

浙江大学实验报告

专业：计算机科学与技术

姓名：沈一芑

学号：322010827

日期：2023/10/14

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____

1. 实验目的和要求

1.1. 实验目的

- 了解 BMP 文件的存储方式与结构组成
- 掌握代码读写 BMP 文件的能力
- 了解 RGB 与 YUV 两种色彩空间，并能实现两者的转换
- 利用 YUV 色彩空间的特性调整图像效果
- 了解灰度图的原理并实现彩色图与灰度图的双向转换

1.2. 实验要求

- 代码读取 BMP 文件，将 RGB 转为 YUV
- 生成读入 BMP 文件的灰度图
- 改变 BMP 文件的 Y 值（明度）
- 输出调整后的彩色 BMP 图像

2. 实验内容和原理

2.1. 实验原理

- BMP 文件的存储格式为四部分：BITMAPFILEHEADER, BITMAPINFOHEADER, Palette, Bitmap Data。

其构成如下表：

```
• typedef struct{  
•     BITMAPFILEHEADER;  
•     BITMAPINFOHEADER;  
•     PALETTE;  
•     BITMAP DATA;  
• }BMPfile;
```

- 各个部分分别记录了 BMP 文件的整体信息、图像基本信息、调色板、像素点数据。其中 BITMAPFILEHEADER 和 BITMAPINFOHEADER 固定占据 14 字节和 40 字节；Palette 调色板只在位深为 1/2/4/8 的 BMP 文件中存在，8 位 BMP 图的调色板占 256*4 字节。而像素点数据则与图片的尺寸大小有关系。对于 RGB 色彩空间的 24 位 BMP 文件，其每一个像素点按顺序存储了 B\G\R 三字节的色彩数据。

- BITMAPFILEHEADER 具体构成如下表：

```
• typedef struct tagBITMAPFILEHEADER{
•     WORD bfType           //type of file, should be 0X4D42
•     DWORD bfSize          //size of whole BMP file
•     WORD bfReserved1      //0
•     WORD bfReserved2      //0
•     DWORD bfOffBits       //offbits from start of BMP file to data
part
• }BMFH;
```

- BITMAPINFOHEADER 具体构成如下表：

```
• typedef struct tagBITMAPINFOHEADER{
•     DWORD biSize          //size of BMIH, 40
•     LONG biWidth          //width of image
•     LONG biHeight         //height of image
•     WORD biPlanes         //planes, 1
•     WORD biBitCount       //bits per pixel 1/2/4/8/16/24/32
•     DWORD biCompression  //way of compression
•     DWORD biSizeImage     //size of image part
•     LONG biXPelsPerMeter
•     LONG biYPelsPerMeter
•     DWORD biClrUsed
•     DWORD biClrImportant
• }BMIH;
```

- Palette 具体构成如下表：

```
• typedef struct {
•     BYTE rgbBlue
•     BYTE rgbGreen
•     BYTE rgbRed
•     BYTE rgbReserved
• }RGBQUAD;
```

2.2. 实验内容

- 读入一张 BMP 文件，即按顺序将对应内容读入到对应部分。
- 生成灰度图，即生成一张 8 位深的 BMP 文件，其中 data 部分仅保留 RGB 转 YUV 后的 Y 值(luminance, 明度)。RGB 与 YUV 色彩空间转换公式如下图：

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.148 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- 改变明度，即改变 Y 值。即先将 24 位图色彩空间转为 YUV，修改 Y 值后重新将色彩空间转为 RGB，写出文件。YUV 与 RGB 色彩空间转换公式如下图：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

3. 实验步骤与分析

3.1. 准备工作

- 采取紧凑存储：

```
• #pragma pack(1)
//to make the storage of struct consecutive, instead of align in 4 bytes
(DWORD)
```

- Typedef 数据类型：

```
• typedef unsigned short WORD;           //2 bytes
• typedef unsigned char BYTE;            //1 bytes
• typedef unsigned int DWORD;            //4 bytes unsigned
• typedef int LONG;                      //4 bytes
```

- 定义结构体 IMAGE，作为 BMPfile。其中参数如注释所示：

```
• typedef struct{
•     BMFH bmfh;
•     BMIH bmih;
•     RGBQUAD rgbquad[256];
•     BYTE *data;
•     int LineBytes, WidthBMP, HeightBMP, SizeImageBMP;
•     //LineBytes is to compliment the width given that bytes mod 4 == 0
•     //size of image part, instead of whole bmp
• } IMAGE;
```

3.2. 读入文件

- 在主函数中进行 FILE 等文件操作。在 readme 中直接利用指针 fp 进行文件读取。按原理中所述的顺序读入 BMFH\BMIH\ (PALETTE)\DATA。

```
int ReadBMP(FILE *fp, IMAGE *input){
•
•    //read bmfh
•    fread(&(input->bmfh), sizeof(BMFH), 1, fp);
•    if(input->bmfh.bfType!=0X4D42)
•        return 1;
•
•    //read bmih
•    fread(&(input->bmih), sizeof(BMIH), 1, fp);
•    input->WidthBMP=input->bmih.biWidth;    //get width
•    input->HeightBMP=input->bmih.biHeight;  //get height
•
•    /* usually useless. only be used when BMIH is not complete
•
•    if(input->bmih.biSizeImage==0)
•        //get the size of the image if there's no direct size
•        input->bmih.biSizeImage=input->bmfh.bfSize-
input->bmfh.bfOffBits;
•        input->SizeImageBMP=input->bmih.biSizeImage;
•
•    */
•
input->LineBytes=(input->bmih.biWidth*(input->bmih.biBitCount/8)+3)/4*
4;
•    //compliment to 4 bytes
•
•    if(input->bmih.biBitCount==8)
•        //only when biBitCount=1/2/4/8 does the image has a rgbquad
•        fread(input->rgbquad, sizeof(RGBQUAD), 256, fp); //read rgbquad
•
•    //read data
•    input->data=new BYTE[input->SizeImageBMP];
•    fread(input->data, sizeof(BYTE), input->SizeImageBMP, fp);
•    fclose(fp);
•
•    return 0;
• }
```

3.3. 色彩空间 RGB 转 YUV

- 复制原 24 位深 BMP 图的 BMFH, BMIH 部分, 对部分信息做修正, 随后初始化调色板, 并反复读取 24 位图的信息, 利用转换公式将色彩空间转换为 YUV, 将 Y 值存入 data 部分。
- 在计算 YUV 时, double R,G,B,Y,U,V 以提高计算精度。

```
void color_to_gray(IMAGE *rgb, IMAGE *gray){
•
•     memcpy(&(gray->bmfh), &(rgb->bmfh), sizeof(BMFH));
•     //copy the bmp file header
•     memcpy(&(gray->bmiH), &(rgb->bmiH), sizeof(BMIH));
•     //copy the bmp information header
•     gray->bmfh.bfOffBits=14+40+256*4;
•     //again calculate the offset, given the rgbquad
•     gray->bmiH.biBitCount=8;
•     gray->WidthBMP=rgb->WidthBMP;
•     gray->HeightBMP=rgb->HeightBMP;
•     gray->LineBytes=(gray->WidthBMP+3)/4*4;
•     //bytes mod 4 = 0. complement
•     gray->bmiH.biSizeImage=gray->LineBytes*gray->HeightBMP;
•     //calculate the size of the image data part
•     gray->SizeImageBMP=gray->bmiH.biSizeImage;
•     gray->bmfh.bfSize=gray->bmfh.bfOffBits+gray->bmiH.biSizeImage;
•     //calculate the size of the whole bmp file
•     gray->data=new BYTE[gray->bmiH.biSizeImage];
•
•     memset(gray->data, 0, sizeof(BYTE)*gray->SizeImageBMP);
•     for(int i=0;i<256;i++){ //initialize the rgbquad
•         gray->rgbquad[i].rgbBlue=i;
•         gray->rgbquad[i].rgbGreen=i;
•         gray->rgbquad[i].rgbRed=i;
•     }
•
•     for(int i=0;i<gray->HeightBMP;i++){ //go through each pixel
•         for(int j=0;j<gray->WidthBMP;j++){ //only needs to calculate Y
•             Double Y = rgb->data[rgb->LineBytes * I + j * 3] * 0.299 +
•             rgb->data[rgb->LineBytes * I + j * 3 + 1] * 0.587 +
•             rgb->data[rgb->LineBytes * I + j * 3 + 2] * 0.114;
•             gray->data[gray->LineBytes*i+j]=Y;
•             //double -> int to improve precision
•         }
•     }
•     return;
• }
```

3.4. 改变 Y 值，色彩空间 YUV 转 RGB

- 使用同样的方法，复制原 24 位 BMP 图像的信息。随后使用同样的方法对每个像素点的 RGB 进行色彩空间转换，转为 YUV。随后对 Y 值，即明度（luminance）做修改。将修改后的 YUV 转换回 RGB 色彩空间，将新图像写入。


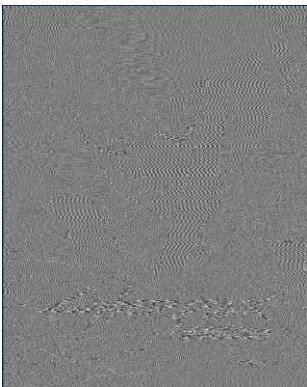

```
• void gray_to_color(IMAGE *rgb, IMAGE *changeY){
•
•     memcpy(&(changeY->bmfh), &(rgb->bmfh), sizeof(BMFH));
•     memcpy(&(changeY->bmih), &(rgb->bmih), sizeof(BMIH));
•     changeY->WidthBMP=rgb->WidthBMP;
•     changeY->HeightBMP=rgb->HeightBMP;
•     changeY->LineBytes=rgb->LineBytes;
•     changeY->SizeImageBMP=changeY->bmih.biSizeImage;
•     changeY->data=new BYTE[changeY->bmih.biSizeImage];
•     //same as color_to_gray; copy information
•
•     for(int i=0;i<changeY->HeightBMP;i++){
•         for(int j=0;j<changeY->WidthBMP;j++){
•             double R, G, B, Y, U, V;
•             B=rgb->data[rgb->LineBytes*i+j*3]; //store in BGR order
•             G=rgb->data[rgb->LineBytes*i+j*3+1];
•             R=rgb->data[rgb->LineBytes*i+j*3+2]; //read in B G R
•             Y=0.299*R+0.587*G+0.114*B;           //calculate Y,U,V
•             U=-0.147*R-0.289*G+0.435*B;
•             V=0.615*R-0.515*G-0.100*B;
•
•             Y=Y*0.4;                               //change luminance
•
•             R=Y+1.4075*V;                           //calculate R,G,B
•             G=Y-0.3455*U-0.7169*V;
•             B=Y+1.779*U;
•
•             changeY->data[changeY->LineBytes*i+j*3]=B; //write data
•             changeY->data[changeY->LineBytes*i+j*3+1]=G;
•             changeY->data[changeY->LineBytes*i+j*3+2]=R;
•         }
•     }
•
•     return;
• }
```

四、实验环境及运行方法







实验环境：Windows10 系统
TDM-GCC 4.9.2 64-bit 编译器
使用 vscode；代码也可直接在 dev-c++打开并编译运行。

五、实验结果展示

正确生成的图像可以直接打卡。部分图像，如图 1 疑惑输出，需通过 vscode 打开。

原图 1	灰度图错误输出	灰度图疑惑输出
		

对于灰度图疑惑输出，同样的程序能正确输出其他图像的灰度图，如下：

原图 2	彩图输出	灰度图输出	$Y=0.75Y$	$Y=0.5Y$	$Y=2Y$
					

$Y=2*Y$ 时补写了 YUV 回转 RGB 时的防溢出代码：

```
if(R>255)R=255;  
if(R<0)R=0;  
if(G>255)G=255;  
if(G<0)G=0;  
if(B>255)B=255;  
if(B<0)B=0;
```

故而对原图 1 出现的部分黑色情况十分困惑不解。程序加了对 Y 溢出的判定和输出，但处理过程中并未出现溢出的提示。且当从网上搜索他人的 RGB 转灰度图程序，复制并对该图片进行测试时，得到的结果仍然大体如上述疑惑输出，但在部分黑色边缘有小型差异。对此我为寻得答案，希望下次课可以和老师交流探讨。

六、心得体会

通过这次实验我理解了 RGB 文件的结构和读写方式。过去在 C 程序设计课的大作业上我曾经直接利用 `graphics.h` 的接口插入 BMP 文件，也用过 `opencv`, `showimg` 等库直接对文件进行输出。但这次实验让我从底层存储开始纯手工实现图片的读入输出，做出后还是很有成就感的。而生成灰度图、改变明度等也让我对色彩空间有了更深的了解。

本次实验印象最深刻的还是有关文件存储的对齐处理。一开始进行读写时数据总是错位。后来经查阅资料才了解到结构体按照最大数据类型进行对齐处理的特性。之后开启 `#pragma pack(1)` 紧凑存储（以 1 字节对齐）才正确读写。又认识到清楚理解文件底层存储方式的重要性。