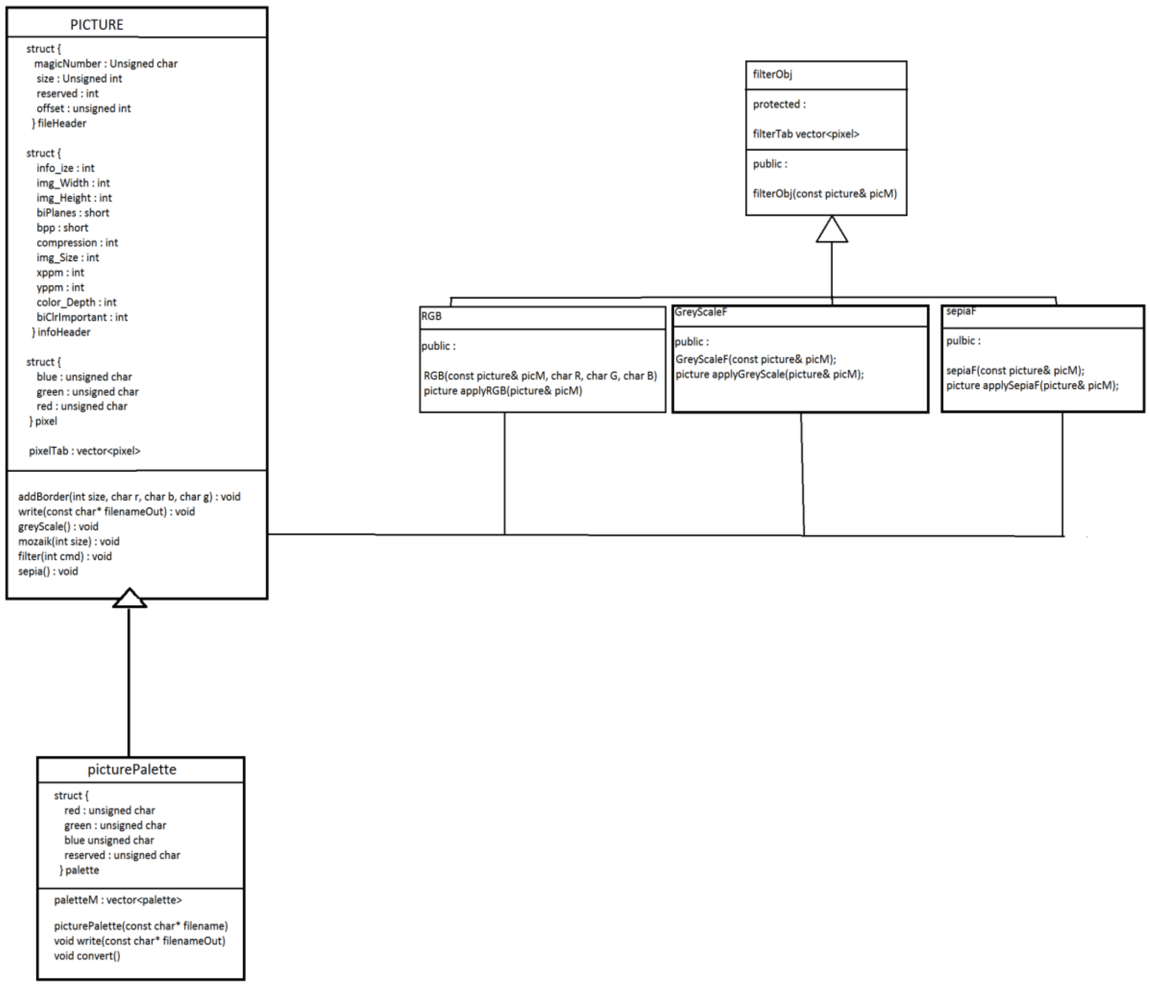
The initial UML Class diagram.

LAPORTE Nathan Claude
Neptun Code : IBIS2E

The UML Class diagram using Parent children inheritance.

Simple inheritance for different types of BMP formats : With and without the palete.

UML Class diagram :



V) Code Sample :

Here are some lines added to the original code, specifically, the parent child inheritance. These code portions do not include the use of the functions in the program. :

```
class filterObj {

protected:

 vector<pixel> filterTab;

public:

  filterObj(const picture& picM);

};


class RGB : public filterObj

{

public:

  RGB(const picture& picM, char R, char G, char B);

  picture applyRGB(picture& picM);

};


class GreyScaleF : public filterObj

{

public:

    GreyScaleF(const picture& picM);

    picture applyGreyScale(picture& picM);

};


class sepiaF : public  filterObj

{

  public:

    sepiaF(const picture& picM);

    picture applySepiaF(picture& picM);

};
```

```
filterObj::filterObj(const picture& picM)
{
    pixel tempor;
    tempor.red = 0;
    tempor.green = 0;
    tempor.blue = 0;
    for(int i =0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
    {
        filterTab.push_back(tempor);
    }
}

RGB::RGB(const picture& picM, char R, char G, char B) : filterObj(picM)
{
    pixel temporary;
    temporary.red = R;
    temporary.green = G;
    temporary.blue = B;
    for (int i = 0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
    {

        filterTab.push_back(temporary);
    }
}
picture RGB::applyRGB(picture& picM)
{
    for (int i = 0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
    {
         if (picM.pixelTab[i].red - filterTab[i].red < 0)
```

```
            {
               picM.pixelTab[i].red = 0;
            }
            else if (filterTab[i].red < picM.pixelTab[i].red)
            {
               picM.pixelTab[i].red =  picM.pixelTab[i].red - filterTab[i].red;
            }
            if (picM.pixelTab[i].blue - filterTab[i].blue < 0)
            {
               picM.pixelTab[i].blue = 0;
            }
            else if (filterTab[i].blue < picM.pixelTab[i].blue)
            {
               picM.pixelTab[i].blue =  picM.pixelTab[i].blue - filterTab[i].blue;
            }
            if (picM.pixelTab[i].green - filterTab[i].green < 0)
            {
               picM.pixelTab[i].green = 0;
            }
            else if (filterTab[i].green < picM.pixelTab[i].green)
            {
               picM.pixelTab[i].green =  picM.pixelTab[i].green - filterTab[i].green;
            }
   }
   return picM;

}


GreyScaleF::GreyScaleF(const picture& picM) : filterObj(picM)
{
   for(int i =0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
   {
       filterTab.push_back(picM.pixelTab[i]);
   }
}


picture applyGreyScaleF(picture& picM)
{
   float rweight2 = 0.2105;
   float gweight2 = 0.7152;
   float bweight2 = 0.0722;
   for(int i =0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
   {
       char btemp2 = picM.pixelTab[i].blue;
       char gtemp2 = picM.pixelTab[i].green;
       char rtemp2 = picM.pixelTab[i].red;
       picM.pixelTab[i].blue = floor(bweight2*btemp2) + floor(rweight2*rtemp2) +
floor(gweight2*gtemp2);
       picM.pixelTab[i].red = floor(bweight2*btemp2) + floor(rweight2*rtemp2) +
floor(gweight2*gtemp2);
       picM.pixelTab[i].green = floor(bweight2*btemp2) + floor(rweight2*rtemp2) +
floor(gweight2*gtemp2);
   }
   return picM;
}



sepiaF::sepiaF(const picture& picM) : filterObj(picM)
{
   for(int i =0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
   {
       filterTab.push_back(picM.pixelTab[i]);
   }
}


picture sepiaF::applySepiaF(picture& picM)
{
   for(int i =0; i < picM.infoHeader.img_Width*picM.infoHeader.img_Height; i++)
   {
       unsigned char rtemp2 = picM.pixelTab[i].red;
       unsigned char gtemp2 = picM.pixelTab[i].green;
       unsigned char btemp2 = picM.pixelTab[i].blue;
       if((rtemp2 * .393) + (gtemp2 *.769) + (btemp2 * .189) > 255)
       {
           picM.pixelTab[i].red = 255;
       }
       else
       {
           picM.pixelTab[i].red = (rtemp2 * .393) + (gtemp2 *.769) + (btemp2 * .189);
       }


       if((rtemp2 * .349) + (gtemp2 *.686) + (btemp2 * .168) > 255)
       {
           picM.pixelTab[i].green = 255;
       }
       else
       {
           picM.pixelTab[i].green = (rtemp2 * .349) + (gtemp2 *.686) + (btemp2 * .168);
       }
       if((rtemp2 * .272) + (gtemp2 *.534) + (btemp2 * .131) > 255)
       {
           picM.pixelTab[i].blue = 255;
       }
       else
       {
           picM.pixelTab[i].blue = (rtemp2 * .272) + (gtemp2 *.534) + (btemp2 * .131);
       }
   }
}
```

The last version of the program can be found on the github. (link in the sources).