**实验报告**

专业：计算机科学与技术

姓名：薛柔

学号：3220104854

日期：2023/10/13

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图像的结构和文件的读写

2. 学会用计算机实现rgb和yuv色彩空间转化

3. 学会灰度图像的转化与表示

4. 能够改变Y值以达到改变图像亮度的目标

**二、实验内容和原理**

1. bmp文件的结构

bmp文件大体上分成四个部分，如下表所示。

|  |
| --- |
| 位图文件头BITMAPFILEHEADER |
| 位图信息头BITMAPINFOHEADER |
| 调色板Palette |
| 实际的位图数据ImageDate |

第一部分位图文件头BITMAPFILEHEADER的结构如下表所示：

|  |  |  |
| --- | --- | --- |
| 名称 | 大小（字节） | 含义 |
| bfType | 2 | 位图类别，根据不同的操作系统而不同，在Windows中，此字段的值总为‘BM’ |
| bfSize | 4 | BMP图像文件的大小 |
| bfReserved1 | 2 | 总为0 |
| bfReserved2 | 2 | 总为0 |
| bfOffBits | 4 | BMP图像数据的地 |

第二部分位图信息头BITMAPINFOHEADER的结构如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 大小（字节） | 字节 |
| biSize | 4 | 本结构的大小，根据不同的操作系统而不同，在Windows中，此字段的值总为28h字节=40字节 |
| biWidth | 4 | BMP图像的宽度，单位像素 |
| biHeight | 4 | 总为0 |
| biPlanes | 2 | 总为0 |
| biBitCount | 2 | BMP图像的色深，即一个像素用多少位表示，常见有1、4、8、16、24和32，分别对应单色、16色、256色、16位高彩色、24位真彩色和32位增强型真彩色 |
| biCompression | 4 | 压缩方式，0表示不压缩，1表示RLE8压缩，2表示RLE4压缩，3表示每个像素值由指定的掩码决定 |
| biSizeImage | 4 | BMP图像数据大小，必须是4的倍数，图像数据大小不是4的倍数时用0填充补足 |
| biXPelsPerMeter | 4 | 水平分辨率，单位像素/m |
| biYPelsPerMeter | 4 | 垂直分辨率，单位像素/m |
| biClrUsed | 4 | BMP图像使用的颜色，0表示使用全部颜色，对于256色位图来说，此值为100h=256 |
| biClrImportant | 4 | 重要的颜色数，此值为0时所有颜色都重要，对于使用调色板的BMP图像来说，当显卡不能够显示所有颜色时，此值将辅助驱动程序显示颜色 |

第三部分是调色板，真色彩图不需要调色板，三个byte分别表示一个像素的RGB分量，一共可以表示2563种颜色。但是一般的图不需要表示这么多颜色，假如一张图只要用16种颜色，我们可以用一个表：表中的每一行记录一种颜色的R、G、B值。这样当我们表示一个象素的颜色时，只需要指出该颜色是在第几行，即该颜色在表中的索引值。举个例子，如果表的第0行为255，0，0(红色)，那么当某个象素为红色时，只需要标明0即可。这张表就是调色板，我在实验中用的都是真色彩图，因此不需要调色板。

2. C语言读写bmp文件

引用库#include <stdlib.h>并用到以下函数：

FILE \*fopen(const char \*filename, const char \*mode)：将名为filename的文件打开（如果没有则新建一个），”rb” mode读取二进制文件，返回该文件的指针。

size\_t fwrite(const void \*ptr, size\_t size, size\_t nmemb, FILE \*stream)：将本地\*ptr指向的元素写入\*stream指向的文件，size\_t size为每个元素的大小（字节），size, size\_t nmemb为元素的个数。

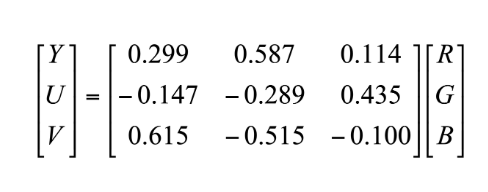
int fclose(FILE \*stream)：关闭\*steam指向的文件。

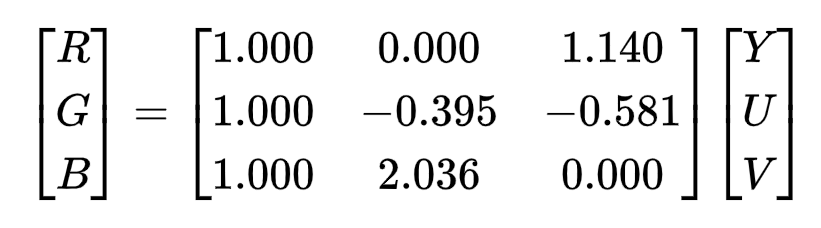
3. RGB和YUV

RGB色彩模式是工业界的一种颜色标准，是通过对红（R）、绿（G）、蓝（B）三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色的，RGB即是代表红、绿、蓝三个通道的颜色，这个标准几乎包括了人类视力所能感知的所有颜色，是运用最广的颜色系统之一。

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。

二者的转化公式如下：





**三、实验步骤与分析**

1. 读取一张彩色的bmp图

先引入库，重新定义1、2、4BYTE长度的类型名，再定义结构体和全局变量

结构体分量的具体含义在前面的表格中详细说明

1. #include <stdio.h>
2. #include <stdlib.h>
3. **typedef** unsigned **char** **BYTE**; // 1 byte
4. **typedef** unsigned **short** **WORD**; // 2 bytes
5. **typedef** unsigned **int** **DWORD**; // 4 bytes
6. **typedef** **int** **LONG**; // 4 bytes
7. //位图文件头
8. **typedef** **struct** tagBITMAPFILEHEADER {
9. **WORD** bfType;//指定文件类型，必须是0x424D，即字符串“BM”
10. **DWORD** bfSize;//指定文件大小，包括这14个字节。
11. **WORD** bfReserved1;
12. **WORD** bfReserved2;//为保留字，不用考虑
13. **DWORD** bfOffBits;
14. } BITMAPFILEHEADER;
16. //位图信息头
17. **typedef** **struct** tagBITMAPINFOHEADER{
18. **DWORD** biSize;
19. **LONG** biWidth;
20. **LONG** biHeight;
21. **WORD** biPlanes;
22. **WORD** biBitCount;
23. **DWORD** biCompression;
24. **DWORD** biSizeImage;
25. **LONG** biXPelsPerMeter;
26. **LONG** biYPelsPerMeter;
27. **DWORD** biClrUsed;
28. **DWORD** biClrImportant;
29. } BITMAPINFOHEADER;
31. //真色彩图的每个像素3BYTE，分别表示RGB的分量
32. **typedef** **struct** tagPIXEL {
33. **BYTE** red;
34. **BYTE** green;
35. **BYTE** blue;
36. } PIXEL;
38. //YUV的结构体
39. **typedef** **struct** tagYUV {
40. **double** Y;
41. **double** U;
42. **double** V;
43. } YUV;

46. //创建图的位图文件头和位图信息头
47. BITMAPFILEHEADER bmfh;
48. BITMAPINFOHEADER bmih;
49. **int** pixel\_num;//图像大小，一般不变

再读取bmp文件，将位图文件头保存在bmfh中，位图信息头保存在bmih中，具体的图像数据放在数组a[]中：

1. **int** main()
2. {
3. //打开文件并判断是否成功
4. **FILE** \*fp;
5. fp = fopen("img29.bmp", "rb"); // rb 打开一个二进制文件
6. **if** (!fp) {
7. printf("BMP Image Not Found!\n");
8. exit(0);
9. }
10. printf("Successfully open the image\n");
12. //读取文件头和图像信息头
13. fread(&bmfh, 14, 1, fp);  //文件头固定14BYTE，信息头固定40BYTE
14. fread(&bmih, 40, 1, fp);
16. //计算图像大小
17. pixel\_num=bmih.biHeight\*bmih.biWidth;
19. //创建数组读取位图数据
20. PIXEL a[pixel\_num];
22. fread(&a, **sizeof**(PIXEL), pixel\_num, fp);
23. fclose(fp);//关闭该图

2. RGB->YUV

利用公式转换RGB和YUV,并将Y控制在[0,255]之间

1. //RGB->YUV
2. YUV b[pixel\_num];
3. **for**(**int** i=0;i<pixel\_num;i++){
4. b[i].Y=a[i].red\*0.299+a[i].green\*0.587+a[i].blue\*0.114;
5. b[i].U=a[i].red\*-0.147+a[i].green\*-0.289+a[i].blue\*0.435;
6. b[i].V=a[i].red\*0.615+a[i].green\*-0.515+a[i].blue\*-0.100;
7. //将Y控制在[0,255]
8. **if**(b[i].Y<0)    b[i].Y=0;
9. **if**(b[i].Y>255)  b[i].Y=255;
10. }

3. 写入一个原图像的灰度图像

这里仍把灰度图看成真色彩图，因此不改变文件头。灰度图可以表示为RGB分量都等于Y的真色彩图。

1. //建立数组存储灰度图数据
2. PIXEL new\_pixel[pixel\_num];
3. **for**(**int** i=0;i<pixel\_num;i++){
4. //灰度图可以表示为RGB分量都等于Y的图
5. new\_pixel[i].red=new\_pixel[i].green=new\_pixel[i].blue=b[i].Y;
6. }
8. //打开一个bmp空白图像并写入数据
9. fp = fopen("gray.bmp", "wb");
10. // fwrite(数据块首地址,元素大小, 元素个数,文件指针)
11. //与原来的图相比，文件头相同
12. fwrite(&bmfh, 14, 1, fp);//文件头
13. fwrite(&bmih, 40, 1, fp);//信息头
14. //写入位图数据
15. fwrite(&new\_pixel, 3, pixel\_num, fp);
16. fclose(fp);

4. 改变亮度数据（即改变Y值并转换回RGB）

YUV转RGB的公式在原理篇中列出。

注意需要控制转换后的RGB在[0,255]之间，否则会出现数值溢出，图像上的表现为高饱和的噪点。

1. //降低亮度，将Y值-30
2. **for**(**int** i=0;i<pixel\_num;i++){
3. b[i].Y=b[i].Y-30;
5. //YUV->RGB
6. **int** temp\_red=b[i].Y+1.140\*b[i].V;
7. **int** temp\_green=b[i].Y-0.395\*b[i].U+0.581\*b[i].V;
8. **int** temp\_blue=b[i].Y+2.000\*b[i].U;
9. //防止溢出
10. **if**(temp\_red>255) temp\_red=255;
11. **if**(temp\_red<0) temp\_red=0;
12. **if**(temp\_green>255) temp\_green=255;
13. **if**(temp\_green<0) temp\_green=0;
14. **if**(temp\_blue>255) temp\_blue=255;
15. **if**(temp\_blue<0) temp\_blue=0;
17. new\_pixel[i].red=temp\_red;
18. new\_pixel[i].green=temp\_green;
19. new\_pixel[i].blue=temp\_blue;
20. }

5. 将改变了亮度的图像数据重写为图像文件

该步骤同写入灰度图。

1. //打开一个bmp空白图像并写入数据
2. fp = fopen("dark.bmp", "wb");
3. fwrite(&bmfh, 14, 1, fp);//文件头
4. fwrite(&bmih, 40, 1, fp);//信息头
5. //写入位图数据
6. fwrite(&new\_pixel, 3, pixel\_num, fp);
7. fclose(fp);

**四、实验环境及运行方法**

* 实验环境：Windows 11 系统

gcc 10.3.0 (tdm64-1) x86\_64-w64-mingw32

* 运行方法：将.exe文件和img29.bmp放在同一个目录下，运行.exe文件，会得到三张新的图像，分别命名为gray.bmp，dark.bmp，light.bmp，即为转化后的灰度图、亮度变为原来的一半的图和亮度变为原来的1.2倍的图。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!”。

如果运行不成功，可以将文件夹中的.c源文件重新编译运行。

如果想得到亮度更高/更低的图像，可以在源文件中改变与Y相加减的系数。

**五、实验结果展示**

输入图像：



图1 原图img29.bmp

灰度图：



图2 灰度图gray.bmp

Y值变为原来的一半：



图3 亮度降低的图dark.bmp

Y值变为原来的1.2倍：



图4 亮度升高的图light.bmp

**六、心得体会**

首先本人设计的程序还有很多不足之处，没有考虑图像宽度并非4的倍数以及使用调色板的bmp图像的情况，对齐并完成相应的功能需要额外的代码。因此本程序只适用于部分bmp图像，比如附件img29.bmp（320\*240像素）。

实验中也遇到了很多困难，我本来想利用调色板制作256色的灰度图，然而此方法需要改动头文件的大量数据，我反复尝试后输出结果仍然是bmp文件损坏。在助教的指导下，我改变方法，用真色彩图的编码格式得到灰度图，得到了预期的输出结果。

希望在后续的实验中，我能完成功能更完备的代码。