**实验报告**

专业：计算机科学与技术

姓名：

学号：

日期：2022/10/08

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**

完成下述实验任务：

1.以二进制形式读取bmp文件头信息及数据信息

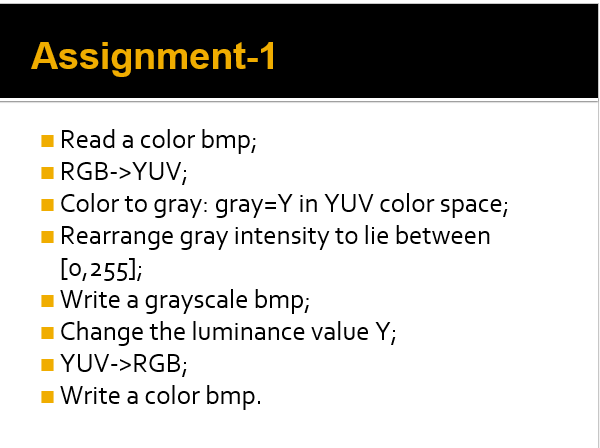
2.将bmp中数据格式由RGB转化为YUV格式，并使其数值分布于0-255间

3.调整灰度，输出原始bmp文件的灰度图像

4.尝试改变YUV中Y的亮度值

5.将YUV文件格式转换为RGB

6.重新写入新图片文件



**二、实验内容和原理**

Bmp图像信息的贮存，往往有固定的格式，用二进制的方式打开bmp文件，可以设置相对映的数据结构接受文件头部信息和数据头部信息，进行数据操作；根据bmp文件自带的数据起始区域偏移量，可便捷访问bmp数据部分。

Bmp图像往往贮存RGB颜色信息，根据特定转换公式，可以得到相对应的YUV值，保留Y值并重新赋予RGB信息，即可获得灰度图片，将原始图片信息载入新文件中，则可生成灰度图。

先调整Y对应值，再通过公式变换将YUV转换为RGB格式，则可以得到不同的亮度图像信息，再配合原始图片信息载入新文件，则可以生成不同亮度bmp图像。

**三、实验步骤与分析**

1.建立存储图像信息的数据结构（头文件）与bmp附带信息基本一一对应。

typedef unsigned char bit;

typedef unsigned short word;

typedef unsigned long dword;

struct tagBITMAPFILEHEADER

{

word bftype; //文件类型

dword bfsize; //文件大小（字节为单位）

word bfreserved1; //保留，必须设为0

word bfreserved2; //保留，必须设为0

dword bfoffbits; //到实际图像实际开始的偏移量

};

struct tagBITMAPINFOHEADER

{

dword bisize; //结构所需字节数

long biwidth; //图像宽度（像素为单位）

long biheight; //图像长度--正值对映图像倒立

word biplanes; //位面数，常为1

word bibitcounts; //比特/像素比

dword bicompression; //压缩类型

dword bisizeimage; //图像大小

long bixpelspermeter; //水平分辨率

long biypelspermeter; //垂直分辨率

dword biclrused; //颜色索引数

dword biclrimportant; //重要影响颜色索引数

};

2.以二进制方式读取bmp图像原始信息，将信息分类别贮存至上述定义数据结构中，并通过偏移量读取RGB数据区域，判断数据输出格式为0-1或者0-255.

//提取bmp文件头部信息

FILE\* fp;

fp = fopen("start.bmp", "rb");

fread(&(file.bftype), 1, sizeof(word), fp);

fread(&(file.bfsize), 1, sizeof(dword), fp);

fread(&(file.bfreserved1), 1, sizeof(word), fp);

fread(&(file.bfreserved2), 1, sizeof(word), fp);

fread(&(file.bfoffbits), 1, sizeof(dword), fp);

fread((&information), sizeof(tagBITMAPINFOHEADER), 1, fp);

//获取bmp大小像素信息

picwidth = information.biwidth;

picheight = information.biheight;

//接收bmp图像数据信息

bit\* rgbpic = (bit\*)malloc(picwidth \* picheight \* 3 \* sizeof(bit));

fseek(fp, file.bfoffbits, SEEK\_SET);

fread(rgbpic, sizeof(bit), picwidth \* picheight \* 3, fp);

short\* yuvpic=(short\*)malloc(picwidth\*picheight\*sizeof(short)\*3);

//测试获得的rgb是否为0-255区间数

//printf("%d", 100 \*rgbpic[89]);

3.利用公式将RGB格式转换为YUV格式，并保留Y值，赋予RGB数据中，创造灰度图。

//利用公式将RGB类型转换为YUV类型

for (i = 0;i < picwidth \* picheight;i++)

{

yuvpic[3 \* i] = 0.11 \* rgbpic[3 \* i] + 0.59 \* rgbpic[3 \* i + 1] + 0.3 \* rgbpic[3 \* i + 2];

yuvpic[3 \* i + 1] = 0.493 \* (rgbpic[3 \* i] - yuvpic[3 \* i]);

yuvpic[3 \* i + 2] = 0.877\*(rgbpic[3 \* i + 2] - yuvpic[3 \* i]);

}

//YUV灰度调整，输出灰度bmp图像

bit\* rgbpic2 = (bit\*)malloc(picwidth \* picheight \*3 \* sizeof(bit));

for (long i = 0; i < picheight \* picwidth; i++)

{

rgbpic2[3 \* i] = yuvpic[3 \* i];

rgbpic2[3 \* i + 1] = yuvpic[3 \* i];

rgbpic2[3 \* i + 2] = yuvpic[3 \* i];

}

4.通过调整Y值，改变图像亮度，并重新利用公式将YUV数据转换为RGB数据，以用于填入新图像信息中。

//改变bmp图像亮度（列举增大减小两种特殊情况）

//缩小亮度

for (i = 0;i < picheight \* picwidth;i++)

yuvpic[3 \* i] = yuvpic[3 \* i] / 1.08;

//增大亮度

for (i = 0;i < picheight \* picwidth;i++)

{

if (yuvpic[3 \* i] \*1.12<=255)

yuvpic[3 \* i] = yuvpic[3 \* i] \* 1.12;

else

yuvpic[3\*i] = 255;

}

//将YUV格式转化为RGB格式

//若YUV相应转换值大于255，则保留255

//若YUV相应转换值小于0，则保留0

bit\* rgbpic4 = (bit\*)malloc(picheight\* picwidth \* sizeof(bit) \* 3);

for (i = 0; i < picwidth \* picheight; i++)

{

if (yuvpic[3 \* i] + 2.0284 \* yuvpic[3 \* i + 1] < 0)

rgbpic4[3 \* i] = 0;

else if (yuvpic[3 \* i] + 2.0284 \* yuvpic[3 \* i + 1] > 255)

rgbpic4[3 \* i] = 255;

else

rgbpic4[3 \* i] = yuvpic[3 \* i] + 2.0284 \* yuvpic[3 \* i + 1];

if (yuvpic[3 \* i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* y uvpic[3 \* i + 2] < 0)

rgbpic4[3 \* i + 1] = 0;

else if (yuvpic[3 \* i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.579 8 \* yuvpic[3 \* i + 2] > 255)

rgbpic4[3 \* i + 1] = 255;

else

rgbpic4[3 \* i + 1] = yuvpic[3 \* i] - 0.3781 \* yuvpic [3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2];

if (yuvpic[3 \* i] + 1.1403 \* yuvpic[3 \* i + 2] < 0)

rgbpic4[3 \* i + 2] = 0;

else if (yuvpic[3 \* i] + 1.1403 \* yuvpic[3 \* i + 2] > 255)

rgbpic4[3 \* i + 2] = 255;

else

rgbpic4[3 \* i + 2] = yuvpic[3 \* i] + 1.1403 \* yuvpic [3 \* i + 2];

}

5.将转换后的图片信息写入新图像中，并生成图像。

//写入相对印要求图片

//建立新图像

FILE\* fp2;

fp2 = fopen("output2.bmp", "wb");

//写入图像信息

fwrite(&(file.bftype), 1, sizeof(file.bftype), fp2);

fwrite(&(file.bfsize), 1, sizeof(file.bfsize), fp2);

fwrite(&(file.bfreserved1), 1, sizeof(file.bfreserved1), fp2);

fwrite(&(file.bfreserved2), 1, sizeof(file.bfreserved2), fp2);

fwrite(&(file.bfoffbits), 1, sizeof(file.bfoffbits), fp2);

fwrite(&information, 1, sizeof(information), fp2);

fwrite(rgbpic2, sizeof(bit), picwidth \* picheight \* 3, fp2);

//释放内存，关闭图像

free(rgbpic2);

fclose(fp2);

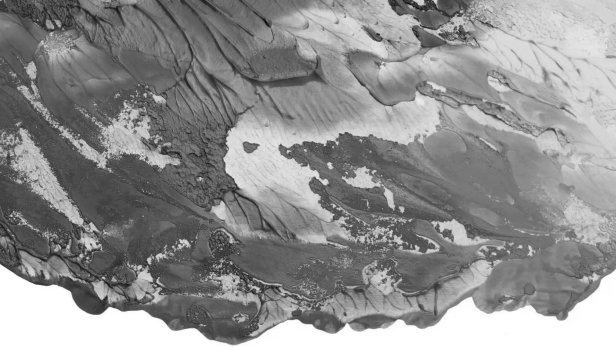
**四、实验环境及运行方法**

用C语言编写，vs正常编译环境运行，其中需要以下语句忽略vs对fopen函数的安全性中断。

#pragma warning(disable:4996) /忽略vs对fopen的安全性错误

对于实验结果，点击运行即可。在调试过程中，会通过printf输出函数检测RGB类型以及YUV类型的输出数据，为0-1还是0-255范围。最后输出三张图片，分别为灰度图，增大亮度图与减小亮度图。

1. **实验结果展示**



原始图片 灰度图



亮度减小图片 亮度增大图片

**六、心得体会**

不得不说，对于图像的C语言处理方法，还是很多需要自学的。虽然老师在上课时已经详细讲解了一些转换公式，和照片数据存储方式，但这些与编程语言的结合，对于第一次作业而言，还是颇具挑战性。

在顿悟用数据结构保存bmp图像内的数据内容，对于图像的操作就和上课的内容开始紧密相连，通过偏移量可以快速地获得数据存放地址，然后根据其通常的大小字节，逐一取出并进行操作，在bmp信息的理解之上，便完成了YUV的转化，灰度图的生成，自由的调节亮度值大小。

本来对于图像的编辑完全依赖于特定的工具，现在居然可以使用C语言来实现一些简单的功能，这更加是对后续的学习充满期待，期待用代码操作代替PS那日的到来。