专业：计算机科学与技术
姓名：秦嘉俊
学号：3210106182
日期：2022 年 10 月 4 日

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

1. 熟悉 BMP 文件结构，进一步学习多媒体文件结构设计的思想。
2. 编程读取彩色 BMP 文件。
3. 实践 RGB 与 YUV 色彩空间的转换，将 BMP 文件转换至 YUV 色彩空间。
4. 编程转化成一张 BMP 灰度图片。
5. 改变 YUV 中的 Y 值，然后转化为一张彩色 BMP 图片。

二、 实验内容和原理

2.1 BMP 文件格式

位图文件 (Bitmap-File，BMP) 格式是 Windows 采用的图像文件存储格式，在 Windows 环境下运行的所有图像处理软件都支持这种格式。BMP 位图文件默认的文件扩展名是 bmp 或者 dib。BMP 文件大体上分为四个部分：图像文件头、图像信息头、调色板、图像数据字节阵列。对于用到调色板的位图，图像数据就是该像素颜色在调色板中的索引值（逻辑色）。对于真彩色图，图像数据就是实际的 R、G、B 值。图像的每一扫描行由表示图像像素的连续的字节组成，每一行的字节数取决于图像的颜色数目和用像素表示的图像宽度。规定每一扫描行的字节数必需是 4 的整倍数，也就是 DWORD 对齐的。扫描行是由底向上存储的，这就是说，阵列中的第一个字节表示位图左下角的像素，而最后一个字节表示位图右上角的像素。

图像文件头
图像信息头
调色板
图像数据

表 1: BMP 文件结构

BMP 具体表示如下：[1]

[1]大小以字节为单位

	名称	大小	内容
图像文件头 BITMAP FILE HEADER	bfType	2	说明文件的类型，该值必需是 0x4D42 也就是字符'BM'。
	bfSize	4	说明该位图文件的大小，用字节为单位
	bfReserved1	2	保留，必须为 0
	bfReserved2	2	保留，必须为 0
	bfOffBits	4	说明从文件头开始到实际的图象数据之间的字节的偏移量。 可以用这个偏移值迅速的从文件中读取到位数据。
图像信息头 BITMAP Information HEADER	biSize	4	说明 BITMAPINFOHEADER 结构所需要的字数。
	biWidth	4	说明图象的宽度，以象素为单位。
	biHeight	4	说明图象的高度，以象素为单位。 注：这个值还指明了该图像是倒向的位图，还是正向的位图。 如果该值是一个正数，说明图像是倒向的， 如果该值是一个负数，则说明图像是正向的。
	biPlane	2	为目标设备说明位面数，其值将总是被设为 1。
	biBitCount	2	说明比特数/象素，其值为 1、4、8、16、24、或 32
	biCompression	4	说明图象数据压缩的类型
	biSizeImage	4	说明图象的大小，以字节为单位。
	biXPelsPerMeter	4	说明水平分辨率，用象素/米表示。
	biYPelsPerMeter	4	说明垂直分辨率，用象素/米表示。
	biClrUsed	4	说明位图实际使用的彩色表中的颜色索引数 （设为 0 的话，则说明使用所有调色板项）。
	biClrImportant	4	说明对图象显示有重要影响的颜色索引的数目， 如果是 0，表示都重要。
调色板	Palette	$N \times 4$	调色板规范。 对于调色板中的每个表项， 这 4 个字节用下述方法来描述 RGB 的值： • 1 字节用于蓝色分量 • 1 字节用于绿色分量 • 1 字节用于红色分量 • 1 字节用于填充符 (设置为 0)
图像数据	Bitmap data	x	该域的大小取决于压缩方法，它包含所有的位图数据字节， 这些数据实际就是彩色调色板的索引号。

2.2 BMP 文件的读写

这里我们使用的是 24 位彩色 BMP 文件，利用 C 语言中的 `fopen fread fseek fwite` 进行读入写出即可。

需要注意的是，我们的 24 位彩色 BMP 文件是不含有调色板的，而输出的 8 位灰度图是需要调色板

的，因此我们需要对图像文件头和图像信息头中的内容进行调整。

2.3 RGB 和 YUV 的相互转化

RBG 颜色模型是三维直角坐标颜色系统中的一个单位正方体在正方体的主对角线上，各原色的量相等，产生由暗到亮的白色，即灰度。（0，0，0）为黑，（1，1，1）为白，正方体的其他 6 个角点分别为红、黄、绿、青、蓝和品红。RGB 颜色模型构成的颜色空间是 CIE 原色空间的一个真子集。RGB 颜色模型通常用于彩色阴极射线管和彩色光栅图形显示器（计算机和电视机采用）。

YUV，是一种颜色编码方法。常使用在各个视频处理组件中。YUV 在对照片或视频编码时，考虑到人类的感知能力，允许降低色度的带宽。YUV 是编译 true-color 颜色空间（color space）的种类，Y'UV, YUV, YCbCr，YPbPr 等专有名词都可以称为 YUV，彼此有重叠。“Y”表示明亮度（Luminance 或 Luma），也就是灰阶值，“U”和“V”表示的则是色度（Chrominance 或 Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。

RGB 转化为 YUV 的公式如下：$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.435 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

而 YUV 转化为 RGB 的公式如下：$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0 & 1.4075 \\ 1.0000 & -0.3455 & -0.7169 \\ 1.0000 & 1.7790 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

当我们想要转为灰度图时，只需要保留 Y 同时舍弃掉 U 和 V 分量即可得到灰度图；当我们想要调节亮度时，只需要改变 Y 分量的值，随后将 YUV 转回 RGB 即可得到被更改过亮度的图片。

三、 实验步骤与分析

3.1 结构体的定义

```
typedef unsigned char BYTE; // 1 byte
typedef unsigned short WORD; // 2 bytes
typedef unsigned int DWORD; // 4 bytes
typedef int LONG; // 4 bytes
#pragma pack(1)
// 图像文件头
typedef struct tagBITMAPFILEHEADER {
    WORD bfType; // 文件类型 "BM"
    DWORD bfSize; // 位图文件的大小，以字节为单位
    WORD bfReserved1;
    WORD bfReserved2; // 1 和 2 均设置为 0
    DWORD OffBits; // 文件头到实际图像数据之间的字节偏移
}BITMAPFILEHEADER;
// 图像信息头
typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
```

```
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrused;
    DWORD biClrImport;
}BITMAPINFOHEADER;
//调色盘
typedef struct Palette {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
}RGBQUAD;
// bmp file structure
typedef struct tagBMPFILESTRUCT {
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    RGBQUAD aColors[256];
    BYTE *aBitmapBits; //会有多少种不同的颜色，这样相同的颜色只需要存储对应行，而不是多次存相同的像素值
}BMPFILE;
```

按照原理 2.1 中的部分定义结构体即可，需要注意的是由于 C 语言在存储结构体时会有对齐方面的问题，而文件中的数据是连续的，这会导致后序的读取出现错误。因此要使其连续排列。因此需要加上 `#pragma pack(1)`.

3.2 BMP 文件的读取

```
int main()
{
    FILE *fp;
    fp = fopen("mouse.bmp", "rb"); // rb 打开一个二进制文件
    if (!fp) {
        printf("BMP Image Not Found!\n");
        exit(0);
    }
    printf("Successfully open the image\n");
    fread(&(a.bmfh), sizeof(BITMAPFILEHEADER), 1, fp);
    fread(&(a.bmih), sizeof(BITMAPINFOHEADER), 1, fp);
    ImageHeight = a.bmih.biHeight;
    ImageWidth = a.bmih.biWidth;
    // 注意字节必须是 4 的整数倍
    if(! a.bmih.biSizeImage) // 注意 biSizeImage 可能为 0 !
        a.bmih.biSizeImage = a.bmfh.bfSize - a.bmfh.OffBits;
    ImageSize = a.bmih.biSizeImage; // 所有像素所占的字节数.
    row_byte = (3 * ImageWidth + 3) / 4 * 4;
    // row_byte = (ImageWidth * a.bmih.biBitCount + 31) / 32 * 4;
    // printf("shabi:%d\n",row_byte);
    // fread(a.aColors, sizeof(RGBQUAD), 256, fp); // 这里的bmp没有调色板
    fseek(fp, a.bmfh.OffBits, SEEK_SET);
    a.aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * ImageSize);
    fread(a.aBitmapBits, ImageSize * sizeof(BYTE), 1, fp);
    fclose(fp);
    Solve_GRAY();
```

```
    Solve_COLOR();
    // free(a.aBitmapBits);
}
```

需要注意的是，biSizeImage 这一项有时会为 0，应该注意到这一种情况，以免之后用到图像大小的时候出现错误。

此外，BMP 是一个二进制文件，因此读入输出都需要在 fopen 时加上“b”选项。

`row_byte` 是用来计算每一行需要多少字节，因为我们读入的是 3 bytes 的 BMP 文件，biWidth 以像素为单位，一个像素三字节，但每一行字节又必须是 4 的整数倍，因此需要做一个对齐。

3.3 转灰度图

```
int Get_Position(int x, int y)
{
    return x * row_byte + y * 3;
}
void RGB_To_YUV(double R, double G, double B, double* Y, double* U, double* V)
{
    *Y = 0.299 * R + 0.587 * G + 0.114 * B;
    *U = -0.147 * R - 0.289 * G + 0.435 * B;
    *V = 0.615 * R - 0.515 * G - 0.100 * B;
}
void Solve_GRAY()
{
    int i, j, new_row_byte;
    // b = a;
    memcpy(&(b.bmfh), &(a.bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(b.bmih), &(a.bmih), sizeof(BITMAPINFOHEADER));
    b.bmfh.OffBits = 256 * 4 + 40 + 14; // 256 种颜色 * 每种颜色四个字节 + 图像文件头 + 图像信息头
    new_row_byte = (ImageWidth + 3) / 4 * 4; // 重新计算每行需要的字节
    b.bmih.biBitCount = 8; // 灰度图只需要 8 位
    b.bmih.biSizeImage = ImageHeight * new_row_byte; // 注意这里行也要变化
    // 注意 biSizeImage 只计算图像数据(bitmap data), 而bfSize还要包括前面的图像信息文件头和调色板
    b.bmfh.bfSize = b.bmih.biSizeImage + b.bmfh.OffBits;
    b.aBitmapBits = (BYTE *)calloc(b.bmih.biSizeImage, sizeof(BYTE));

    for (i = 0; i < 256; i++)
        b.aColors[i].rgbBlue = b.aColors[i].rgbGreen = b.aColors[i].rgbRed = i;

    for (i = 0; i < ImageHeight; i++)
        for (j = 0; j < ImageWidth; j++) {
            int now = Get_Position(i, j);
            double B = a.aBitmapBits[now]; // 注意顺序是 B G R
            double G = a.aBitmapBits[now+1];
            double R = a.aBitmapBits[now+2];
            double Y, U, V;
            RGB_To_YUV(R, G, B, &Y, &U, &V);
            // if(Y > 255 || Y < 0) printf("shabi");
            Update(Y); // 将 Y rearrange 到 [0, 255] 这个区间上
            b.aBitmapBits[new_row_byte * i + j] = Y;
        }
    FILE *fp = fopen("mouse_gray.bmp", "wb");
    Print(&b, fp);
```

```
}
```

我们将 RGB 转换为 YUV 后仅保留 Y, 就可以得到一张灰度图。灰度图相比于 24 位彩色图，多了一项调色板。彩色表/调色板是单色、16 色和 256 色图像文件所特有的，相对应的调色板大小是 2、16 和 256，调色板以 4 字节为单位，每 4 个字节存放一个颜色值，图像的数据是指向调色板的索引。在这里我们本来的彩色图是没有调色板的，因此需要更改文件和信息头的一些信息。

3.4 改变亮度

```
void YUV_To_RGB(double Y, double U, double V, double* R, double* G, double* B)
{
    *R = Y + 1.4075 * V;
    *G = Y - 0.3455 * U - 0.7169*V;
    *B = Y + 1.779 * U;
}
void Solve_COLOR()
{
    int i, j;
    for (i = 0; i < ImageHeight; i++)
        for (j = 0; j < ImageWidth; j++) {
            int now = Get_Position(i, j);
            double B = a.aBitmapBits[now];
            double G = a.aBitmapBits[now+1];
            double R = a.aBitmapBits[now+2];
            double Y, U, V;
            RGB_To_YUV(R, G, B, &Y, &U, &V);
            Y *= 0.5;
            // Y *= 1.5;
            Y = Update(Y);
            YUV_To_RGB(Y, U, V, &R, &G, &B);
            R = Update(R);
            G = Update(G);
            B = Update(B);
            a.aBitmapBits[now] = B;
            a.aBitmapBits[now+1] = G;
            a.aBitmapBits[now+2] = R;
        }
    FILE *fp = fopen("mouse_dark.bmp", "wb");
    // FILE *fp = fopen("mouse_light.bmp", "wb");
    Print(&a, fp);
}
```

这里我们仍然要输出彩色图,所以不需要改变原本的结构体,只是将 RGB 转为 YUV 后改变 Y 的值,再换回 RGB 即可。我这里尝试了将 Y 缩小一半（得到 `mouse_dark.bmp`）和扩大一倍（得到 `mouse_light.bmp`）。需要注意的是，在进行 RGB YUV 的转化时，要使用 double 类型，减小精度误差。

3.5 输出 BMP 文件

```
void Print(BMPFILE *tmp, FILE *fp)
{
    if(!fp) {
```

```
        printf("ERROR!");
        exit(0);
    }
    if((tmp->bmih).biBitCount == 8) // 如果是灰度图，那需要输出调色板
        fwrite(tmp, sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD), 1, fp);
    else // 彩色图不需要调色板，直接输出即可
        fwrite(tmp, sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER), 1, fp);
    fwrite(tmp->aBitmapBits, (tmp->bmih).biSizeImage * sizeof(BYTE), 1, fp);
    fclose(fp);
    free(tmp->aBitmapBits); // 不要忘了 free
}
```

我们每次在调用 `Print` 前打开一个文件（以便我们在不同的函数打开不同的文件，如 `mouse_gray` `mouse_dark` 就分别对应转为灰度图和改变亮度的函数）然后把文件描述符传入函数并进行输出。需要注意的是在我们的实现中灰度图有调色板但彩色图没有，以及最后要关闭文件描述符，释放内存。

四、 实验环境及运行方法

4.1 实验环境

Windows 10 系统

gcc 10.3.0 (tdm64-1) x86_64-w64-mingw32

4.2 运行方法

源文件为 lab1.c, 在源文件所在文件夹里有我们的 24 位彩色 BMP 图像 (mouse.bmp), 使用 VSCode 打开这个文件夹，并选中 lab1.c 点击 Run Code 即可开始运行。当终端出现“Successfully open the image”时说明我们成功打开了图像，否则会输出“BMP Image Not Found!”

随后我们的程序会对图像进行转化，并输出 mouse_gray.bmp 和 mouse_dark.bmp. 然后我更改源代码第 137 行的内容（`Y *= 0.5;`）改为 `Y *= 1.5;`，并修改 `Solve_Color` 函数里的打开文件名称，改为 mouse_light.bmp. 再通过第一段的方式运行即可得到亮度更高的 mouse_light.bmp.

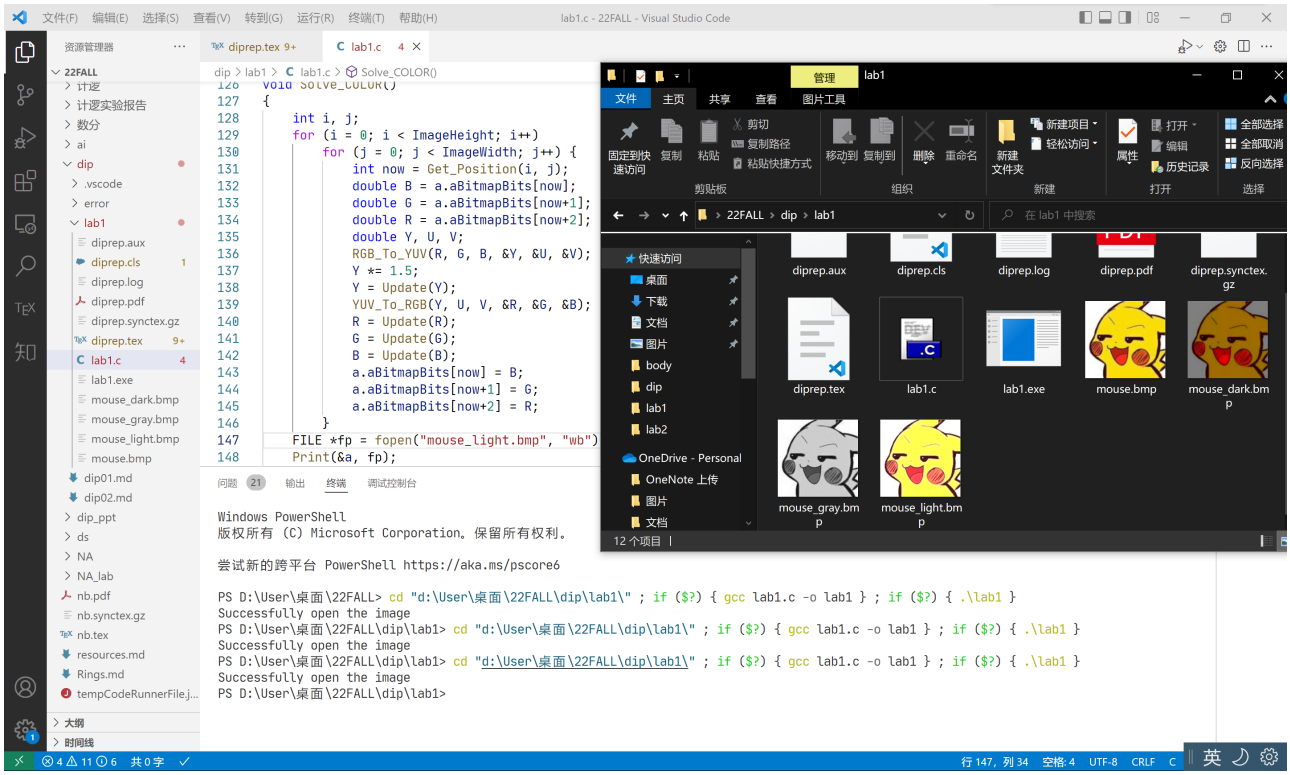


图 1: 运行结束后

五、 实验结果展示

输入图像：

图 2: 输入的 24 位彩色 BMP 图像 (mouse.bmp)

输出图像：

图 3: 灰度图 (mouse_gray.bmp)

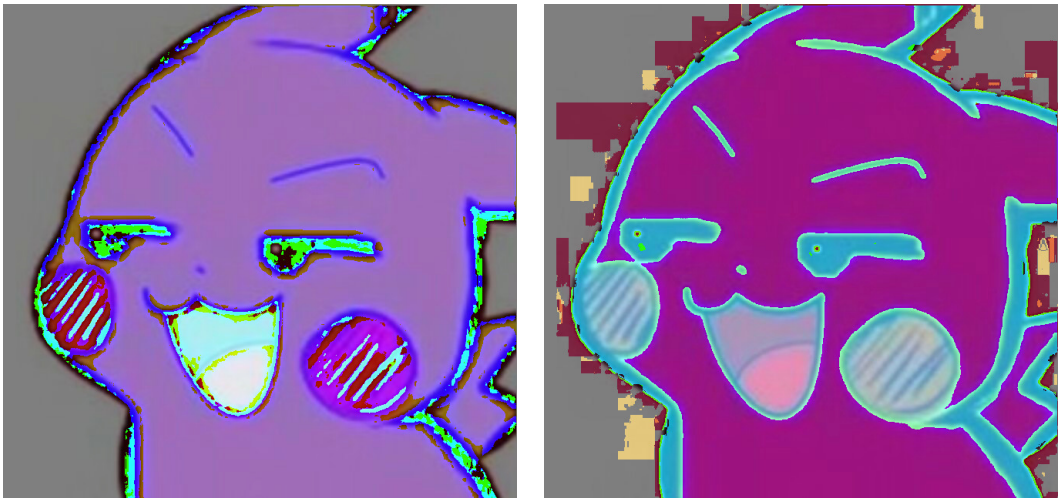
图 4: 调整亮度后的彩色 BMP 图 1(mouse_dark.bmp)

图 5: 调整亮度后的彩色 BMP 图 2(mouse_light.bmp)

六、 心得体会

第一次自己上手实验,还是有很多不适应的地方,也出了很多问题。包括但不限于没注意到 biSizeImage 可能为 0; 有的图有调色板，有的图没有（这还是借助了 UltraEdit 查看二进制文件才发现）；灰度图要改变文件信息；row bytes 必须是 4 的倍数等问题。

还有两个比较关键的问题，要使用 double 来减少精度带来的误差，以及要检查转化后的 RGB YUV 是否超过 [0, 255] 的边界。



(a) 没有检查分量是否越界　　(b) 没有使用 double

经过了这么多折腾，也算是完成了这个 lab，我也感受到了图像信息处理的有趣之处，期待之后更加精彩的实验！