专业：计算机科学与技术
姓名：林海滨
学号：3220103645
日期：2023/10/14

# 浙江大学实验报告

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：bmp 文件读写及 rgb 和 yuv 色彩空间转化

## 一、实验目的和要求

通过项目实践，理解 bmp 文件的构成方式，掌握 bmp 文件的读入输出要点，以及 rgb 和 yuv 两种颜色模型的相互转换原理。

本实验要求编写一个程序，读入一个 bmp 图像，首先将其 rgb 转换为 yuv 空间来进行灰度处理，后转回 rgb，输出一张只有灰度，并且灰度分布在（0,255）的图像。另外，将原图用 yuv 处理，将 y 变动一定的值，转回 rgb 后再输出一张图片。最终得到两张输出图片并观察效果。

## 二、实验内容和原理

（简述实验有关的基本原理）

本次实验有两个难点，一是 bmp 文件的读入和输出，另外一个就是 rgb 和 yuv 空间的相互转化。

bmp 文件的全称叫做 bitmap，位图，其意思就是每一个像素的信息被存在若干个位上。这样的存储方式可以非常精确地记录像素信息，但是它因为过于精确，会导致文件过大。总之，知道了 bmp 的存储原理，针对 bmp 文件的读入，我们只需要知道几个位图的信息：行数和列数、像素起始点的位置。一般图都是 24 位的，也就是每一个像素由三个字节（B,G,R）记录，如果有顾虑可以在文件头查看 biBitCount 的量。

读入 bmp，首先要读入文件头。一些琐碎的量不再赘述，直奔主题——行数和列数分别是 bitWidth 和 bitHeight 中，它们位于 BitmapInfoHeader 中，使用 8 个字节存储。而像素起始点由 bfOffbits 存储，我们可以将指针移到 bfOffbits 的位置（文件开始为 0）进行读取。

需要注意的一点是 bmp 文件由于计算机读取方便需要，每读一行的数据都需要为四的倍数，如果一行末尾没有凑到 4 的倍数，那么补 0 来凑整。

Bmp 的输出如果注意到以上要点就非常好处理了。只需要将头文件原封不动地输出，然后将像素三个字节三个字节地输出（注意顺序是 B、G、R！），记得行尾补 0 即可。

## 三、实验步骤与分析

（每个步骤结合对应部分的源代码分析）

我将这个程序分成以下步骤：

1. 准备工作

**(1) 定义像素、提前写明转化条件**

```
class Pixel
{
public:
    unsigned char Bytes[3];
    double YUV[3];
    void ConvertRGB()
    {
        double buf1, buf2, buf3;
        buf1 = 1.0000 * YUV[0] + 1.1398 * YUV[2];
        buf2 = 0.9996 * YUV[0] - 0.3954 * YUV[1] - 0.5805 * YUV[2];
        buf3 = 1.0020 * YUV[0] + 2.0361 * YUV[1] - 0.0005 * YUV[2];
//avoid overflow, restrict byte 255
        if (buf1 > 255)
            buf1 = 255;
        if (buf2 > 255)
            buf2 = 255;
        if (buf3 > 255)
            buf3 = 255;
```

```
        Bytes[0] = round(buf1);
        Bytes[1] = round(buf2);
        Bytes[2] = round(buf3);
    }
    void ConvertYUV()
    {
        double buf1, buf2, buf3;
        buf1 = 0.299 * Bytes[0] + 0.587 * Bytes[1] + 0.114 *
Bytes[2];
        buf2 = -0.147 * Bytes[0] - 0.289 * Bytes[1] + 0.435 *
Bytes[2];
        buf3 = 0.615 * Bytes[0] - 0.515 * Bytes[1] - 0.100 *
Bytes[2];
        YUV[0] = buf1;
        YUV[1] = buf2;
        YUV[2] = buf3;
    }
};
```

**(2) 定义文件信息头**

```
class BMPHeader
{
public:
```

```
    unsigned char bfType[2];
    unsigned int bfSize;
    unsigned short bfReserved1;
    unsigned short bfReserved2;
    unsigned int bfOffBits;
} BMPHeader;
```

**(3) 定义图像信息头**

```
class BMPInfoHeader
{
public:
    unsigned int biSize;
    unsigned int biWidth;
    unsigned int biHeight;
    unsigned short int biPlanes;
    unsigned short int biBitCount;
    unsigned int biCompression;
    unsigned int biSizeImage;
    unsigned int biXPelsPerMeter;
    unsigned int biYPelsPerMeter;
    unsigned int biClrUsed;
    unsigned int biClrImportant;
```

```
} BMPInfoHeader;
```

**(4) 撰写读入读出文件头的预函数**

```
void ReadHeaders(FILE *Input)
{
    fread(&BMPHeader.bfType, sizeof(char), 2, Input);
    fread(&BMPHeader.bfSize, sizeof(int), 1, Input);
    fread(&BMPHeader.bfReserved1, sizeof(short), 1, Input);
    fread(&BMPHeader.bfReserved2, sizeof(short), 1, Input);
    fread(&BMPHeader.bfOffBits, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biSize, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biWidth, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biHeight, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biPlanes, sizeof(short), 1, Input);
    fread(&BMPInfoHeader.biBitCount, sizeof(short), 1, Input);
    fread(&BMPInfoHeader.biCompression, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biSizeImage, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biXPelsPerMeter, sizeof(int), 1,
Input);
    fread(&BMPInfoHeader.biYPelsPerMeter, sizeof(int), 1,
Input);
    fread(&BMPInfoHeader.biClrUsed, sizeof(int), 1, Input);
    fread(&BMPInfoHeader.biClrImportant, sizeof(int), 1, Input);
```

```
}

void WriteHeaders(FILE *Output)
{
    fwrite(&BMPHeader.bfType, sizeof(char), 2, Output);
    fwrite(&BMPHeader.bfSize, sizeof(int), 1, Output);
    fwrite(&BMPHeader.bfReserved1, sizeof(short), 1, Output);
    fwrite(&BMPHeader.bfReserved2, sizeof(short), 1, Output);
    fwrite(&BMPHeader.bfOffBits, sizeof(int), 1, Output);
    fwrite(&BMPInfoHeader.biSize, sizeof(int), 1, Output);
    fwrite(&BMPInfoHeader.biWidth, sizeof(int), 1, Output);
    fwrite(&BMPInfoHeader.biHeight, sizeof(int), 1, Output);
    fwrite(&BMPInfoHeader.biPlanes, sizeof(short), 1, Output);
    fwrite(&BMPInfoHeader.biBitCount, sizeof(short), 1, Output);
    fwrite(&BMPInfoHeader.biCompression, sizeof(int), 1,
Output);
    fwrite(&BMPInfoHeader.biSizeImage, sizeof(int), 1, Output);
    fwrite(&BMPInfoHeader.biXPelsPerMeter, sizeof(int), 1,
Output);
    fwrite(&BMPInfoHeader.biYPelsPerMeter, sizeof(int), 1,
Output);
    fwrite(&BMPInfoHeader.biClrUsed, sizeof(int), 1, Output);
```

```
    fwrite(&BMPInfoHeader.biClrImportant, sizeof(int), 1,
Output);
}
```

**(5) 灰度重映射关系函数**

```
// remap range[a,b] into [0,255]
double remap(double a, double b, double x)
{
    if (a == b)
        return 255.0 / 2;
    else
        return 255.0 * (x - a) / (b - a);
}
```

**(6) 通道交换准备函数**

```
// for channel R and B to swap
void swap(unsigned char *a, unsigned char *b)
{
    int buf = *a;
    *a = *b;
    *b = buf;
}
```

2. 执行部分

**(1) 文件指针打开操作**

```
FILE *BMP = fopen(".\\Try2.bmp", "rb");
FILE *Output1 = fopen(".\\output1.bmp", "wb+");
FILE *Output2 = fopen(".\\output2.bmp", "wb+");
if (BMP)
{
// omitted
}
else
{
  printf("\033[0m\033[1;33mThe file doesn't exist!\033[0m");
}
```

**(2) 读入文件头**

```
    ReadHeaders(BMP);
```

**(3) 输出文件头**

```
    WriteHeaders(Output1);
    WriteHeaders(Output2);
```

(4) 跳过赘余文件头（调色板）

```
for (int i = 55; i <= BMPHeader.bfOffBits; i++)
{
    unsigned char buf;
    fread(&buf, sizeof(char), 1, BMP);
    fwrite(&buf, sizeof(char), 1, Output1);
    fwrite(&buf, sizeof(char), 1, Output2);
}
```

(5) 读入 RGB 文件并转换为 YUV 空间，得出灰度分布范围

```
double HighestY = 0, LowestY = 255;
for (int i = 0; i < BMPInfoHeader.biHeight; i++)
{
    for (int j = 0; j < BMPInfoHeader.biWidth; j++)
    {
        Pixel pixel;
        fread(pixel.Bytes, sizeof(char), 3, BMP);
        swap(&pixel.Bytes[0], &pixel.Bytes[2]);
        pixel.ConvertYUV();
        if (LowestY > pixel.YUV[0])
            LowestY = pixel.YUV[0];
        if (HighestY < pixel.YUV[0])
            HighestY = pixel.YUV[0];
```

```
            }
            // please notice the number of pixels because there
will be some rear zeros.
            for (int j = BMPInfoHeader.biWidth * 3; j % 4 != 0; j++)
            {
                unsigned char buf;
                fread(&buf, sizeof(buf), 1, BMP);
            }
        }
```

**(6) 重新调整指针到像素开头，进行图像处理**

```
        rewind(BMP);
        for (int i = 1; i <= BMPHeader.bfOffBits; i++)
        {
            unsigned char buf;
            fread(&buf, sizeof(char), 1, BMP);
        }
        for (int i = 0; i < BMPInfoHeader.biHeight; i++)
        {
            for (int j = 0; j < BMPInfoHeader.biWidth; j++)
            {
           //omitted
            }
```

**(7) 拷贝像素**

```
Pixel pixel, Copy;
fread(pixel.Bytes, sizeof(char), 3, BMP);
swap(&pixel.Bytes[0], &pixel.Bytes[2]);
pixel.ConvertYUV();
Copy = pixel;
```

**(8) 新像素只取灰度、重映射后转回 RGB 输出**

```
// remapping Y to [0,255]
Copy.YUV[0] = remap(LowestY, HighestY,
pixel.YUV[0]);
// step4: apply transform1 to the copy
Copy.YUV[1] = Copy.YUV[2] = 0;
// step5: convert the pixel into RGB
Copy.ConvertRGB();
// step6: output the pixel to file1
swap(&Copy.Bytes[0], &Copy.Bytes[2]);
fwrite(Copy.Bytes, sizeof(char), 3, Output1);
```

**(9) 旧像素进行 Y 值灰度处理，转回 RGB 输出**

```
                // step7: apply transform2 to original image
                if (deltaY > 0)
                {
                    pixel.YUV[0] = (pixel.YUV[0] + deltaY >=
255) ? 255 : pixel.YUV[0] + deltaY;
                }
                else
                {
                    pixel.YUV[0] = (pixel.YUV[0] + deltaY < 0) ?
0 : pixel.YUV[0] + deltaY;
                }
                // step8: convert the pixel into RGB
                pixel.ConvertRGB();
                // step9: output the pixel to file2
                swap(&pixel.Bytes[0], &pixel.Bytes[2]);
                fwrite(pixel.Bytes, sizeof(char), 3, Output2);
```

**(10)文件指针关闭操作**

```
fclose(BMP);
fclose(Output1);
fclose(Output2);
```

## 四、实验环境及运行方法

（说明程序的编译环境和具体测试方法）

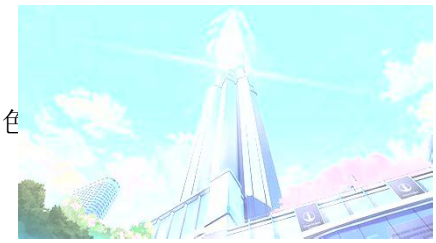
该程序使用 vscode 软件编写完成，可以使用 MingW 进行编译后，在执行程序的文件夹中放入 input.bmp 文件，输出 output1.bmp 以及 output2.bmp，目前经测试可以在 windows 系统上稳定运行。

## 五、实验结果展示

（展示实验中的输入输出图像等）

原图片(带调色

输出 1：灰度图



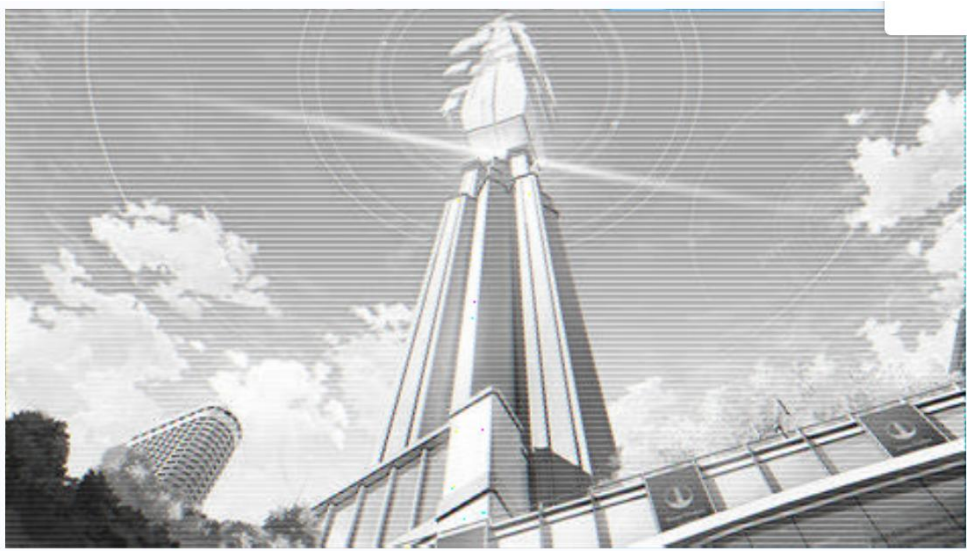
输出 2：加亮图

## 六、心得体会

在本次实验中我遇到了 3 个比较大的 BUG。

第一个 bug 是程序不能停止运行，导致输出的 bmp 图片文件极大，并且存在大量乱码。经过大量排除发现问题处于 for (int j = 0; j < BMPInfoHeader.biWidth; i++)，这是一个经典的错误，但是绝对不可轻视，经过这次教训，当之后程序如果出现无法停止的状况，不妨可以检查一下循环变量是否写对。

第二个 bug 是程序在运行后输出如下的有条纹的错误灰度图。经过查看二进制文件发现存在大量乱码。经过比对发现乱码的交界处正好是行末换行处，而这个地方有两个补 0，我在程序中没有考虑。也就是说，通过这个 BUG，我对 BMP 的结构有了进一步的了解。



第三个 BUG 是增加灰度 Y 是可能导致颜色产生反向突变，就像下面的图片显现的一样。起初我判断是由于灰度 Y 超过 255 溢出，但实际上做了限制后仍然无法解决这个问题，之后在调试转 RGB 函数的时候发现最后 RGB 计算结果部分超过了 255，所以强制转换为 unsigned char 了之后导致了颜色突变。进行了限制之后发现程序运行恢复了正常。