

# 浙江大学实验报告

专业：计算机科学与技术

姓名：颜晗

学号：3200105515

日期：2021/10/22

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

目的：了解 bmp 文件的基本结构；掌握使用程序语言（例如 C 语言）读写 bmp 文件的方法；了解图像文件的几种色彩表示方法并学会它们间的相互转化。

要求：使用 C 语言完成实验内容，独立编写 bmp 图像文件操作的各函数并完成操作后输出图像文件，不允许使用已有的图像操作标准库。

## 二、实验内容和原理

使用编程软件完成如下内容：1.读取一个彩色 bmp 图像文件。2.将 bmp 图片 rgb 色彩改为 yuv 表示。3.将彩色图改为灰度图。4.将灰度图的灰度重新调整到 0-255 之间 5.将灰度图输出到独立的 bmp 文件中。6.改变亮度值 y。7.将 yuv 色彩重新转变为 rgb。8.将彩色 rgb 图像输出到一个独立的 bmp 文件中。

实验原理：bmp 图像的本质依旧是文件，完全可以将文本文件操作的方法用到图像文件中，熟悉 bmp 文件的结构便可按序将二进制数据读取、修改与写入。

## 三、实验步骤与分析

大部分操作都是读取像素点或调色板信息进行逐一计算，除计算公式外代码基本相同，后面几项步骤将省略代码，仅介绍计算方法与一些注意点。

### 1. 基本结构的定义

如下所示代码为此次实验中定义的结构体，前三个为标准 bmp 文件中共同包含的信息结构体，而结构体“myBmpFile”则是本人定义用于在程序中储存 bmp 文件信息的结构体，其中文件头与信息头的大小固定，可直接使用结构体，而调色板与具体的 bmp 图像信息的大小不固定，暂时指定一个指针，待读取到文件后可申请内存。

```
1. typedef struct
2. {
3.     unsigned short    bfType;
4.     unsigned long     bfSize;
5.     unsigned short    bfReserved1;
```

```

6.  unsigned short    bfReserved2;
7.  unsigned long     bfOffBits;
8. }myBitMapFileHeader; //位图文件头
9.
10. typedef struct
11. {
12.  unsigned long     biSize;
13.  long             biWidth;
14.  long             biHeight;
15.  unsigned short    biPlanes;
16.  unsigned short    biBitCount;
17.  unsigned long     biCompression;
18.  unsigned long     biSizeImage;
19.  long             biXPelsPerMeter;
20.  long             biYPelsPerMeter;
21.  unsigned long     biClrUsed;
22.  unsigned long     biClrImportant;
23. } myBitMapInfoHeader; //位图文件头
24.
25. typedef struct
26. {
27.  unsigned char rgbBlue; // 该颜色的蓝色分量
28.  unsigned char rgbGreen; // 该颜色的绿色分量
29.  unsigned char rgbRed; // 该颜色的红色分量
30.  unsigned char rgbReserved;
31. } myRgbQuad; // 调色板
32.
33. typedef struct
34. {
35.  myBitMapFileHeader BFH; //文件头
36.  myBitMapInfoHeader BIH; //信息头
37.  myRgbQuad *Plaquette; // 调色板指针
38.  unsigned char* imageData;
39. }myBmpFile;

```

## 2. 读取一个彩色 bmp 图像文件。

由于代码注释已将重要信息解释完全，此处仅简单描述。

使用文件读取函数 `fread`，可将文件中的指定大小的信息转移到指定空间中，而为了代码的可读性，以及处理信息时的层次与整齐性，需要将整个文件拆开来读取，也是由于文件中各信息的类型不同，为了方便与正确地表示信息，需要分开读取。

还需注意内存和指针的使用，函数结束时会释放局部变量的内存，存储信息时可能会遇

见信息突然改变的情况，就是使用的内存在实际上已不属于程序主动管理，被某些项目占用。

另外，读取时遇见一个奇怪的 bug，文件头结构体在 `sizeof()` 时值总是为 16 字节，导致后续写文件时出错，只能将文件头各元素分开读取。

```
1. void ReadBmp(myBmpFile *bmpImg, const char* path)
2. {
3.     FILE *pFile;
4.     myBitMapFileHeader bmpFileHeader;
5.     myBitMapInfoHeader bmpInfoHeader;
6.     myRgbQuad* quad=NULL;
7.     unsigned char *bmpData;
8.     // 基本结构体的声明，作为中间变量用于转移信息。
9.
10.    pFile = fopen(path, "rb");
11.    if(!pFile) perror("Failed to open file");//打开文件
12.
13.
14.    fread(&bmpFileHeader.bfType, 2, 1, pFile);
15.    fread(&bmpFileHeader.bfSize, 4, 1, pFile);
16.    fread(&bmpFileHeader.bfReserved1, 2, 1, pFile);
17.    fread(&bmpFileHeader.bfReserved2, 2, 1, pFile);
18.    fread(&bmpFileHeader.bfOfffBits, 4, 1, pFile);
19.    bmpImg->BFH=bmpFileHeader; // 读取位图文件头
20.
21.
22.    fread(&bmpInfoHeader, sizeof(myBitMapInfoHeader), 1, pFile);
23.    bmpImg->BIH=bmpInfoHeader;// 读取位图信息头
24.
25.    /* 由于 8 位和 24 位存在有无调色板以及像素颜色表示方法的区别，故所有函数均将它们分开处理 */
26.    if(bmpInfoHeader.biBitCount==8){ // 8 位 bmp 图像
27.        fseek(pFile, bmpInfoHeader.biSize-40, SEEK_CUR);
28.        // 由于寻找的部分图信息头结构体大小不一定为 40，故需要重定位文件指针寻找所需信息
29.        quad=(myRgbQuad*)malloc(sizeof(myRgbQuad)*256);
30.        bmpImg->Palette=(myRgbQuad*)malloc(sizeof(myRgbQuad)*256);
31.        fread(quad, sizeof(myRgbQuad)*256, 1, pFile);
32.        memcpy(bmpImg->Palette, quad, sizeof(myRgbQuad)*256);
33.        /* 转移调色板信息 */
34.
35.        bmpData=(unsigned char*)malloc(sizeof(unsigned char)*bmpInfoHeader.biSizeImage);
36.        bmpImg->imageData=(unsigned char*)malloc(sizeof(unsigned char)*bmpInfoHeader.biSizeImage);
```

```

37.
38.  fread(bmpData,sizeof(unsigned char)*bmpInfoHeader.biSizeImage,1,pFile
    );
39.
40.  memcpy(bmpImg->imageData,bmpData,sizeof(unsigned char)*bmpInfoHeader.
    biSizeImage);
41.  /*转移位图像素信息*/
42.
43.  bmpImg->BFH.bfSize=54+bmpImg->BIH.biSizeImage+sizeof(myRgbQuad)*256;
44.  //重新确定文件大小
45. }
46.  else if(bmpInfoHeader.biBitCount==24){//24 位图的处理
47.  fseek(pFile,bmpInfoHeader.biSize-40,SEEK_CUR); //重定位
48.  bmpImg->BIH.biSize=40;
49.  bmpImg->Palette = quad;//调色板指针为空。
50.
51.  bmpData=(unsigned char*)malloc(sizeof(unsigned char)*bmpInfoHeader.bi
    SizeImage);
52.  bmpImg->imageData=(unsigned char*)malloc(sizeof(unsigned char)*bmpInf
    oHeader.biSizeImage);
53.  fread(bmpData,bmpInfoHeader.biSizeImage,1,pFile);
54.  memcpy(bmpImg->imageData,bmpData,sizeof(unsigned char)*bmpInfoHeader.
    biSizeImage);
55.  /*转移像素信息*/
56.  bmpImg->BFH.bfSize=54+bmpImg->BIH.biSizeImage;
57. }
58. fclose(pFile);
59.}

```

### 3. 将 bmp 图片 rgb 色彩改为 yuv 表示。

两种颜色表示有相应的公式进行转换，只要以像素点为单位进行数学计算即可。需要注意的有：1.bmp 图像一行像素的字节数不一定为 width\*3，而是大于 width\*3 的最小的 4 的倍数；2.虽然二进制表示相同，但是 u, v 两个分量是有符号数，由于不需要输出 yuv 图像，可以直接当无符号，但是后面进行转回 rgb 时要进行类型转换。

```

1. void rgb2yuv(myBmpFile *bmpImg)
2. {
3.  int i,j;
4.  unsigned char r,g,b;
5.
6.  if(bmpImg->Palette){//8 位图像修改调色板信息
7.  for(i=0;i<256;i++)
8.  {
9.  r=bmpImg->Palette[i].rgbRed;

```

```

10.     g=bmpImg->Palette[i].rgbGreen;
11.     b=bmpImg->Palette[i].rgbBlue;
12.     //存储分量信息，便于计算，精简表达式
13.
14.     if((bmpImg->Palette[i].rgbRed=(unsigned char)(0.299*r+0.587*g+0.114
        *b))>255)
15.         bmpImg->Palette[i].rgbRed=(unsigned char)255;//最大设为 255
16.     bmpImg->Palette[i].rgbGreen=(char)(-0.147*r-0.289*g+0.435*b);
17.     bmpImg->Palette[i].rgbBlue=(char)(0.615*r-0.515*g-0.100*b);
18.     //三个分量分别计算
19. }
20. }
21. else{//24 位图像需要直接在图像像素信息处修改，其余与 8 位同
22.     long height=bmpImg->BIH.biHeight;
23.     long width=bmpImg->BIH.biWidth;
24.     int offset=0;
25.     if(width*3%4!=0){
26.         offset=4-width*3%4;
27.     }
28.     long width_=width*3+offset; //计算一行字节数
29.
30.     for(i=0;i<height-1;i++){
31.         for(j=0;j<width-1;j++){
32.             r=bmpImg->imageData[i*width_+j*3];
33.             g=bmpImg->imageData[i*width_+1+j*3];
34.             b=bmpImg->imageData[i*width_+2+j*3];
35.             if((bmpImg->imageData[i*width_+j*3]=(unsigned char)(0.299*r+0.587*g+
                0.114*b))>255)
36.                 bmpImg->imageData[i*width_+j*3]=(unsigned char)255;
37.             bmpImg->imageData[i*width_+1+j*3]=(char)(-0.147*r-0.289*g+0.435*b);
38.             bmpImg->imageData[i*width_+2+j*3]=(char)(0.615*r-0.515*g-0.100*b);
39.
40.         }
41.     }
42. }
43. }

```

#### 4. 将彩色图改为灰度图。

灰度实际上就是上一步所计算的 y 分量，只需要将 u, v 两个分量都变成 y，图像便变成了灰色，由于下面还需要由 yuv 转回 rgb，此处需要另一个结构体存灰度信息，否则会丢失 u, v 分量。实现方法依旧是循环遍历计算。

```
1. void yuv2Gray(myBmpFile *Bgray,myBmpFile *bmpImg)
2. {
3.     Bgray->BFH=bmpImg->BFH;
4.     Bgray->BIH=bmpImg->BIH;
5.
6.     int i,j;
7.
8.     if(bmpImg->Palette){//8 位图处理, 依旧只需要调整调色板
9.         Bgray->Palette=(myRgbQuad*)malloc(sizeof(myRgbQuad)*256);
10.        Bgray->imageData=(unsigned char *)malloc(sizeof(unsigned char)*bmpImg
            ->BIH.biSizeImage);
11.        for(i=0;i<256;i++)
12.        {
13.            if((Bgray->Palette[i].rgbRed=bmpImg->Palette[i].rgbRed)>255)
14.                Bgray->Palette[i].rgbRed=(unsigned char)255;
15.            Bgray->Palette[i].rgbGreen=bmpImg->Palette[i].rgbRed;
16.            Bgray->Palette[i].rgbBlue=bmpImg->Palette[i].rgbRed;
17.            //分量全部赋值 y
18.        }
19.        memcpy(Bgray->imageData,bmpImg->imageData,sizeof(unsigned char)*bmpIm
            g->BIH.biSizeImage);
20.    }
21.    else{//24 位图像处理
22.        Bgray->Palette=NULL;
23.        Bgray->imageData=(unsigned char *)malloc(sizeof(unsigned char)*bmpImg
            ->BIH.biSizeImage);
24.
25.        long height=bmpImg->BIH.biHeight;
26.        long width=bmpImg->BIH.biWidth;
27.        int offset=0;
28.        if(width*3%4!=0){
29.            offset=4-width*3%4;
30.        }
31.        long width_=width*3+offset;//计算一行字节数
32.
33.        for(i=0;i<height-1;i++){
34.            for(j=0;j<width-1;j++){
35.                if((Bgray->imageData[i*width_+j*3]=bmpImg->imageData[i*width_+j*3])
                    >255)
36.                    Bgray->imageData[i*width_+j*3]=(unsigned char)255;
37.                Bgray->imageData[i*width_+1+j*3]=bmpImg->imageData[i*width_+j*3];
38.                Bgray->imageData[i*width_+2+j*3]=bmpImg->imageData[i*width_+j*3];
39.            }
40.        }
```

```
41. }  
42. }
```

## 5. 将灰度图的灰度重新调整到 0-255 之间

即使用线性映射使原本处于某一区间的灰度值映射到[ 0, 255]，借助公式 “ $(\text{灰度值} - \text{最小值}) / (\text{最大值} - \text{最小值}) * 255$ ” 即可实现。但是由于所寻找的图像大都具有白色与黑色，已经覆盖全部区间，灰度调整的效果并不明显。

```
1. void Rearrange(myBmpFile *Bgray)  
2. {  
3.     int i,j;  
4.     unsigned char max,min;  
5.  
6.     if(Bgray->Palette){  
7.         max=0;min=Bgray->Palette[0].rgbRed;  
8.         for(i=0;i<256;i++){  
9.             if(Bgray->Palette[i].rgbRed>max)  
10.                 max=Bgray->Palette[i].rgbRed;  
11.             if(Bgray->Palette[i].rgbRed<min)  
12.                 min=Bgray->Palette[i].rgbRed;  
13.         }//寻找最大值与最小值  
14.         unsigned char scope=max-min;//所处区间长度  
15.  
16.         for(i=0;i<256;i++){  
17.             {  
18.                 Bgray->Palette[i].rgbRed=(unsigned char)(255*(Bgray->Palette[i].rgbRed-min)/scope);  
19.                 Bgray->Palette[i].rgbGreen=Bgray->Palette[i].rgbRed;  
20.                 Bgray->Palette[i].rgbBlue=Bgray->Palette[i].rgbRed;  
21.             }  
22.  
23.         }  
24.     }else{  
25.         long height=Bgray->BIH.biHeight;  
26.         long width=Bgray->BIH.biWidth;  
27.         int offset=0;  
28.         if(width*3%4!=0){  
29.             offset=4-width*3%4;  
30.         }  
31.         long width_=width*3+offset;//计算一行字节数  
32.  
33.         max=0;min=Bgray->imageData[0];  
34.         for(i=0;i<height-1;i++){  
35.             for(j=0;j<width-1;j++){
```

```

36.     if(Bgray->imageData[i*width_+j*3]>max)
37.         max=Bgray->imageData[i*width_+j*3];
38.     if(Bgray->imageData[i*width_+j*3]<min)
39.         min=Bgray->imageData[i*width_+j*3];
40. }//寻找最大值与最小值
41. }
42. unsigned char scope=max-min; //所处区间长度
43.
44.
45. for(i=0;i<height-1;i++){
46.     for(j=0;j<width-1;j++){
47.         Bgray->imageData[i*width_+j*3]=(unsigned char)(255*(Bgray->imageData[i*width_+j*3]-min)/scope);
48.         Bgray->imageData[i*width_+1+j*3]=Bgray->imageData[i*width_+j*3];
49.         Bgray->imageData[i*width_+2+j*3]=Bgray->imageData[i*width_+j*3];
50.     }
51. }
52.
53. }
54.
55. }

```

## 6. 将灰度图输出到独立的 bmp 文件中。

同读取，确定了 bmp 结构体只需确定输入输出源，将 fread 函数改为 fwrite 函数，使信息依次转移到指定路径的文件中，即可生成 bmp 文件。

## 7. 改变亮度值 y 并将 yuv 色彩重新转变为 rgb。

转换回去原理依旧是数学公式，如第 3 步所示，u，v 实际上是有符号数，在计算时需要类型转换，否则结果差距会很大，导致图像色彩混乱。

亮度值直接在公式中将 y 增加 20 再参与运算即相当于改变了。只需要在“yuv2rgb”函数内的计算部分改变变化值就可以改变输出的效果。

## 8. 将彩色 rgb 图像输出到一个独立的 bmp 文件中。

同第 6 步，设置路径使用 WriteBmp 函数即可。

## 四、实验环境及运行方法







本人使用 dev-c++ 5.11 版 C 语言工程运行代码，如果电脑上安装有 dev-c++，单击文件“bmpwork.dev”即可打开工程文件，点击编译运行的图标或按 f11 即可。

使用 vscode 运行的方法：打开程序所处文件夹（包含 bmp.h、bmp.c 以及 main.c 文件），在终端输入“gcc bmp.c main.c -o main”，敲击 enter 键后即可编译生成 main.exe，再直接在终端输入“.\main”或在文件夹中打开 main.exe 即可运行程序得到结果。



注意：在运行程序前，保证待操作的 bmp 文件在对应的路径中。

## 五、实验结果展示

原图	灰度图	Yuv(Y+20)->rgb
		
		

上面 3 张为 24 位图，下面 3 张为 8 位图

下面还有部分实验过程的内容输出，仅供参考。

未改变 y 值的图像 可见还原度极高	另一张，存在部分失真	未进行类型转换，使用无符号 u,v
		

## 六、心得体会

本次实验我们独立完成了 **bmp** 文件的部分基本操作，对 **bmp** 文件结构和图像的色彩表示有了一个基本的认识，见识到了人类智慧的伟大成果。在图像信息占据了我们生活中相当一部分的当下，对图像有深入了解势必对我们处理信息有巨大帮助，而如何更好的存储，使用图像信息，也是我们仍需要思考的问题。让人们眼中的世界更加丰富多彩，让人们足不出户也可以见识到广阔的世界，这是图像处理正在做的，也是仍要做的。

当然，在使用 C 语言实现功能时，也增强了调试复杂代码的能力，读取修改文件的过程虽然痛苦，但是完成任务看见自己写出的漂亮图片时成就感油然而生。