专业：计算机科学与技术
姓名：
学号：
日期：2021/10/24

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. 读取一张彩色 BMP 图片
2. 将 RGB 转换成 YUV
3. 设置灰度值为 YUV 颜色空间中的 Y（灰度要转化成[0,255]）
4. 写入一张灰度图片
5. 改变 Y 值
6. 将 YUV 转换成 RGB
7. 写入一张彩色图片

二、实验内容和原理

1. BMP 文件的格式

BMP 文件数据分成 4 个部分，分别是位图文件头、图像信息头、调色板和位图数据。在本实验中主要用到的数据有：位图文件头的文件表示，即最前面两个字符'BM'，这两个字符的 16 进制 ASCII 码为 04D42H；位图文件头偏移地址为 0AH 且为 4 个字节的数据，为文件开始到位图数据之间的偏移量，用来跳转到位图数据块；图像信息头使用到偏移地址为 12H 的位图宽度和 16H 的位图高度，这两个数据都是 4 个字节，且单位都是像素；另外就是位图的数据，注意位图存储时每一行最后会填上 0 以满足一行是 4 的倍数，且图像一般是倒着存储的。

2. RGB 和 YUV 的转换

RGB 是我们较熟悉的颜色空间，而 YUV 中 Y 表示明亮度，U 和 V 表示的是色度，作用是用来描述影像的色彩和饱和度。RGB 和 YUV 的转换方程有很多，笔者选择了其中一种可以确保转换后 Y 保持[0,255]之间的方法，如下：

$$Y = 0.299R + 0.587G + 0.144B$$
$$U = -0.169R - 0.331G + 0.500B + 128$$
$$V = 0.500R - 0.419G - 0.081B + 128$$

$$R = Y + 1.403 * (V - 128)$$
$$G = Y - 0.343 * (U - 128) - 0.714 * (V - 128)$$
$$B = Y + 1.770 * (U - 128)$$

三、实验步骤与分析

1. 读取一张 BMP 彩色图片

笔者选用一张 Lena 的 BMP 照片进行处理。读取图片使用的函数是 ReadData()。

首先，对于读入的图片进行验证是否是 BMP 文件，方法就是验证最前面两个字符是否是'BM'（16 进制 ASCII 码是 04D42H）。笔者使用一个 unsigned short 的名为 bmFlag 来读取。

```
// verify the file
fseek(finput, 0L, SEEK_SET);
fread(&bmFlag, sizeof(char), 2, finput);
if (bmFlag != 0x4d42) // Ox4d42 is the ASCII of 'B' 'M'
{
    printf("This is not BMP file!\n");
    exit(1);
}
```

然后，读取图片的宽度和高度。将文件指针转移到 12H 和 16H 处，然后分别用 unsigned int 的 height 和 width 变量来读取长度和宽度（均为 4 字节）。

```
// read the width and height of the BMP
fseek(finput, 0x12L, SEEK_SET);
fread(&width, sizeof(char), 4, finput);
fseek(finput, 0x16L, SEEK_SET);
fread(&height, sizeof(char), 4, finput);
```

接下来读取 4 个字节的位图数据偏移地址，笔者使用一个 unsigned int 的 offsetData 来读取。

```
// read the offset of data
fseek(finput, 0xaL, SEEK_SET);
fread(&offsetData, sizeof(char), 4, finput);
```

最后是读取位图数据。笔者首先计算出每一行有多少字节。计算的方法是每个像素占 3 个字节即 24 位，每一行又必须是 4 的倍数，得出公式为

$$dataSizePerLine = (3 * width + 3) / 4 * 4$$

所以笔者每次读入一行，并将每个像素的 RGB 分别存到一个数组中去。这三个数组分别命名为 r、g、b。注意由于 BMP 是倒置存储，笔者在这里也是倒着读入的。

```
// BMP request the dataSizePerLine be the multiple of 4
int dataSizePerLine = (width * 3 + 3) / 4 * 4;
unsigned char *dataLine = (unsigned char*)malloc(sizeof(char) * dataSizePerLine);
*r = (float*)malloc(sizeof(float) * width * height);
*g = (float*)malloc(sizeof(float) * width * height);
*b = (float*)malloc(sizeof(float) * width * height);

int i, j;
for (i = 0; i < width * height; i++)
    (*r)[i] = (*g)[i] = (*b)[i] = 0;

for (i = 0; i < height; i++)
{
    // every time we fread one line and start to process
    fread(dataLine, sizeof(char), dataSizePerLine, finput);
    for (j = 0; j < width; j++)
    {
        // pay attention to the direction
        (*r)[width * (height - i - 1) + j] = (float)dataLine[j * 3 + 2];
        (*g)[width * (height - i - 1) + j] = (float)dataLine[j * 3 + 1];
        (*b)[width * (height - i - 1) + j] = (float)dataLine[j * 3];
    }
}
free(dataLine);
```

2. 将 RGB 转换成 YUV

函数为 ConvertRGB2YUV。和 RGB 类似，笔者也建立了 3 个数组，分别是 y、u、v，使用上面原理中提到的公式进行转换即可。笔者先建立一个转换矩阵，然后套用公式。

```
// transform matrix is as follows
float T[3][3] = {
                {0.299, 0.587, 0.114},
                {-0.169, -0.331, 0.5},
                {0.5, -0.419, -0.081}
                };
*y = (float*)malloc(sizeof(float) * width * height);
*u = (float*)malloc(sizeof(float) * width * height);
*v = (float*)malloc(sizeof(float) * width * height);

int i, j;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        // the following transformation ensures that Y U V is [0, 255]
        (*y)[i * width + j] = r[i * width + j] * T[0][0] + g[i * width + j] * T[0][1] + b[i * width + j] * T[0][2];
        (*u)[i * width + j] = r[i * width + j] * T[1][0] + g[i * width + j] * T[1][1] + b[i * width + j] * T[1][2] + 128;
        (*v)[i * width + j] = r[i * width + j] * T[2][0] + g[i * width + j] * T[2][1] + b[i * width + j] * T[2][2] + 128;
    }
}
```

3. 设置 Y 为灰度值

只要将 RGB 三个通道都设置成上面得到的 Y 值即可，因为上面的变换确保了 Y 是

[0,255]的。笔者直接调用之后将提到的 WriteData 函数，如下图：

```
WriteData(fgoutput, finput, y, y, y);
```

注意上面后三个参数为 RGB，笔者直接用 Y 来代替即可生成灰度图片。

4. YUV 转换为 RGB

函数为 ConvertYUV2RGB。实现上面的 RGB 转为 YUV 之后，再将 YUV 转化成 RGB 是同理的，公式也在上面列出。

```
// the transformation matrix is as follows
float T[3][3] = {
                {1, 0, 1.403},
                {1, -0.343, -0.714},
                {1, 1.770, 0}
                };
int i, j;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        r[i * width + j] = y[i * width + j] * T[0][0] + (u[i * width + j] - 128) * T[0][1] + (v[i * width + j] - 128) * T[0][2];
        g[i * width + j] = y[i * width + j] * T[1][0] + (u[i * width + j] - 128) * T[1][1] + (v[i * width + j] - 128) * T[1][2];
        b[i * width + j] = y[i * width + j] * T[2][0] + (u[i * width + j] - 128) * T[2][1] + (v[i * width + j] - 128) * T[2][2];
    }
}
```

5. 改变 Y 值

函数为 ChangeY()。笔者采用改变 Y 值的方法是将所有 y 值除以 1.2。

```
void ChangeY(float *y)
{
    int i;
    for (i = 0; i < width * height; i++)
    {
        y[i] /= 1.2;
    }
}
```

6. 用 RGB 写入一张图片

函数为 WriteData()。首先将原图位图数据之前的数据都读入到新文件里。

```
// copy the data before the graphic data
char *infor = (char*)malloc(sizeof(char) * offsetData);
fseek(finput, 0, SEEK_SET);
fread(infor, sizeof(char), offsetData, finput);
fseek(foutput, 0, SEEK_SET);
fwrite(infor, sizeof(char), offsetData, foutput);
free(infor);
```

然后使用一个长度为上述 dataSizePerLine 的数组 dataLine 来一行一行写入数据。将三个数组 r、g、b 中数据从最下行读入 dataLine。

```
// write color information to the file
fseek(foutput, offsetData, SEEK_SET);
int dataSizePerLine = (width * 3 + 3) / 4 * 4;
unsigned char *dataLine = (unsigned char*)malloc(sizeof(char) * dataSizePerLine);

int i, j;
for (i = 0; i < height; i++)
{
    for (j = 0; j < dataSizePerLine; j++)
        dataLine[j] = 0;

    for (j = 0; j < width; j++)
    {
        // pay attention to the direction
        dataLine[j * 3 + 2] = (unsigned char)r[width * (height - i - 1) + j];
        dataLine[j * 3 + 1] = (unsigned char)g[width * (height - i - 1) + j];
        dataLine[j * 3] = (unsigned char)b[width * (height - i - 1) + j];
    }
    fwrite(dataLine, sizeof(char), dataSizePerLine, foutput);
}

free(dataLine);
```

四、实验环境及运行方法

编译环境：

gcc 6.3.0、Windows 11 Insider Preview 10.0.22483.1011 (rs_prerelease)

具体测试方法：

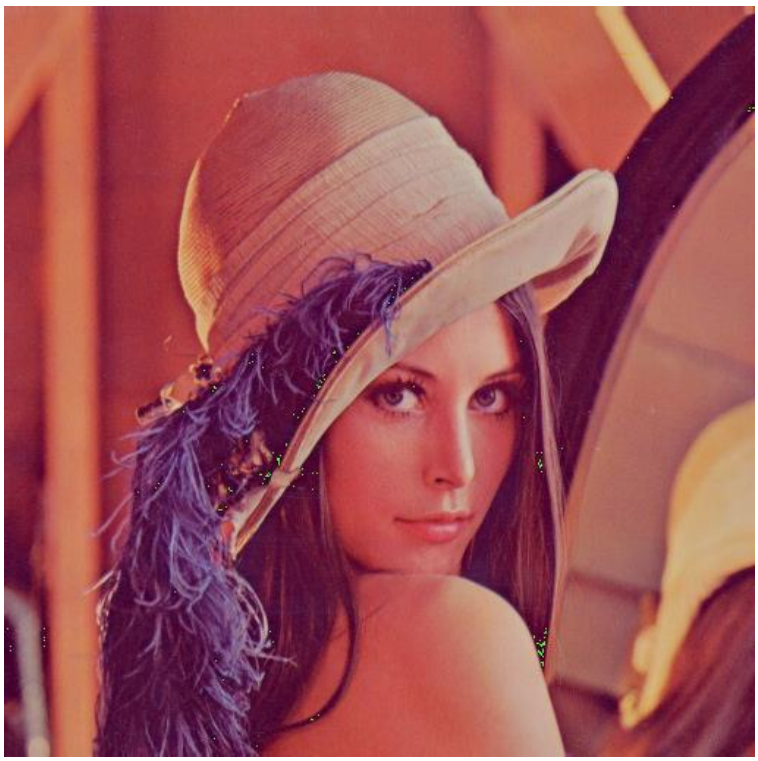
在命令行中输入命令 gcc main.c data.c -o main 并运行 main.exe 即可。在 main.c 第 11 行规定了输入图片的名称为 input.bmp，在第 19 行和 27 行分别规定了输出灰度图像的名称为 grayoutput.bmp 和彩色图片名 coloroutput.bmp。读者可以替换输入图片进行测试。

五、实验结果展示

输入图片：

输出灰度图片：

输出彩色图片：



六、心得体会

本次实验笔者成功降低了图片的亮度，但也产生了一些没有完全复原的点，离完全成功还有一点点距离。但是这次实验让我对 BMP 文件的格式更为了解，也对 RGB 和 YUV 颜色

空间有了跟深刻的认识，这次实验让我对图像信息处理产生了很大的兴趣，当最终生成出图片时感到非常大的成就感和满足感！