二、*实验内容和原理*

1、BMP格式

BMP位图文件可看成由4个部分组成：位图文件头（bitmap-file header）、位图信息头（bitmap-information header）、彩色表（color table）和定义位图的字节(位图数据，即图像数据，Data Bits 或Data Body)阵列。

位图文件头的格式可以用以下结构描述。

```
typedef struct FILEHEADER{
    // unsigned short bftype;      // Must be 0x4d42.
    unsigned int   bfSize;         // Size of Figure.
    unsigned short bfReserved1;    // Reserved1, must be 00.
    unsigned short bfReserved2;    // Reserved2, must be 00.
    unsigned int   bfOffBits;      // Offsets from header to info.
}BMPFILEHEADER;
```

位图信息头可以用以下结构描述。

```
typedef struct INFOHEADER{
    unsigned int   biSize;         // Size of this structure
    unsigned int   biWidth;        // Width
    unsigned int   biHeight;       // Height
    unsigned short biPlanes;       // always is 1
    unsigned short biBitCount;     // Kind of BMP, bit/pixel = 1 4 8 16 24 32
    unsigned int   biCompression;
    unsigned int   biSizeImage;
    unsigned int   biXPelsPerMeter;
    unsigned int   biYPelsPerMeter;
    unsigned int   biClrUsed;
    unsigned int   biClrImportant;
 }BMPINFOHEADER;
```

调色板可以用以下结构描述。

```
typedef struct tagRGBQUAD{
    unsigned char rgbBlue;     //该颜色的蓝色分量
    unsigned char rgbGreen;    //该颜色的绿色分量
    unsigned char rgbRed;      //该颜色的红色分量
    unsigned char rgbReserved; //保留值
}RGBQUAD;
```

位图的字节可以用以下结构描述。

```
typedef struct RGBSPACE{
    unsigned char blue;
    unsigned char green;
    unsigned char red;
}RGB;
```

2、RGB格式

red	**必要参数：1 byte。数值范围从 0 到 255，表示颜色的红色成份。**
green	必要参数：1 byte。数值范围从 0 到 255，表示颜色的绿色成份。
blue	必要参数：1 byte。数值范围从 0 到 255，表示颜色的蓝色成份。

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。

3、YUV格式

YUV是编译true-color颜色空间（color space）的种类，“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。

4、RGB -> YUV

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.435 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

5、RGB -> YUV

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & -0.0000 & 1.1398 \\ 0.9996 & -0.3954 & -0.5805 \\ 1.0020 & 2.0361 & -0.0005 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{2}$$

但是实际实验中，我并没有用这两条公式，原因是颜色输出的差异较大，我更换了另一个更精确的公式，在步骤中会有体现。

三、实验步骤与分析

1、定义结构

分别定义图像文件头、图像信息头、调色板数据结构、RGB数据结构和YUV数据结构。前四者的定义在原理中有体现，YUV空间数据结构定义如下。

```
typedef struct YUVSPACE{
    unsigned char Y;
    unsigned char U;
    unsigned char V;
}YUV;
```

在实际的实验过程中我发现一个问题：如果单纯定义完整的图像文件头的数据，会出现bug，读取错误。但是分开定义文件类型 bftype 和其他的文件头内容之后，bug就消失了。所以在实验中我采用了如下定义。

```
    unsigned short bftype;
    // head includes head information of BMP file.
    BMPFILEHEADER head;
    // info includes figure information of BMP file.
    BMPINFOHEADER info;
```

2、读取文件判断是否符合规定

判断包括此图像是否为BMP格式文件，并且bitcount是否为24位。

```
    // infp is original BMP file.
    // rgb2yuvfp is RGB -> YUV BMP output file.
    // yuv2rgbfp is YUV -> RGB BMP output file.
    FILE *infp,*rgb2yuvfp,*yuv2rgbfp;

    // read F1.bmp
    if((infp=fopen("F1.bmp","rb")) == NULL){
        printf("OPEN FAILED!\n");
        return 0;
    }

    // bftype must be 0x4d42.
    // There is a question: I can't read the file correctly if read bftype in struct FILEHEADER.
    fread(&bftype,1,sizeof(unsigned short),infp);
    if(bftype != 0x4d42){
        printf("This figure is not BMP!\n");
        Sleep(1000);
        return 0;
    }

```

```
    // read HEADER
    fread(&head,1,sizeof(BMPFILEHEADER),infp);
    fread(&info,1,sizeof(BMPINFOHEADER),infp);

    // We takes 24 bitcount bmp for experiment item.
    if(info.biBitCount != 24){
        printf("This BMP is not 24 biBitCount!\n");
        Sleep(1000);
        return 0;
    }

    // A pixel contains 3 bytes data, R, G, B.
    int size = info.biSizeImage / 3;

    // define 3 array
    RGB inrgb[size];
    YUV rgb2yuv[size];
    RGB yuv2rgb[size];

    // read information.
    fread(inrgb,size,sizeof(RGB),infp);
    fclose(infp);
```

3、RGB -> YUV

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \left(\begin{bmatrix} 66 & 129 & 25 \\ -38 & -74 & 112 \\ 112 & -94 & -18 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + 128 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) \div 256 + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \tag{3}$$

```
    int i;
    // change RGB into YUV.
    for(i=0;i<=size-1;i++){
      rgb2yuv[i].Y = ((  66 * inrgb[i].red + 129 * inrgb[i].green +
 25 * inrgb[i].blue + 128) >> 8) +  16;

      rgb2yuv[i].U = (( -38 * inrgb[i].red -  74 * inrgb[i].green +
112 * inrgb[i].blue + 128) >> 8) + 128;

      rgb2yuv[i].V = (( 112 * inrgb[i].red -  94 * inrgb[i].green -
 18 * inrgb[i].blue + 128) >> 8) + 128;

      rgb2yuv[i].Y = cut(rgb2yuv[i].Y); // Rearrage Y to [0,255].
    }
    // Output RGB to YUV Figure.
    if((rgb2yuvfp = fopen("F1_RGB2YUV.bmp","wb")) == NULL){
```

```
        printf("Create Outfile1 FAILED!\n");
        Sleep(1000);
        return 0;
    }

    // Write header information and data into file.
    fwrite(&bftype,1,sizeof(unsigned short),rgb2yuvfp);
    fwrite(&head,1,sizeof(BMPFILEHEADER),rgb2yuvfp);
    fwrite(&info,1,sizeof(BMPINFOHEADER),rgb2yuvfp);

    for(i=0;i<=size-1;i++){
        fwrite(&rgb2yuv[i].Y,1,sizeof(unsigned char),rgb2yuvfp);
        fwrite(&rgb2yuv[i].Y,1,sizeof(unsigned char),rgb2yuvfp);
        fwrite(&rgb2yuv[i].Y,1,sizeof(unsigned char),rgb2yuvfp);
    }

    fclose(rgb2yuvfp);
    printf("Output YUV Image successfully!\n");

```

4、YUV -> RGB

```
    // Change YUV into RGB.

    for(i=0;i<=size-1;i++){
        int C = rgb2yuv[i].Y - 16;
        int D = rgb2yuv[i].U - 128;
        int E = rgb2yuv[i].V - 128;
        yuv2rgb[i].red   = cut(( 298 * C            + 409 * E + 128) >> 8);
        yuv2rgb[i].green = cut(( 298 * C  - 100 * D - 208 * E + 128) >> 8);
        yuv2rgb[i].blue  = cut(( 298 * C  + 516 * D           + 128) >> 8);

    }
    // Write YUV to RGB file.
    if((yuv2rgbfp = fopen("F1_YUV2RGB.bmp","wb")) == NULL){
        printf("Create Outfile2 FAILED!\n");
        Sleep(1000);
        return 0;
    }
    // Write header information and data into file.
    fwrite(&bftype,1,sizeof(unsigned short),yuv2rgbfp);
    fwrite(&head,1,sizeof(BMPFILEHEADER),yuv2rgbfp);
```

```
    fwrite(&info,1,sizeof(BMPINFOHEADER),yuv2rgbfp);

    for(i=0;i<=size-1;i++){
        fwrite(&yuv2rgb[i].blue ,1,sizeof(unsigned char),yuv2rgbfp);
        fwrite(&yuv2rgb[i].green,1,sizeof(unsigned char),yuv2rgbfp);
        fwrite(&yuv2rgb[i].red  ,1,sizeof(unsigned char),yuv2rgbfp);
    }

    fclose(yuv2rgbfp);
    printf("YUV to RGB successfully!\n");

```

5、调整亮度

```
    for(i=0;i<=size-1;i++){
        newyuv[i].Y = 0.8 * rgb2yuv[i].Y;
        newyuv[i].U =       rgb2yuv[i].U;
        newyuv[i].V =       rgb2yuv[i].V;
    }
```

6、Rearragement

```
  int ymax = 0, ymin = 300;

    for(i=0;i<=size-1;i++){
        if(newyuv[i].Y > ymax) ymax = newyuv[i].Y;
        if(newyuv[i].Y < ymin) ymin = newyuv[i].Y;
    }

    float k = 255/(ymax-ymin);
    float b = -k*ymin;
```

对于超过255或者小于0的情况，对其进行线性映射。

四、实验环境及运行方法

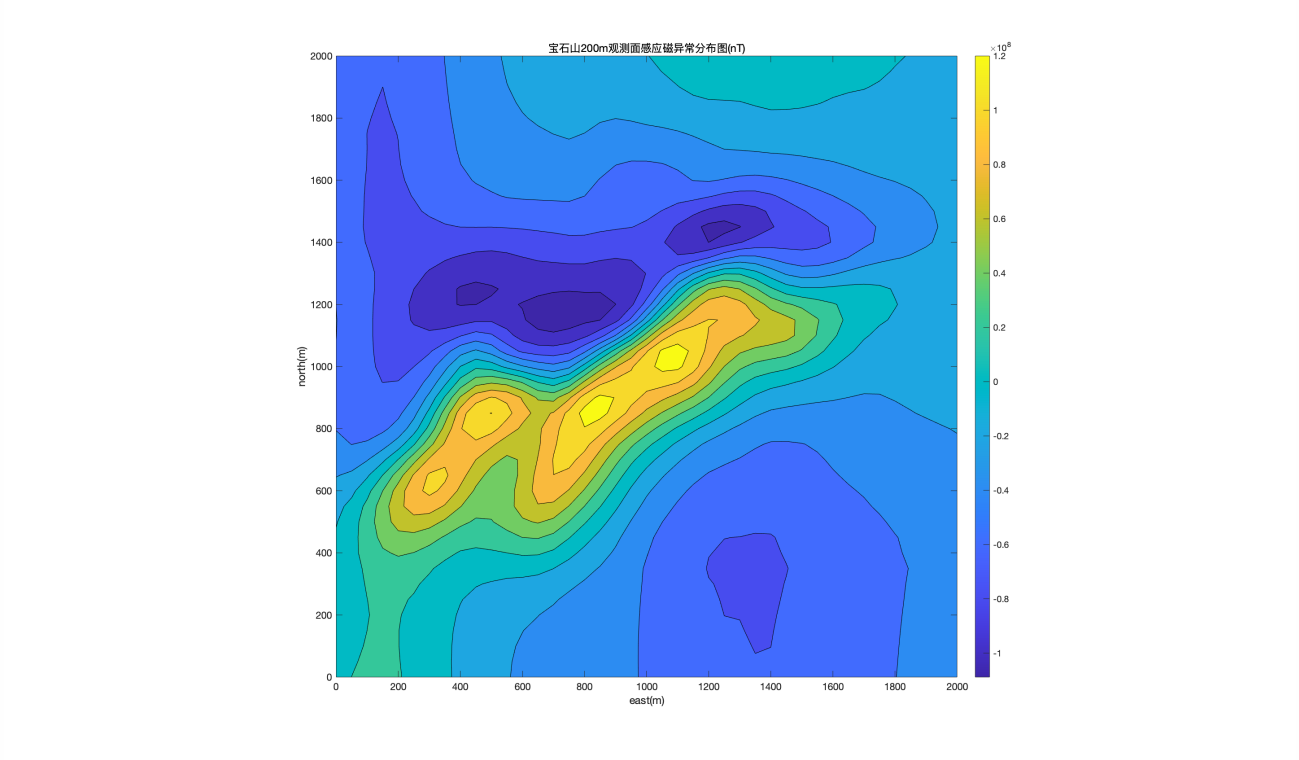
实验运行在MACOS系统下VMware软件的Windows10虚拟机下的Dev C++软件。

只需打开源代码文件，点击编译即可。示例用图放在文件夹中，可以自行更换。

五、实验结果展示

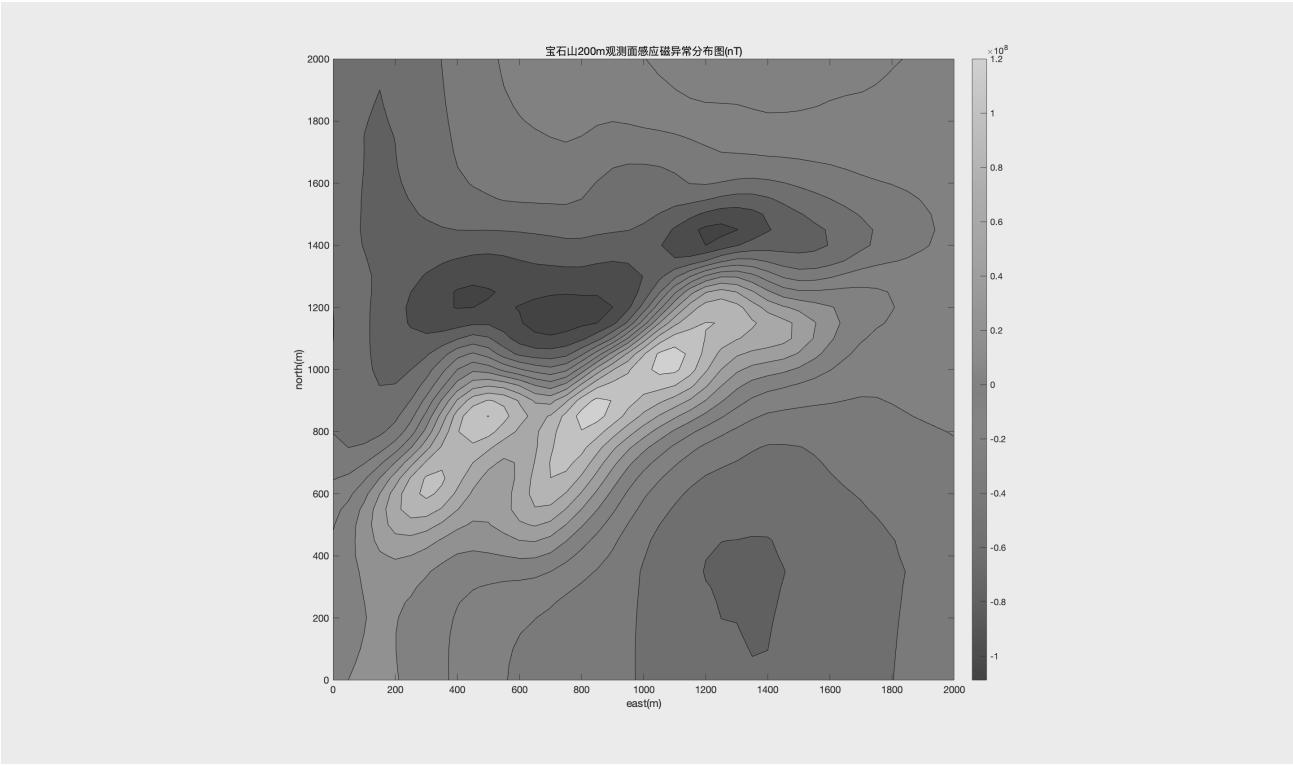
1、原始图像

原始图像我采用了我之前在场论课上用MATLAB制作的西湖宝石山感应磁异常分布图。这张图片是BMP文件，并且有较高的清晰度，同时有很多大的色彩块，色块之间区分度大，边界明显，能很好地反应颜色效果。

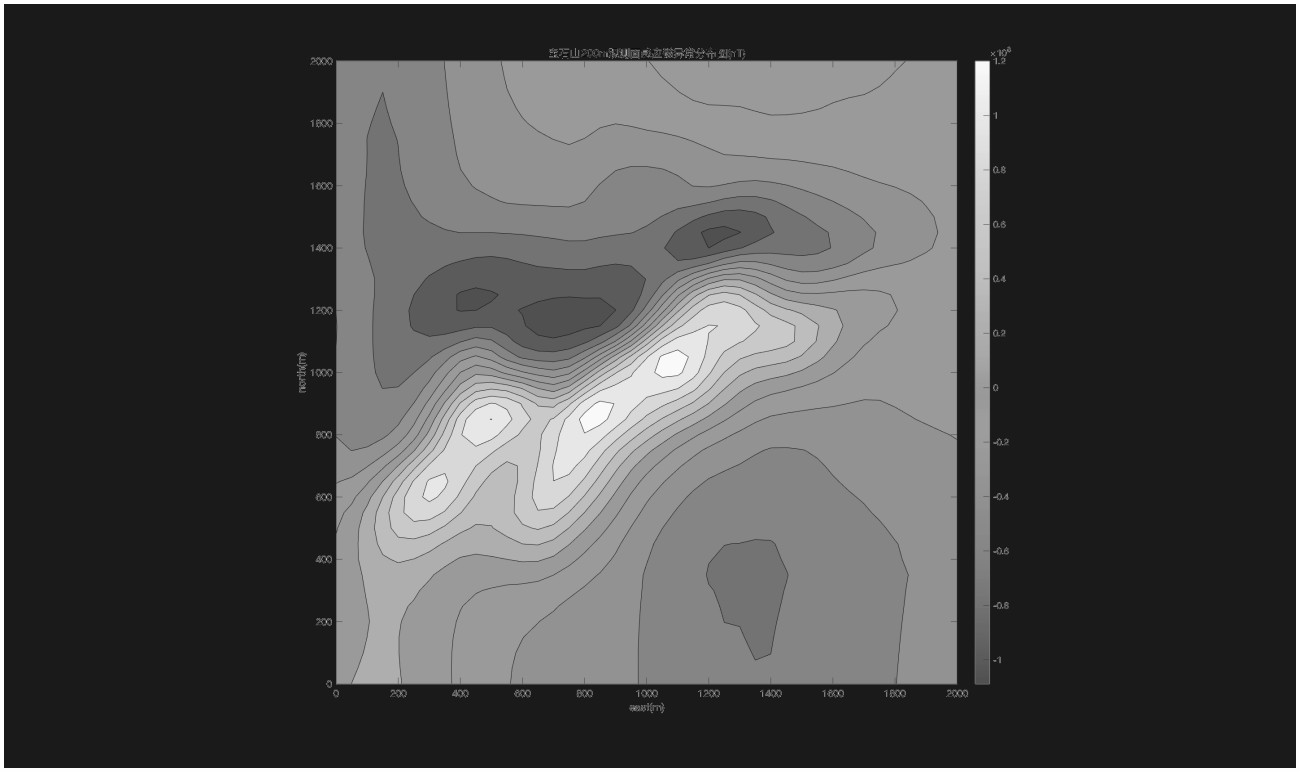


2、RGB -> YUV 灰度图像

（1）原始灰度图像

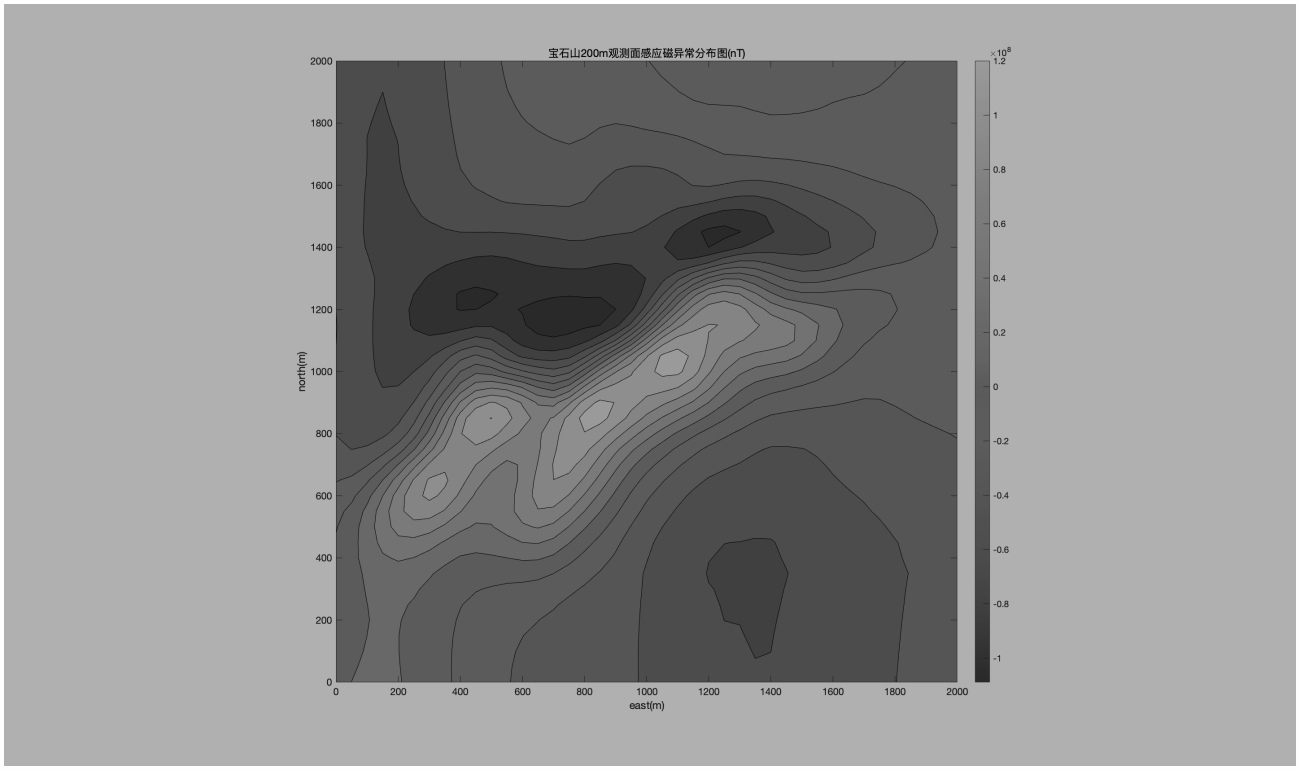


（2）将Y设置成原来的1.2倍



可以看到画面亮度有所提升，但是不知道为什么，背景自动调成了黑色。试了很久一直没有解决问题。

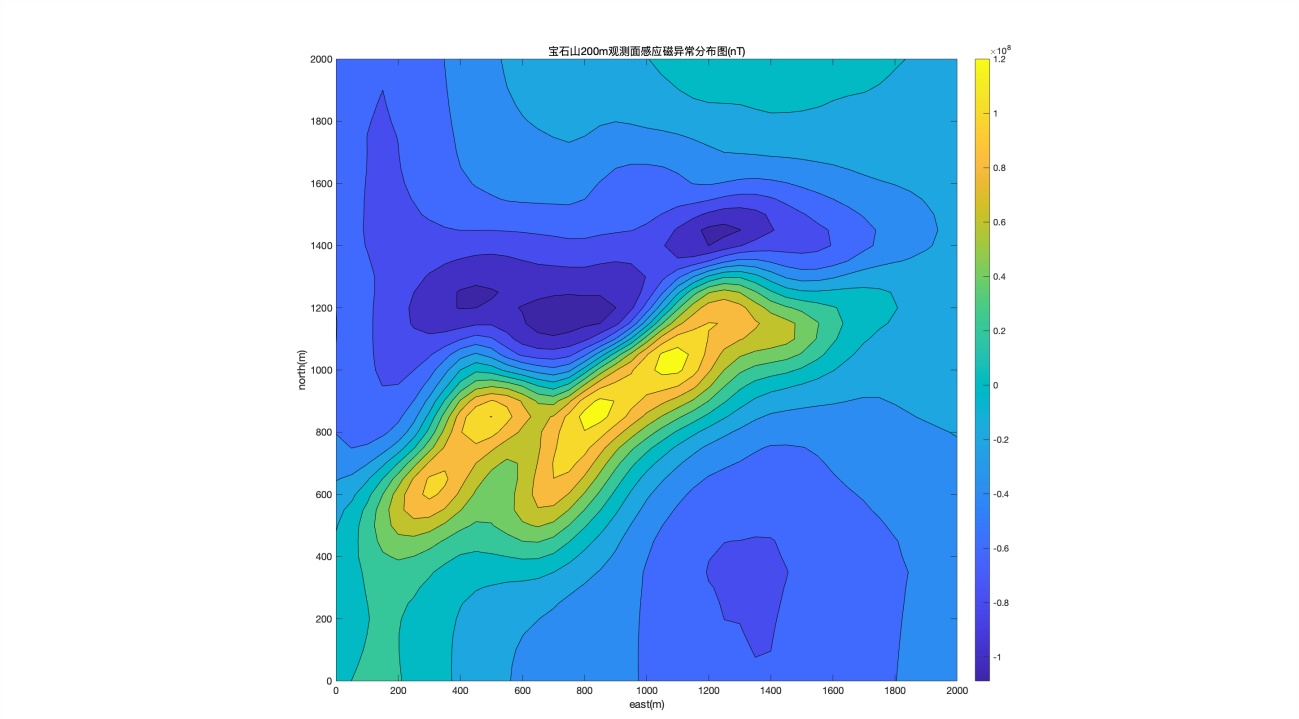
（3）将Y设置为原来的0.8倍



可以看到画面明显变暗了。

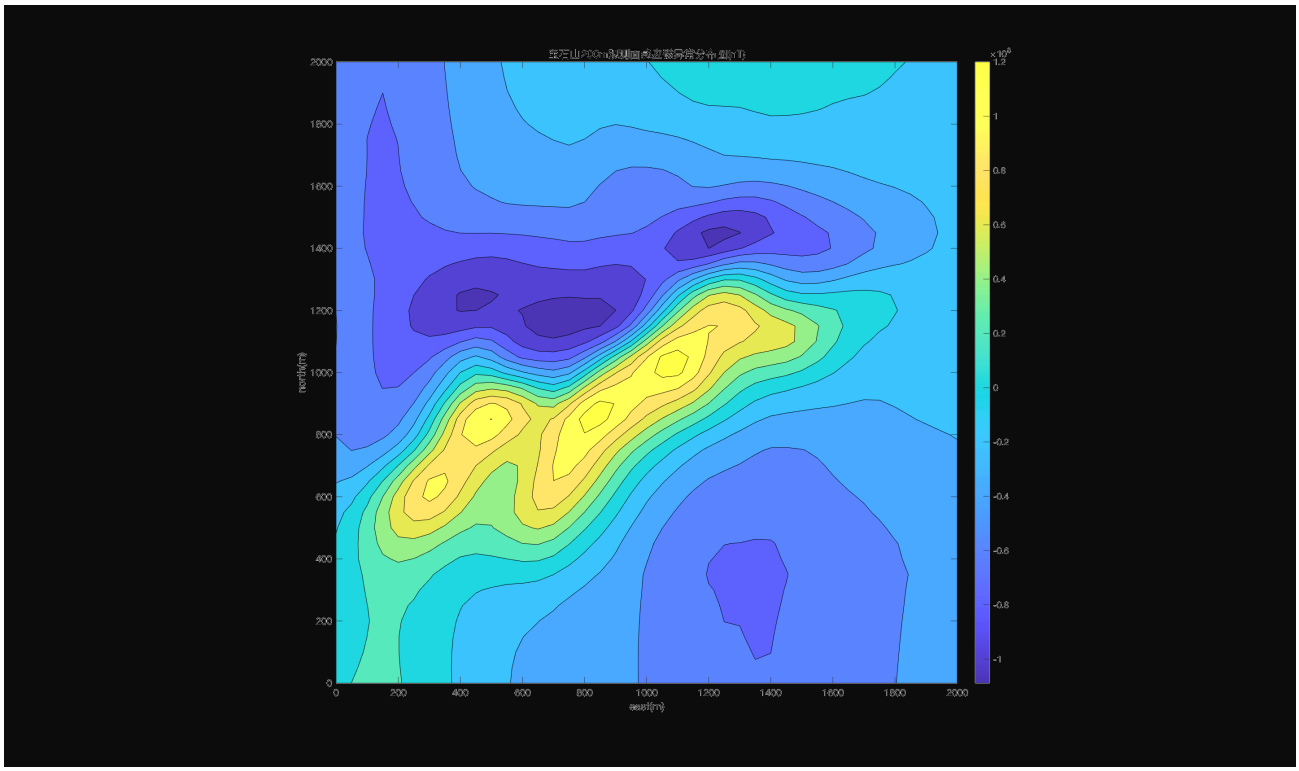
3、YUV -> RGB

（1）原始YUV转换



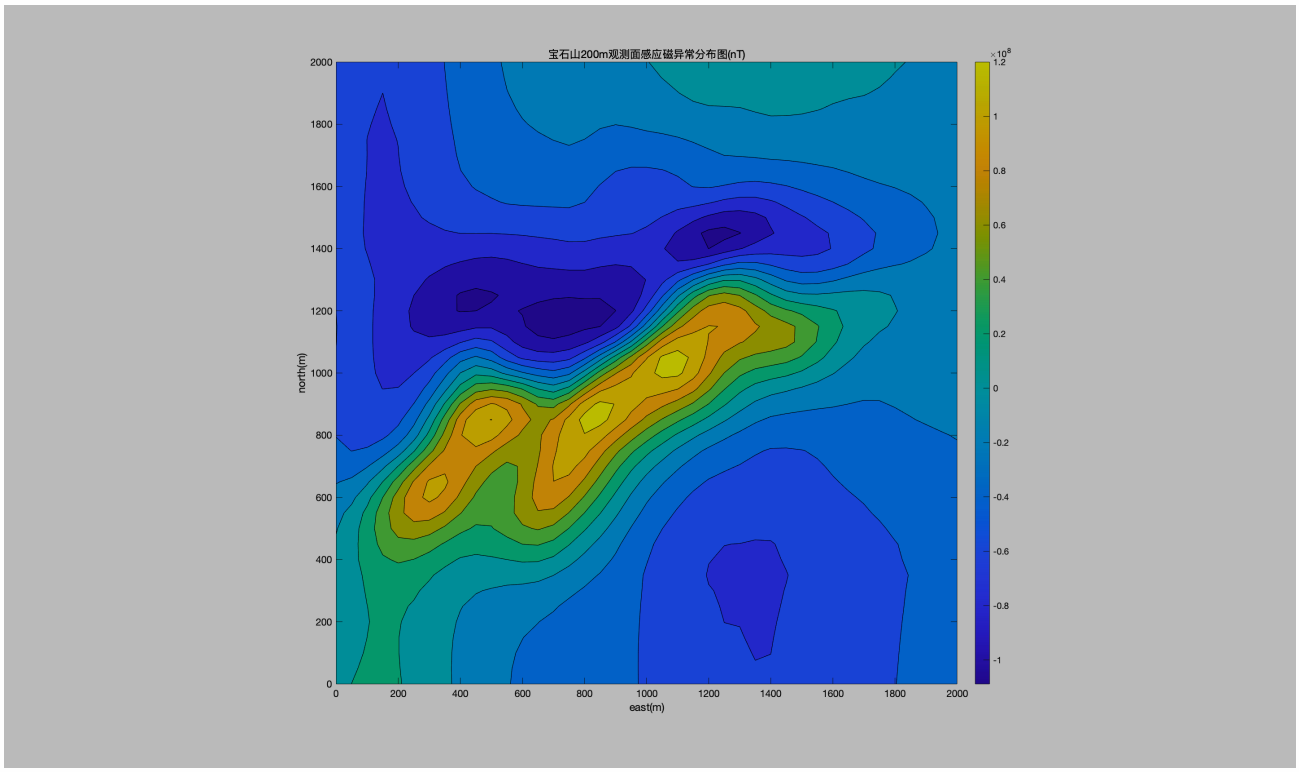
可以看到，在通过一次YUV空间到RGB空间后，图片的质量仍然保留，没有改变。

（2）将Y设置成原来的1.2倍



画面明显变亮，但是也遇到了背景变黑的问题。

（3）将Y设置成原来的0.8倍



画面在经过YUV变换后，减小Y的值后，整体亮度下降。

六、心得体会

1、讨论

（1）如果需要读入其他bitcount类型的BMP文件，只需要更改调色板数据和读入数据类型即可。

（2）根据老师上课给的RGB <–> YUV 公式制作出来的效果图并不是很好，因此我又换了一种公式进行处理。

（3）不知到为什么，在放大Y时，尽管做了线性映射但还是会出现画面变黑的情况。

2、注意点

（1）要分开读取文件类型bftype和文件信息头，否则会出现bug。原理未知。

（2）如果不清楚数据到底储存在哪个位置，可以利用文件中的 bfOffBits 即偏移量来判断起始位置，调用 fseek(fp,0,bfOffBits) 命令移动指针。

（3）打开一个文件都要判断一下是否为 NULL ，以确保文件创建成功。

（4）读写文件要用二进制写法比如 fopen("xx.bmp","wb")。

（5）YUV空间中的Y分量由RGB转化而来可能会超过255，因此我在程序中加了一个截断函数 cut 来将超出的部分截断，实验下来的效果还可以，原始图像和YUV2RGB图像没有很大的区别。

（6）由于实验中用到的量几乎都是正值，因此要采用无符号数 unsigned 来定义变量。并且要注意分量是 int 型还是 short 型，或是 char 型。一个变量读取错误，会引发一系列变量读取错误。可以提供一个经验，24位的BMP图，信息头的大小是54个字节，可以根据此来计算，是否完整读取了信息头。

3、心得体会

本次实验我们基于C语言完整地读取和写出了一张BMP图，是尝试的一种新格式，还是很有意义的。同时还深入理解了RGB色彩空间和YUV色彩空间的组成。从转换灰度图、恢复RGB彩图的过程中，真正地在底层实现了一张图片的转换，这和美图秀秀操作来比，有意义了很多，也很有成就感。这次实验也复习了文件读取等知识，收获很多。