专业：计算机科学与技术
姓名：张祎迪
学号：3220102157
日期：2023/10/8

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

实验目的

1.理解BMP图像的文件结构和像素点色彩存储方式。

2.掌握RGB到YUV，YUV到RGB的转换原理和方法。

实验要求

1.编写程序完成BMP图像由RGB到YUV和由YUV到RGB的转换。

2.读取彩色BMP图像并将其转换为灰度图像。

3.将灰度图像的灰度值重新映射到[0, 255]范围内，保存灰度图像。

4.修改灰度图像中的亮度通道 Y （例如增加或减小亮度），将修改后的YUV 图像重新转换回 RGB 颜色并保存。

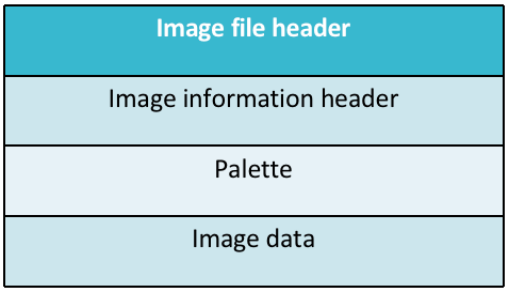
二、实验内容和原理

1.BMP的文件格式：

BMP（Bitmap）是一种常见的图像文件格式，是Windows系统的一种标准文件格式。BMP 位图文件默认的文件扩展名是 bmp 或者 dib。大多数情况下，BMP 图像采用非压缩方式；然而，它也支持图像压缩，例如 RLE 格式。下面介绍BMP文件的具体文件结构。

> BMP 图像每一行扫描由表示图像像素的连续的字节组成，每一行以四字节对齐（以0补齐）。图像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。

（1）BMP文件整体结构：



BMP 文件 大体上分为四个部分:图像文件头（Image file header）、图像信息头（Image information header）、调色板（Palette）、图像数据字节阵列（Image data）。对用到调色板的位图，图像数据为该像素颜色在调色板中的索引值。对于真彩色图，图像数据就是实际的 R、G、B 值。

（2）BMP图像文件头（Image file header）

Start	Size(Byte)	Name	Purpose
1	2	bfType	Must always be set to 'BM' to declare that this is a .bmp-file
3	4	bfSize	Specifies the size of the file in bytes.
7	2	bfReserved1	Must always be set to zero.
9	2	bfReserved2	Must always be set to zero.
11	4	bfOffBits	Specifies the offset from the beginning of the file to the bitmap data

（3）BMP图像信息头（image information header）

Size(Byte)	Name	Purpose
4	biSize	Number of bytes to define BITMAPINFOHEADER structure
4	biWidth	Image width (number of pixels)
4	biHeight	Image height (number of pixels).
2	biPlane	Number of planes. Always be 1.
2	biBitCount	Bits per pixel (Bits/pixel), which is 1, 4, 8, 16, 24 or 32.
4	biCompression	Compression type. Only non-compression is discussed here: BI_RGB.
4	biSizeImage	Image size with bytes. When biCompression=BI_RGB, biSizeImage=0.
4	biXPelsPerMeter	Horizontal resolution, pixels/meter.
4	biYPelsPerMeter	Vertical resolution, pixels/meter
4	biClrUsed	Number of color indices used in the bitmap (0->all the palette items are used).
4	biClrImportant	Number of important color indices for image display. 0->all items are important.

- 注明：$biHeight$ 可以表示图像是否倒置，$biHeight$ 为正数表示倒置（inverted）， 负数表示正常（upright)，大多数BMP文件是倒置的位图，即 $biHeight$>0。

（4）调色板（Palette）

调色板的大小为$N*4$ (bytes)。调色板中的每一项用1 字节表示蓝色分量、1 字节表示绿色分量、1 字节表示红色分量、1 字节用于填充符 (设置为 0)。

（5）图像数据字节阵列（Image data）

图像数据字节阵列存储了调色板的索引号，或者取决于颜色深度的RGB值。其大小取决于图像大小和颜色深度。

2.BMP文件的读入和存储

利用 C 语言中的 *fopen fread fwrite* 进行读入、写出即可。在读入与写出的过程中，需要注意 BMP文件信息的调整，例如由24位彩色BMP转为灰色图时，需要注意调色板、位深等的调整。

3.RGB 和 YUV 的相互转化

RGB（红绿蓝）和YUV是两种颜色表示方式，常用于图像和视频处理中。RGB是一种直接表示颜色的方式，其中红色（R）、绿色（G）和蓝色（B）分量的值决定了颜色的具体外观。RGB可以浮点表示方式：取值范围为 `0.0 ~ 1.0`；或以整数表示：取值范围为 `0 ~ 255` 或者 `00 ~ FF`。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是一种颜色空间，基于 `YUV` 的颜色编码是流媒体的常用编码方式，这种表达方式起初是为了彩色电视与黑白电视之间的信号兼容；其中：Y：表示明亮度（Luminance 或 Luma），也称灰度图。U、V：表示色度（Chrominance 或 Chroma），作用是描述影像的色彩及饱和度，用于指定像素的颜色。

- RGB转化为YUV的公式如下：

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.435 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

- YUV 转化为RGB的公式如下：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.0000 & 1.3707 \\ 1.0000 & -0.3376 & -0.6980 \\ 1.0000 & 1.7324 & 0.0000 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{2}$$

由BMP彩色图转为灰度图时，只需将RGB转为YUV，并只保留 Y分量 即可得到灰度图在调节BMP图片亮度时，只需将RGB转为YUV，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

1.根据BMP文件结构定义结构体

```
// 定义BMP文件头的大小为54个字节，通常情况下固定不变
#define HEADERSIZE 54

// 定义几种不同数据类型的别名，以便在代码中使用更具可读性的名称
typedef unsigned char byte;
typedef unsigned short word;
typedef unsigned int dword;

// 使用#pragma pack(1)指令设置结构体的字节对齐方式为1字节
#pragma pack(1)

// 定义BMP文件头的结构体
typedef struct {
    word bfType;          // BMP文件的类型标识，通常为"BM"
    dword bfSize;         // BMP文件的大小，以字节为单位
    word reserved1;       // 保留字段，设置为0
    word reserved2;       // 保留字段，设置为0
    dword bfOffset;       // 位图数据的偏移量，即文件头的大小
} BMPHeader;

// 定义BMP信息头的结构体
typedef struct {
```

```
    dword size;            // 信息头的大小，通常为40字节
    int width;             // 图像的宽度，以像素为单位
    int height;            // 图像的高度，以像素为单位
    word planes;           // 颜色平面数，通常为1
    word bitCount;         // 每个像素的位数
    dword compression;     // 压缩类型
    dword imageSize;       // 位图数据的大小，以字节为单位
    int xPixelsPerMeter;   // 水平分辨率，每米的像素数
    int yPixelsPerMeter;   // 垂直分辨率，每米的像素数
    dword colorsUsed;      // 使用的颜色数
    dword colorsImportant; // 重要颜色数
} BMPInfoHeader;

// 定义调色板中的一个颜色的结构体
typedef struct {
    byte blue;     // 蓝色分量
    byte green;    // 绿色分量
    byte red;      // 红色分量
    byte reserved; // 保留字段，通常设置为0
} pallete;

// 定义包含BMP文件头、BMP信息头、调色板和位图数据的完整BMP图像的结构体
typedef struct {
    BMPHeader bmph;          // BMP文件头
    BMPInfoHeader bmpih;     // BMP信息头
    pallete colours[256];    // 调色板，通常用于8位颜色索引图像
    byte *bitmap;            // 位图数据
} BMP;
```

2.BMP文件的读入

```
int main(){
    // 打开名为"input.bmp"的BMP文件以读取二进制数据
    FILE *inputFile = fopen("input.bmp", "rb");

    // 检查文件是否成功打开
    if (!inputFile) {
        printf("Unable to open input BMP file.\n"); // 输出错误信息
        return 0; // 退出程序
    }

    printf("successfully loaded!"); // 成功加载的提示信息

    BMP bmp; // 创建一个BMP结构体变量用于存储BMP文件的信息和数据

    // 从文件中读取BMP文件头和信息头
    fread(&(bmp), HEADERSIZE, 1, inputFile);
    int width = bmp.bmpih.width; // 获取图像的宽度
    int height = bmp.bmpih.height; // 获取图像的高度
    // 如果图像大小为0，则计算图像数据大小
    if (bmp.bmpih.imageSize == 0) {
```

```
21          bmp.bmpih.imageSize = bmp.bmph.bfSize - bmp.bmph.bfOffset;
22      }
23
24      // 为位图数据分配内存
25      bmp.bitmap = (byte*)malloc(sizeof(byte) * (bmp.bmph.bfSize - HEADERSIZE));
26
27      // 从文件中读取位图数据
28      fread(bmp.bitmap, bmp.bmph.bfSize - HEADERSIZE, 1, inputFile);
29      fclose(inputFile); // 关闭文件
30
31      // 转换图像为灰度图像
32      Gray_Transform(bmp, height, width);
33      //修改亮度
34      Brightness_Transform(bmp,height,width);
35      free(bmp.bitmap);
36      return 0;
37  }
```

需要注意如果原始未给出图像数据大小的信息，需要自己计算（line 19）

3.RGB与YUV的相互转换

```
1  void RGB_YUV(double R,double G,double B,double* Y,double* U,double* V){
2      *Y=0.299*R + 0.587*G+0.114*B;
3      *U = -0.147*R-0.289*G+0.435*B;
4      *V = 0.615*R-0.515*G-0.100*B;
5  }
6  void YUV_RGB(double Y,double U,double V,double* R,double* G,double* B){
7      *R = Y+1.3707*V;
8      *G = Y-0.3376*V-0.6980*U;
9      *B = Y+1.7324*U;
10 }
```

按照原理中的公式进行相互转换即可。

4.生成灰度图

```
1  void Gray_Transform(BMP bmp, int height, int width) {
2      BMP Gbmp; // 创建一个新的BMP结构体用于存储灰度图像
3      int row = (width + 3) / 4 * 4; // 计算每行像素数据的字节数，确保按4字节对齐
4      // 复制原始BMP的文件头和信息头到Gbmp
5      memcpy(&Gbmp, &bmp, HEADERSIZE);
6      Gbmp.bmpih.bitCount = 8; // 设置位图的位深度为8位
7      Gbmp.bmph.bfOffset = 256 * 4 + HEADERSIZE; // 计算新的数据偏移量
8      Gbmp.bmpih.imageSize = height * row; // 计算新的图像数据大小
9      Gbmp.bmph.bfSize = Gbmp.bmpih.imageSize + Gbmp.bmph.bfOffset; // 计算新的文件大
   小
10     // 为新的位图数据分配内存并初始化
11     Gbmp.bitmap = (byte*)malloc(Gbmp.bmpih.imageSize * sizeof(byte));
12     // 初始化灰度调色板，将RGB值与索引关联
13     for (int i = 0; i < 256; i++) {
```

```
        Gbmp.colours[i].blue = i;
        Gbmp.colours[i].green = i;
        Gbmp.colours[i].red = i;
    }
    int old_byte = (3 * width + 3) / 4 * 4; // 计算每行原始像素数据的字节数
    // 遍历原始图像像素，计算YUV值并填充到新的位图数据中
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int cnt = i * old_byte + j * 3;
            double B = bmp.bitmap[cnt];
            double G = bmp.bitmap[cnt + 1];
            double R = bmp.bitmap[cnt + 2];
            double Y, U, V;
            RGB_YUV(R, G, B, &Y, &U, &V);
            Adjust(&Y);//防止Y越界
            Gbmp.bitmap[row * i + j] = Y;
        }
    }
    // 打开文件并将新的BMP数据写入文件
    FILE* fp = fopen("gray.bmp", "wb");
    fwrite(&(Gbmp.bmph), HEADERSIZE, 1, fp); // 写入文件头
    fwrite(Gbmp.colours, 4 * 256, 1, fp); // 写入调色板数据
    fwrite(Gbmp.bitmap, Gbmp.bmpih.imageSize, 1, fp); // 写入位图数据
    fclose(fp); // 关闭文件
    free(Gbmp.bitmap);
}
```

(1) 需要注意由原始24位bmp彩图转为8位bmp灰度图的几个变化：

- bitcount由24变为8，同时导致每行的字节数改变（即由一个像素点对应三个字节变为一个像素点1一个字节），同时要注意按照4字节对齐。
- 增加了调色板，联通像素点的字节变化，会导致bfOffSet,imageSize,bfSize的改变。

（2）在转为灰度图时需要注意Y不能越界，采用Adjust函数调整

```
void Adjust(double* x){
    if(*x>=0){
        if(*x>255.0){*x=255.0;}
    }
    else{*x = 0.0;}
}
```

5.修改亮度

```
void Brightness_Transform(BMP bmp, int height, int width) {
    int row = (3 * width + 3) / 4 * 4; // 计算每行像素数据的字节数，确保按4字节对齐
    // 遍历图像的每个像素
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int cnt = i * row + j * 3; // 计算当前像素在位图数据中的偏移量
            // 获取当前像素的RGB值并将其转换为YUV颜色空间
```

```
            double B = bmp.bitmap[cnt];
            double G = bmp.bitmap[cnt + 1];
            double R = bmp.bitmap[cnt + 2];
            double Y, U, V;
            RGB_YUV(R, G, B, &Y, &U, &V);
            // 调整亮度
            Y = Y * 0.5;
            Adjust(&Y);//防止Y越界
            // 将调整后的YUV值转换回RGB颜色空间
            YUV_RGB(Y, U, V, &R, &G, &B);
            //防止RGB越界
            Adjust(&R);
            Adjust(&G);
            Adjust(&B);
            // 更新图像像素的RGB值
            bmp.bitmap[cnt] = (byte)B;
            bmp.bitmap[cnt + 1] = (byte)G;
            bmp.bitmap[cnt + 2] = (byte)R;
        }
    }
    // 打开文件并将调整后的BMP数据写入文件
    FILE* fp = fopen("brightness.bmp", "wb");
    fwrite(&(bmp), HEADERSIZE, 1, fp); // 写入文件头
    fwrite(bmp.bitmap, bmp.bmpih.imageSize, 1, fp); // 写入位图数据
    fclose(fp); // 关闭文件
}
```

可改变**Y=Y*0.5**来做其他的亮度调整

四、实验环境及运行方法

实验环境:

MacBook Air M2 Sonoma 14.0

Apple clang version 15.0.0(arm64-apple-darwin23.0.0)

运行方法:

打开lab01文件夹，用vscode打开其中的code文件夹，其中包含源文件hw1.c，可执行文件hw1mac,hw1.exe和24位彩色BMP图像`input.bmp`。

> （1）打开hw1.c并点击Run Code可开始运行。输出"successfully loaded!"表示文件正常读入，之后会完成实验要求的灰度图转化和修改亮度的操作，并输出相应的图片。
>
> （2）如果是Mac用户 在终端中cd进入code目录 输入 chmod +x hw1mac 为其添加执行权限，接着输入 ./hw1mac 可得到灰度图和一张经过“Y=Y*0.5”亮度调整的bmp彩色图。
>
> （3）如果是windows用户，可运hw1.exe,输出效果与（2）相同

注：程序默认修改亮度为调暗`Y=Y*0.5`(在源代码165行)，可修改使得运行对图片亮度进行不同的调节。

五、实验结果展示

输入24位彩色bmp图像：

输出灰度图：

调整亮度变暗（Y=Y*0.5）

调整亮度变亮（Y=Y*1.5）

六、心得体会

在本次实验中，最开始进行灰度图转换的时候一直不成功，首先简单看输出灰度图的二进制文件，发现只有header信息，发现是由于input.bmp原始的imageSize为零，在程序中进行了重新计算。但之后仍然转换不成功，发现是由于使用了错误的input.bmp(最开始我错误地使用了32位的input.bmp)，利用$HexFxiend$ 仔细查看后发现错误，转换了input.bmp的格式，得以顺利输出灰度图。

在调整色图亮度的过程中，由于最开始没有注意到调整后的RGB会越界的问题导致输出的图片严重失真，增加了Adjust函数后成功输出。