浙江大学 实验报告

戴毅阳（3200104915）

课程名称：图像信息处理　指导老师：宋明黎

实验名称：bmp文件读写及RGB和YUV颜色空间的转化

一、实验目的和要求

1. 目的：学习并掌握bmp文件的格式；了解RGB与YUV颜色空间之间的转换方式；能够将bmp图片文件在灰度和彩色之间转换
2. 要求：

   （1）读入一个bmp文件的信息

   （2）实现颜色空间从RGB到YUV的转化

   （3）将颜色转换为灰度图像（gray = Y -> YUV）

   （4）调整灰度值，线性分布于[0, 255]

   （5）改变亮度值Y

   （6）实现颜色空间从YUV到RGB的转化

   （7）将图片重新转换为彩色图像

二、实验内容和原理

1. 读写bmp文件： 该内容设计到bmp文件（主要为24位图）内容，因此需要对于bmp文件的内容和结构有比较深入的了解

   1. bmp文件头(bmp file header):14 bytes
   2. 位图信息头(bitmap information):40 bytes
   3. 调色板(color palette):chosen
   4. 位图数据(image data)

   下方即为bmp图像对应结构在C语言中的储存形式：

```
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef int LONG;

struct BITMAPFILEHEADER{
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved1, bfReserved2;
    DWORD bfOffBits;
};

struct BITMAPINFOHEADER{
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter, biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
};

struct RGBQUAD{
    BYTE rgbBlue;
    BYTE rgbRed;
    BYTE rgbGreen;
    BYTE rgbReserved;
};
```

2. 颜色空间RGB和YUV之间转换： $$\begin{bmatrix} Y\ U\ V \end{bmatrix} = \begin{bmatrix} 0.299&0.587&0.114\ -0.147&0.289&0.435\ 0.615&-0.515&-0.100 \end{bmatrix} \begin{bmatrix} R\ G\ B \end{bmatrix} $$

由于RGB和YUV颜色空间存在上述转换关系，因此在对图片进行颜色空间转换的操作时，我们只需要从bmp文件中读出对应的RGB或YUV数据，运用上述矩阵，求解另一空间的对应值

$$\begin{cases} Y = 0.299\times R + 0.587\times G + 0.114\times B \ Pb =-0.169\times R - 0.331 \times G + 0.500\times B \ Pr = 0.500\times R - 0.439\times G - 0.081\times B \end{cases}$$

三、实验步骤及分析

1.读写bmp文件

下方即为读入bmp文件内容时的代码:

```
    FILE *fp1;
    string bmpname;
```

```
    cout << "Please enter the name of the bmp file(without .bmp) you want
to operate: " << endl;
    cin >> bmpname;
    bmpname += ".bmp";

    fp1 = fopen(bmpname.c_str(), "rb");
    if(!fp1) cout << "fail to open " << bmpname << endl;
    else{
        // Here, the value of variable bfType must be read in before seek
the fp
        // in the whole file as there is something special about the
sizeof().
        WORD bfType;
        fread(&bfType, 1, sizeof(WORD), fp1);
        if(bfType != 0x4D42){
            cout<<"It's not a bmp!"<<endl;
            return 0;
        }

        fread(&bmpHead, sizeof(BITMAPFILEHEADER), 1, fp1);
        fread(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, fp1);
        // cout << bmpInfo.biBitCount << endl;
        if(bmpInfo.biBitCount == 8) fread(plate, sizeof(RGBQUAD), 256,
fp1);
        // As the main type of the bmp nowadays is 24 bytes, the
colorplate is not essential.
        fseek(fp1, bmpHead.bfOffBits, 0);

        int size = bmpInfo.biSizeImage / 3;
        RGB *bmpdata = new RGB[size];
        fread(bmpdata, 1, sizeof(RGB)*size, fp1);//读取bmp数据信息
        fclose(fp1);
```

需要注意的是，此处并未像bmp的BITMAPFILEHEADER中存入表示文件类型的WORD bfType，由于我在完成代码过程中，一次性读入BITMAPFILEHEADER bmpHead（如果包括bfType）时，则一定会报错。在我查询一些资料后发现，由于C/C++中sizeof()的特殊性，它无法准确计算结构体内成员的size_t返回值（大小/字节），则为了后续处理方便，单独讲WORD bfType列出，单独从文件读入，也当作文件打开正确的第一步判定。

2.RGB to YUV

上一步已经成功读入bmp文件的信息，随后只需要根据二中提出的对应变换矩阵，即可得出对应在YUV颜色空间下的图像数据

```
    BYTE red, blue, green;
    for(int i=0; i<size; i++){
        red = bmpdata[i].R;
        blue = bmpdata[i].B;
        green = bmpdata[i].G;
        res[i].Y = 16 + 0.257*red + 0.504*green + 0.098*blue;
        res[i].U = 128 - 0.148*red - 0.291*green + 0.439*blue;
```

```
            res[i].V = 128 + 0.439*red - 0.368*green - 0.071*blue;
        }
```

此处需要说明的是，在网上查阅资料时，发现RGB和YUV之间转换是有不同的公式的，因此在尝试多种公式后选择了数字YUV与数字RGB之间的转换BT601。
(https://blog.csdn.net/weixin_33816946/article/details/94712493)

3.将图像转为灰度图像

此处我才用了两种方式

一是直接讲RGB三种通道的值直接用YUV空间的Y值覆盖，最终较好地实现图像类型的转换：

```
        RGB *gray = new RGB[size];
        for(int i=0; i<size; i++){
            gray[i].B = res[i].Y;
            gray[i].G = res[i].Y;
            gray[i].R = res[i].Y;
            // cout << (int)res[i].Y << endl;
        }
        FILE *fp2;
        fp2 = fopen("gray.bmp", "wb");
        if(!fp2) cout << "error" << endl;
        fwrite(&bfType, sizeof(WORD), 1, fp2);
        fwrite(&bmpHead, sizeof(BITMAPFILEHEADER), 1, fp2);
        fwrite(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, fp2);
        fwrite(gray, sizeof(RGB)*size, 1, fp2);
        fclose(fp2);
        delete[] gray;
```

二是通过一步步修改bmpdata中的值来实现

```
            LONG LineByte = (bmpInfo.biWidth*8/8 + 3)/4*4;
            FILE *fp4;
            BYTE *newdata = new BYTE[size];
            fp4 = fopen("another_gray.bmp", "wb");
            fread(&bfType, 1, sizeof(WORD), fp4);
            fread(&bmpHead, sizeof(BITMAPFILEHEADER), 1, fp4);
            fread(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, fp4);
            fread(newdata, size, 1, fp4);
            if(!fp4) cout << "fail to create" << endl;
            else{
                bmpHead.bfSize =
    14+40+sizeof(RGBQUAD)*256+LineByte*bmpInfo.biHeight;
                bmpHead.bfOffBits = 14+40+sizeof(RGBQUAD)*256;
                bmpInfo.biBitCount = 8;
                bmpInfo.biSizeImage = LineByte*bmpInfo.biHeight;
                fwrite(&bfType, sizeof(WORD), 1, fp4);
                fwrite(&bmpHead, sizeof(BITMAPFILEHEADER), 1, fp4);
```

```
            fwrite(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, fp4);
            for(int i=0; i<256; i++){
                plate[i].rgbRed = i;
                plate[i].rgbBlue = i;
                plate[i].rgbGreen = i;
            }
            fwrite(plate, sizeof(RGBQUAD), 256, fp4);
            int sum = 0, cnt = 0;
            fwrite(newdata, LineByte*bmpInfo.biHeight, 1, fp4);
        }

        fclose(fp4);
```

但需要说明的是，由于对于其中RGB数据储存的不确定，以及对于调色板RGBQUAD的不熟悉，该实现的效果并不成功，在有些图像处理上会出现错位，模糊等情况。

4.YUV to RGB 其过程与2中的及其类似，此处便不再赘述其细节，直接附上对应代码：

```
        FILE *fp3;
        fp3 = fopen("YUV_to_RGB.bmp", "wb");
        RGB *recover_rgb = new RGB[size];
        if(!fp3) cout << "wrong" << endl;
        fwrite(&bfType, sizeof(WORD), 1, fp3);
        fwrite(&bmpHead, sizeof(BITMAPFILEHEADER), 1, fp3);
        fwrite(&bmpInfo, sizeof(BITMAPINFOHEADER), 1, fp3);
        // There seems to be a lot of standards to transform the RGB to
YUV, here I use the
        // standard (BT601) --> 数字YUV <--> 数字RGB
        for(int i=0;i<size;i++){
            int Y = res[i].Y - 16;
            int U = res[i].U - 128;
            int V = res[i].V - 128;
            // Here I try to use bit operation to simplify and accelerate
the performing speed.
            recover_rgb[i].R = overflow((298*Y + 409*V + 128) >> 8);
            recover_rgb[i].G = overflow((298*Y - 100*U - 208*V + 128) >>
8);
            recover_rgb[i].B = overflow((298*Y + 516*U + 128) >> 8);
        }
        fwrite(recover_rgb, sizeof(RGB)*size, 1, fp3);
        fclose(fp3);
```

四、运行环境及方法

1. 编译器： clang++
2. 编辑器： vscode
3. 操作系统：macOS
4. 运行方法：已写makefile

```
make
make run
...(输入文件名即可)
make clean
```

五、成果展示

由于.bmp文件无法内附于markdown文件中，因此测试结果将以附件的形式上传 需要说明的是，其中2.bmp处理后对于黑白颜色会有错误（尚不清楚原因）

六、心得体会

完成此次assignment1让我对于bmp的文件结构有了更为深刻的认识，原来一张图片的内部结构是这样复杂而又有其规律性；在对于YUV和RGB颜色空间转换中，让我认识到了，数学对于图像处理中的作用。