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

Assignment-1

- Read a color bmp;
- RGB->YUV;
- Color to gray: gray=Y in YUV color space;
- Rearrange gray intensity to lie between [0,255];
- Write a grayscale bmp;
- Change the luminance value Y;
- YUV->RGB;
- Write a color bmp.

二、实验内容和原理

1、bmp 图像存储格式

bfType	Describe the file type. It must be 0x4D42, namely, 'BM'
bfSize	Describe the bitmap file size with Byte.
bfReserved1	Reserved, must be zero.
bfReserved2	Reserved, must be zero.
bfOffBits	Describe the offset from the beginning of the fileheader to the real image data with bytes. This parameter is necessary because the length of "BITMAPINFOHEADER" and "Palette" will change according to different situations. Such offset enables you to access the bitmap data quickly.

文件头

biSize	Number of bytes to define BITMAPINFOHEADER structure
biWidth	Image width (number of pixels)
biHeight	Image height (number of pixels). Note: Besides describing height, "biHeight" can be also denote whether the image is upright or not. (Positive->inverted, Negative->upright). Most of the BMP files are inverted bitmap, namely, biHeight>0.
biPlanes	Number of planes. Always be 1..
biBitCount	Bits per pixel (Bits/pixel), which is 1, 4, 8, 16, 24 or 32.
biCompression	Compression type. Only non-compression is discussed here: BI_RGB.
biSizeImage	Image size with bytes. When biCompression=BI_RGB, biSizeImage=0.
biXPelsPerMeter	Horizontal resolution, pixels/meter.
biYPelsPerMeter	Vertical resolution, pixels/meter
biClrUsed	Number of color indices used in the bitmap (0->all the palette items are used).
biClrImportant	Number of important color indices for image display. 0->all items are important.

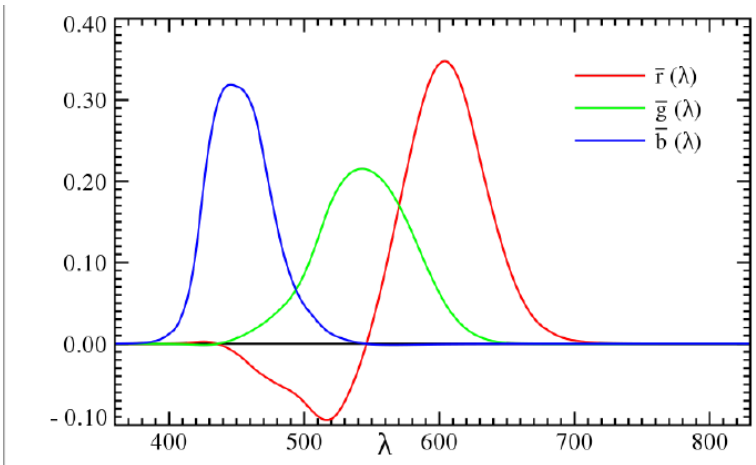
位图信息头

Palette	N*4 bytes	For each item in the Palette, these FOUR bytes are for RGB values: 1 byte for blue 1 byte for green 1 byte for red 1 byte always ZERO.
Bitmap data		Its size depends on image size and color depth. It stores the index number of palette, or RGB value, which depends on the color depth.

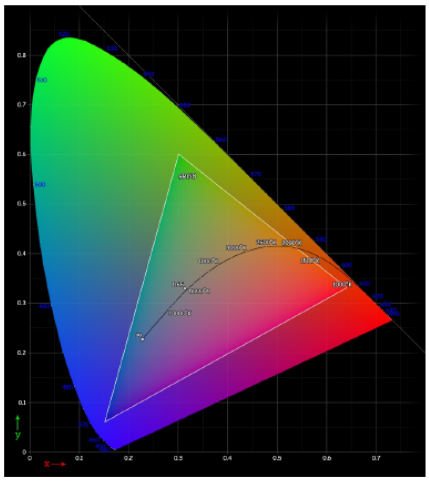
调色盘和图像数据

2、RGB

基于实验结果，红绿蓝三色被选为三原色



1931 CIE RGB system: Standard color vision observer's three-primary color spectrum distribution of stimulation



Color space

3、RGB 与 YUV 转换公式

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.435 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

三、实验步骤与分析

1、结构体创建

```
typedef unsigned short UINT16;
typedef unsigned long DWORD;
typedef unsigned char UNCHAR;
```

根据 Windows 的规则，先创建对应字节数的 UINT16，DWORD，UNCHAR

```
//存储bmp文件头
#pragma pack(push)
#pragma pack(2) //设置2字节缩进
typedef struct tagBITMAPFILEHEADER{
    UINT16 bfType;          //BM
    DWORD  bfSize;          //size of file
    UINT16 bfReserved1;
    UINT16 bfReserved2;
    DWORD bfOffBits;        //header offset
}BITMAPFILEHEADER;          //14字节
#pragma pack(pop)
```

由于结构体的对齐原则，为创建 14 字节的 bmp 文件头结构体，必须使用#pragma

```
//存储位图信息头
typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;           //信息头大小
    DWORD biWidth;          //图像宽度
    DWORD biHeight;         //图像高度
    UINT16 biPlanes;        //图像的面位数
    UINT16 biBitCount;      //每个像素的位数
    DWORD biCompression;    //压缩类型
    DWORD biSizeImage;      //图文件大小
    DWORD biXPelsPerMeter; //水平分辨率
    DWORD biYPelsPerMeter; //垂直分辨率
    DWORD biClrUsed;        //使用的色彩数
    DWORD biClrImportant;   //重要的颜色数
}BITMAPINFOHEADER;          //40字节
```

与 bmp 位图信息头存储内容一致

```
//储存rgb图像信息
typedef struct tagrbgIMAGE{
    UNCHAR **r;//指向二维矩阵，表示图像的R信息
    UNCHAR **g;//指向二维矩阵，表示图像的G信息
    UNCHAR **b;//指向二维矩阵，表示图像的B信息
}rgbIMAGE;
typedef rgbIMAGE *rgbIMAGEINFO;
//存储yuv图像信息
typedef struct tagyuvIMAGE{
    UNCHAR **y;//指向二维矩阵，表示图像的Y信息
    char **u;//指向二维矩阵，表示图像的U信息
    char **v; //指向二维矩阵，表示图像的V信息
}yuvIMAGE;
typedef yuvIMAGE *yuvIMAGEINFO;
```

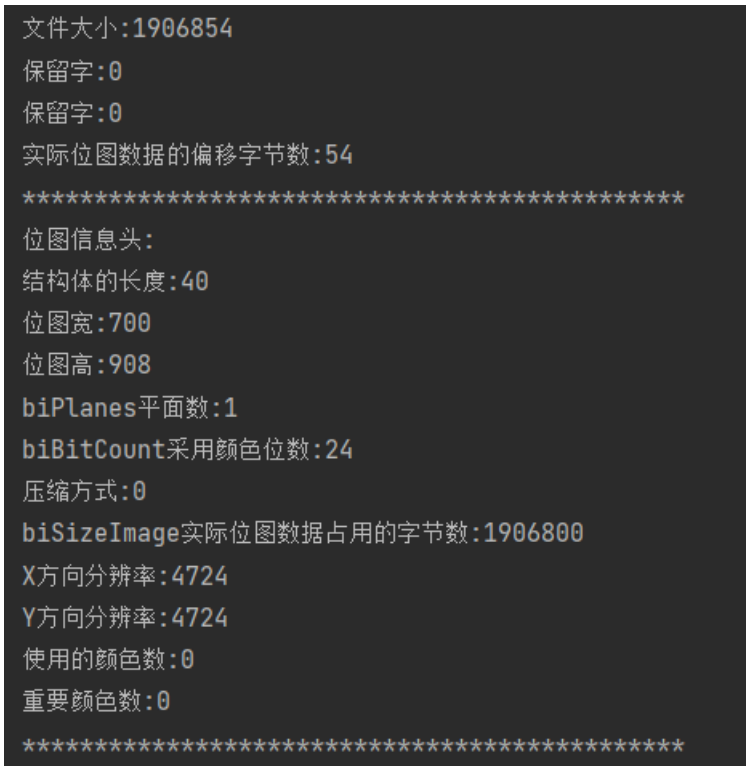
待储存 rgb 和 yuv 的结构体

2、bmp 文件信息的读入及打印

```c
//open bmp file
FILE *fpbmp;
FILE *fpout, *fpout2;
BITMAPFILEHEADER bmpfileheader;    //指向bmp文件头
BITMAPINFOHEADER bmpinfoheader;    //指向bmp信息头
BITMAPFILEHEADER targetfileheader; //指向目标bmp 文件头
BITMAPINFOHEADER targetinfoheader; //指向目标bmp 信息头
rgbIMAGEINFO rgbPicPoint;          //指向存储rgb图信息的结构体
rgbIMAGEINFO outrgb;               //指向yuv转到rgb的图信息的结构体
yuvIMAGEINFO yuvPicPoint;          //指向存储yuv图信息的结构体
char *filename;
int row, col;
int i, j;

//输入文件绝对路径
printf( _Format: "Please enter the filename:");
filename = (char *) malloc( _Size: 100 * sizeof(char));
scanf( _Format: "%s", filename);
//只读打开一个二进制文件
fpbmp = fopen(filename,  _Mode: "rb");
//如果文件读取失败，返回空指针
if(!fpbmp){
    printf( _Format: "Can not open the picture!");
    return 0;
}
//让fpbmp指向bmp文件的开始
fseek(fpbmp,  _Offset: 0, SEEK_SET);
//读取文件信息
fread(&bmpfileheader, sizeof(BITMAPFILEHEADER),  _Count: 1, fpbmp);  //读取文件头信息
fread(&bmpinfoheader, sizeof(BITMAPINFOHEADER), _Count: 1, fpbmp);  //读取信息头信息
showBmpHead(&bmpfileheader);                                  //输出文件头信息
printf( _Format: "*****************************************\n");
showBmpInfoHead(&bmpinfoheader);                              //输出信息头信息
printf( _Format: "*****************************************\n");
```

利用 fread 将对应字节数的信息读入结构体，然后用两个打印函数打印内容



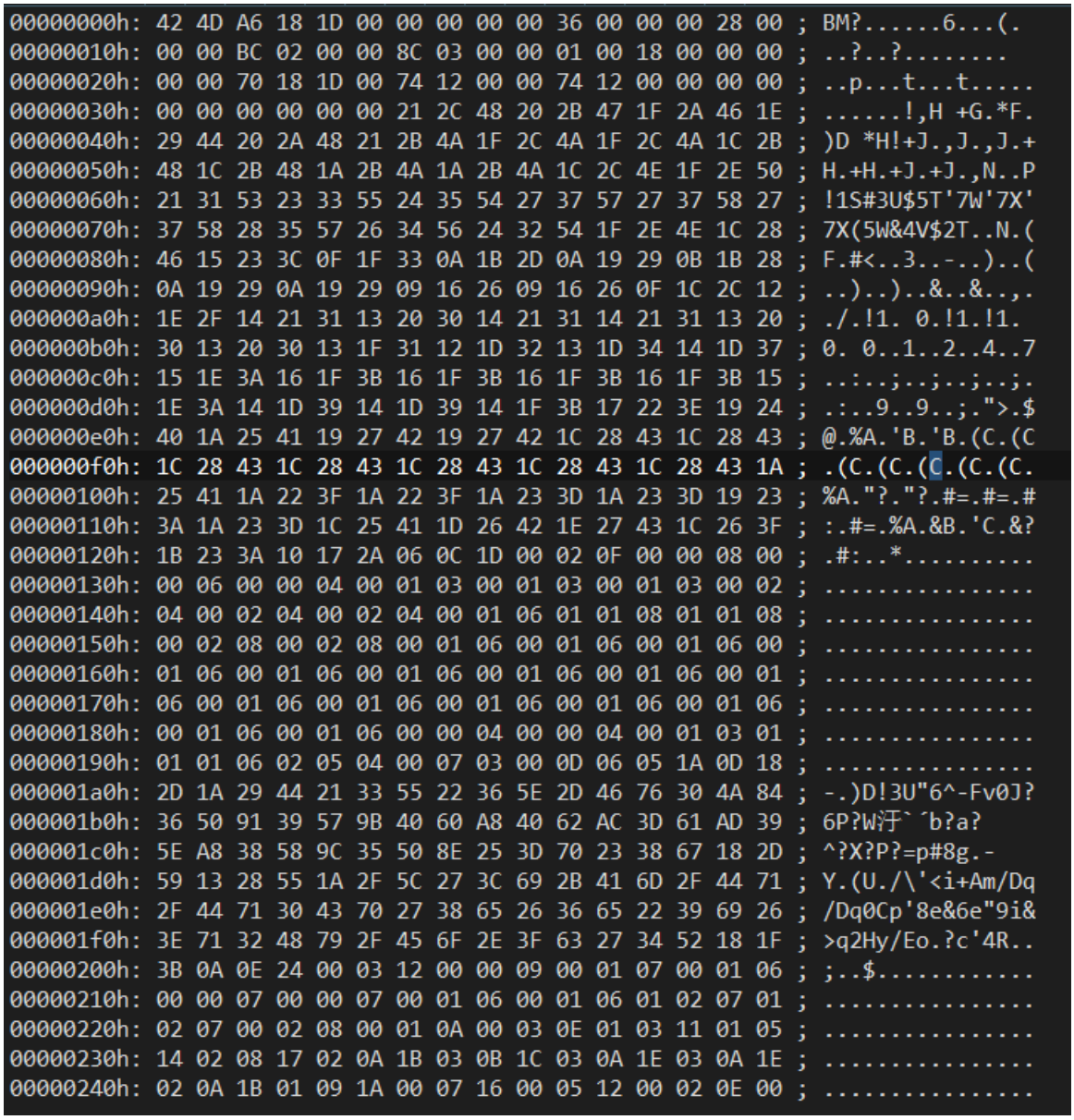
打印结果正确

2、RGB 转 YUV

首先要读取文件的 rgb 信息，默认为 24 位深度以上的 bmp 文件

```c
rgbIMAGEINFO bmp2rgb(BITMAPFILEHEADER* pBmpHead, BITMAPINFOHEADER* pBmpInfoHead, FILE* fp){
    int bfoffbits = pBmpHead->bfOffBits;    //从文件头开始到实际图像数据之间的字节的偏移量
    rgbIMAGEINFO rgb = (rgbIMAGEINFO) malloc(sizeof(rgbIMAGE));
    if(!rgb){
        printf( _Format: "Out of memory!");
        return NULL;
    }
    int row, col;
    fseek(fp, bfoffbits, SEEK_SET);
    int width = pBmpInfoHead->biWidth;     //获得图像的高
    int height = pBmpInfoHead->biHeight;  //获得图像的宽
    //初始化
    rgb->b = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
    for(row = 0; row < height; row++){
        rgb->b[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
    }
    rgb->g = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
    for(row = 0; row < height; row++){
        rgb->g[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
    }
    rgb->r = (UNCHAR **) malloc( _Size: sizeof(UNCHAR*) * height);
    for(row = 0; row < height; row++){
        rgb->r[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
    }
    //读取图像rgb信息
    for(row = 0; row < height; row++){
        for(col = 0; col < width; col++){
            fread(&rgb->b[row][col], sizeof(UNCHAR),  _Count: 1, fp);
            fread(&rgb->g[row][col], sizeof(UNCHAR),  _Count: 1, fp);
            fread(&rgb->r[row][col], sizeof(UNCHAR),  _Count: 1, fp);
        }
    }
    return rgb;
```

注意，输入函数的参数为文件指针 fp，虽之前已 fread 过，但变动的是 fp 的 ptr 指针，fp 本身依然代表文件头，因此要使用 fseek 偏移 offbit 的量。



```
00000000h: 42 4D A6 18 1D 00 00 00 00 00 36 00 00 00 28 00 ; BM?......6...(.
00000010h: 00 00 BC 02 00 00 8C 03 00 00 01 00 18 00 00 00 ; ..?..?........
00000020h: 00 00 70 18 1D 00 74 12 00 00 74 12 00 00 00 00 ; ..p...t...t.....
00000030h: 00 00 00 00 00 00 21 2C 48 20 2B 47 1F 2A 46 1E ; ......!,H +G.*F.
00000040h: 29 44 20 2A 48 21 2B 4A 1F 2C 4A 1F 2C 4A 1C 2B ; )D *H!+J.,J.,J.+
00000050h: 48 1C 2B 48 1A 2B 4A 1A 2B 4A 1C 2C 4E 1F 2E 50 ; H.+H.+J.+J.,N..P
00000060h: 21 31 53 23 33 55 24 35 54 27 37 57 27 37 58 27 ; !1S#3U$5T'7W'7X'
00000070h: 37 58 28 35 57 26 34 56 24 32 54 1F 2E 4E 1C 28 ; 7X(5W&4V$2T..N.(
00000080h: 46 15 23 3C 0F 1F 33 0A 1B 2D 0A 19 29 0B 1B 28 ; F.#<..3..-..)..(
00000090h: 0A 19 29 0A 19 29 09 16 26 09 16 26 0F 1C 2C 12 ; ..)..)..&..&..,.
000000a0h: 1E 2F 14 21 31 13 20 30 14 21 31 14 21 31 13 20 ; ./.!1. 0.!1.!1.
000000b0h: 30 13 20 30 13 1F 31 12 1D 32 13 1D 34 14 1D 37 ; 0. 0..1..2..4..7
000000c0h: 15 1E 3A 16 1F 3B 16 1F 3B 16 1F 3B 16 1F 3B 15 ; ..:..;..;..;..;.
000000d0h: 1E 3A 14 1D 39 14 1D 39 14 1F 3B 17 22 3E 19 24 ; .:..9..9..;.">.$
000000e0h: 40 1A 25 41 19 27 42 19 27 42 1C 28 43 1C 28 43 ; @.%A.'B.'B.(C.(C
000000f0h: 1C 28 43 1C 28 43 1C 28 43 1C 28 43 1C 28 43 1A ; .(C.(C.(C.(C.(C.
00000100h: 25 41 1A 22 3F 1A 22 3F 1A 23 3D 1A 23 3D 19 23 ; %A."?."?.#=.#=.#
00000110h: 3A 1A 23 3D 1C 25 41 1D 26 42 1E 27 43 1C 26 3F ; :.#=.%A.&B.'C.&?
00000120h: 1B 23 3A 10 17 2A 06 0C 1D 00 02 0F 00 00 08 00 ; .#:..*.........
00000130h: 00 06 00 00 04 00 01 03 00 01 03 00 01 03 00 02 ; ................
00000140h: 04 00 02 04 00 02 04 00 01 06 01 01 08 01 01 08 ; ................
00000150h: 00 02 08 00 02 08 00 01 06 00 01 06 00 01 06 00 ; ................
00000160h: 01 06 00 01 06 00 01 06 00 01 06 00 01 06 00 01 ; ................
00000170h: 06 00 01 06 00 01 06 00 01 06 00 01 06 00 01 06 ; ................
00000180h: 00 01 06 00 01 06 00 00 04 00 00 04 00 01 03 01 ; ................
00000190h: 01 01 06 02 05 04 00 07 03 00 0D 06 05 1A 0D 18 ; ................
000001a0h: 2D 1A 29 44 21 33 55 22 36 5E 2D 46 76 30 4A 84 ; -.)D!3U"6^-Fv0J?
000001b0h: 36 50 91 39 57 9B 40 60 A8 40 62 AC 3D 61 AD 39 ; 6P?W汀`ˊb?a?
000001c0h: 5E A8 38 58 9C 35 50 8E 25 3D 70 23 38 67 18 2D ; ^?X?P?=p#8g.-
000001d0h: 59 13 28 55 1A 2F 5C 27 3C 69 2B 41 6D 2F 44 71 ; Y.(U./\'<i+Am/Dq
000001e0h: 2F 44 71 30 43 70 27 38 65 26 36 65 22 39 69 26 ; /Dq0Cp'8e&6e"9i&
000001f0h: 3E 71 32 48 79 2F 45 6F 2E 3F 63 27 34 52 18 1F ; >q2Hy/Eo.?c'4R..
00000200h: 3B 0A 0E 24 00 03 12 00 00 09 00 01 07 00 01 06 ; ;..$............
00000210h: 00 00 07 00 00 07 00 01 06 00 01 06 01 02 07 01 ; ................
00000220h: 02 07 00 02 08 00 01 0A 00 03 0E 01 03 11 01 05 ; ................
00000230h: 14 02 08 17 02 0A 1B 03 0B 1C 03 0A 1E 03 0A 1E ; ................
00000240h: 02 0A 1B 01 09 1A 00 07 16 00 05 12 00 02 0E 00 ; ................
```

利用 ultraedit 观察，很明显数据是 bgr 这样分布的，每个颜色 bgr 分别占一个字节，故直接 fread(&rgb->b[row][col], sizeof(UNCHAR), 1, fp)一个个字节地读取填入二级指针指向的二维数组即可。

```
rgbPicPoint = bmp2rgb(&bmpfileheader, &bmpinfoheader, fpbmp);
```

结果的地址存入指向 rgb 图信息的结构体的指针

```
yuvIMAGEINFO rgb2yuv(rgbIMAGEINFO rgb, int height, int width){
    yuvIMAGEINFO yuv = (yuvIMAGEINFO) malloc(sizeof(yuvIMAGE));
    if(!yuv){
        printf( _Format: "Out of memory!");
        return NULL;
    }
    int row, col;
    //初始化
    yuv->y = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
    for(row = 0; row < height; row++){
        yuv->y[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
    }
    yuv->u = (char**) malloc( _Size: sizeof(char*) * height);
    for(row = 0; row < height; row++){
        yuv->u[row] = (char*) malloc( _Size: sizeof(char) * width);
    }
    yuv->v = (char**) malloc( _Size: sizeof(char*) * height);
    for(row = 0; row < height; row++){
        yuv->v[row] = (char*) malloc( _Size: sizeof(char) * width);
    }
    //用公式将rgb信息转换为yuv
    for(row = 0; row < height; row++){
        for(col = 0; col < width; col++){
            yuv->y[row][col] = (UNCHAR) (0.299 * rgb->r[row][col] + 0.587 * rgb->g[row][col] + 0.144 * rgb->b[row][col]);
            yuv->u[row][col] = (char) (-0.147 * rgb->r[row][col] - 0.289 * rgb->g[row][col] + 0.435 * rgb->b[row][col]);
            yuv->v[row][col] = (char) (0.615 * rgb->r[row][col] - 0.515 * rgb->g[row][col] - 0.100 * rgb->b[row][col]);
        }
    }
    return yuv;
}
```

读取完后使用简单的公式转换为 yuv 并存储

```
//将RGB图片转为YUV
yuvPicPoint = rgb2yuv(rgbPicPoint, height, width);
```

3、Rearrange gray intensity to lie between [0,255]

```
UNCHAR**grayimage = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
for(row = 0; row < height; row++){
    grayimage[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
}
grayimage = rearrange(yuvPicPoint->y, height, width);
```

首先创建二维数组 grayimage

```
UNCHAR **rearrange(UNCHAR **a,int row,int col){
    UNCHAR **rea = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * row);
    int i,j;
    UNCHAR max, min, scope;
    for(i = 0; i < row; i++){
        rea[i] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * col);
    }
    max = findmaxorminnum(a, row, col, choice: 1);
    min = findmaxorminnum(a, row, col, choice: 2);
    scope = max - min;
    for(i = 0; i < row; i++){
        for(j = 0; j < col; j++){
            rea[i][j] = (UNCHAR) (255 * (a[i][j] - min) / scope);
        }
    }
    return rea;
}
```

转灰度图只需 y，因此直接对 y 进行 rearrange，结果存入 grayimage 数组

4、改变 y 的亮度

```
//Change the luminance value Y
for(row = 0; row < height; row++){
    for(col = 0; col < width; col++){
        yuvPicPoint->y[row][col] += 5;
    }
}
```

5、画灰度图

由于 8 位灰度图需要颜色板，为简化操作，采用 24 位图的输出，只需连续三字节同一灰度 y 值即可实现

```
/*Write a grayscale bmp*/
//将信息写入文件
fpout = fopen( _Filename: "gray.bmp",  _Mode: "wb");
if(!fpout){
    printf( _Format: "Can not write the file!");
}
else{
    fwrite(&bmpfileheader, sizeof(BITMAPFILEHEADER),  _Count: 1, fpout);      //将文件头写入文件
    fwrite(&bmpinfoheader, sizeof(BITMAPINFOHEADER),  _Count: 1, fpout);      //将信息头写入文件
    UNCHAR *gray = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * height * width * 3);  //用一个行向量储存图像数据
    //将图像按24位存储，并将原二维数列扩充3倍，放入一维矩阵
    unsigned long k = 0;
    for(row = 0; row < height; row++){
        for(col = 0; col < width; col++){
            gray[k++] = grayimage[row][col];
            gray[k++] = grayimage[row][col];
            gray[k++] = grayimage[row][col];
        }
    }
    //将这个一维数列表示的图像数据写入文件
    fwrite(gray, sizeof(UNCHAR),  _Count: height * width * 3, fpout);
    //关闭写入的文件
    fclose(fpout);
    printf( _Format: "Successfully obtained grayscale image!\n");
}
```

存储的是二维数组，为方便输出，用一维数组 gray 存值

6、yuv 转 rgb

```
//yuv->rgb
outrgb = (rgbIMAGEINFO) malloc(sizeof(rgbIMAGE));
//初始化
outrgb->b = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
for(row = 0; row < height; row++){
    outrgb->b[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
}
outrgb->g = (UNCHAR**) malloc( _Size: sizeof(UNCHAR*) * height);
for(row = 0; row < height; row++){
    outrgb->g[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
}
outrgb->r = (UNCHAR **) malloc( _Size: sizeof(UNCHAR*) * height);
for(row = 0; row < height; row++){
    outrgb->r[row] = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * width);
}
yuvPicPoint->y = rearrange(yuvPicPoint->y, height, width);
//通过公式将yuv转换为rgb
for(row = 0; row < height; row++){
    for(col = 0; col < width; col++){
        outrgb->r[row][col] = (UNCHAR) (1.0000 * yuvPicPoint->y[row][col] + 1.1398 * yuvPicPoint->v[row][col]);
        outrgb->g[row][col] = (UNCHAR) (0.9996 * yuvPicPoint->y[row][col] - 0.3954 * yuvPicPoint->u[row][col] - 0.5805 * yuvPicPoint->v[row][col]);
        outrgb->b[row][col] = (UNCHAR) (1.0020 * yuvPicPoint->y[row][col] + 2.0361 * yuvPicPoint->u[row][col] - 0.0005 * yuvPicPoint->v[row][col]);
    }
}
```

同样的初始化和公式转换

7、画转换后的颜色图

```
//write a color bmp
fpout2 = fopen( _Filename: "lenaps.bmp",  _Mode: "wb");
if(!fpout2){
    printf( _Format: "Wrong!Can not open the file!\n");
}
fwrite(&bmpfileheader, sizeof(BITMAPFILEHEADER),  _Count: 1, fpout2);          //将文件头写入文件
fwrite(&bmpinfoheader, sizeof(BITMAPINFOHEADER),  _Count: 1, fpout2);          //将信息头写入文件
UNCHAR *rgb2 = (UNCHAR*) malloc( _Size: sizeof(UNCHAR) * height * width * 3);   //用一个行向量储存图像数据
//按照bgr的顺序将3个二维矩阵存储为一个一维矩阵
unsigned long k = 0;
for(row = 0; row < height; row++){
    for(col = 0; col < width; col++){
        rgb2[k++] = outrgb->b[row][col];
        rgb2[k++] = outrgb->g[row][col];
        rgb2[k++] = outrgb->r[row][col];
    }
}
//将这个一维数列表示的图像数据写入文件
fwrite(rgb2, sizeof(UNCHAR),  _Count: height*width*3, fpout2);
//关闭文件
fclose(fpout2);
printf( _Format: "Successfully obtained RGB image!\n");
```

与画灰度图算法基本一致

8、停止

```
system( _Command: "pause");
```

四、实验环境及运行方法

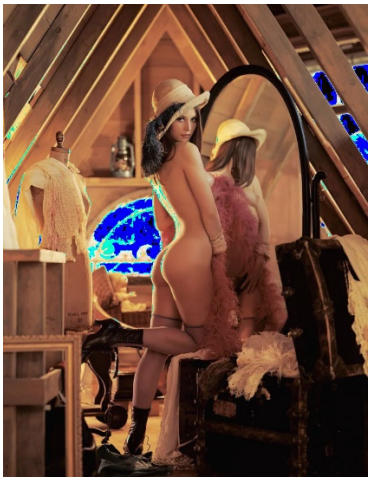
编译环境：C 语言，ide 使用 clion，利用 cmakelist 编译。用 dev 可直接打开 main.c 源文件进行编译

测试方法：断点单步测试（无测出问题）、使用 UltraEdit 查看画出的 bmp 的二进制数据来找输出算法的问题

五、实验结果展示

（展示实验中的输入输出图像等）





原图　　灰度图　　YUV 转 RGB 后

生成的图片与编译文件在同一路径下

六、心得体会

本次 bmp 处理作为第一次作业来说难度较大，但我也成功地做了出来。从中，我巩固加强了结构体、二进制文件操作以及高级指针的应用，学习了#pragma 指令和@param 的注释方法，了解了查看二进制文件的技巧，收获颇丰。