浙江大学实验报告

专业：计算机科学与技术
姓名：NoughtQ
学号：1145141919810
日期：2024 年 10 月 9 日

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

- 实现对位图(BMP)文件的读取和写入
- 实现 RGB 与 YUV 色彩空间的转化
- 在彩色图的基础上生成一张灰度图
- 通过改变 Y 值来调整原图的亮度

二、 实验内容和原理

2.1 BMP 文件格式

BMP（bitmap，位图）图像是 Windows 系统中的一种标准文件格式，它以无压缩的方式组织图片信息。以下是 BMP 的文件结构：

Image File Header
Image Information Header
Palette
Image Data

Table 1: BMP 文件结构

1. **Image File Header（图像文件头）**：
   - `bfType`（2 Byte）：值固定为 `0x4D42`，即 `BM`，表明这是一个 BMP 文件
   - `bfSize`（4 Byte）：文件大小
   - `bfReversed1`（2 Byte）：保留值，固定为 0
   - `bfReserved2`（2 Byte）：保留值，固定为 0
   - `bfOffBits`（4 Byte）：从文件开头到实际图像数据之间的偏移量(offset)。由于后面的 `BITMAPINFOHEADER` 和 `Palette` 视情况而改变，所以依靠这个偏移量能够提高访问位图的速度
2. **Image Infomation Header（图像信息头）**：
   - `biSize`：`BITMAPINFOHEADER` 结构体的大小
   - `biWidth`：图像的宽度（单位：像素）

- `biHeight`：图像的高度（单位：像素）
  - 值为正数的时候，图像是倒立的；值是负数的时候，图像才是正向的。这是因为读取位图文件时是从下往上读取像素的
  - 大多数的位图都是倒立的，即这个值是正数
  - 不过 API 函数会在显示位图前将倒立图像自动转过来，因此不需要我们手动调整
- `biPlanes`：位面(plane)数，值固定为 1
- `biBitCount`：每像素的比特数（bit / pixel），其值为 1、4、8、16、24 或 32，但大多数图像是 24 位或 32 位
- `biCompression`：压缩类型，这里只讨论未压缩类型，其值为 `BI_RGB`
- `biSizeImage`：图像的大小（单位：字节），若 `BiCompression = BI_RGB`，其值为 0
- `biXPelsPerMeter`：图像的水平分辨率(horizontal resolution)（单位：像素/米）
- `biYPelsPerMeter`：图像的垂直分辨率(vertical resolution)（单位：像素/米）
- `biClrUsed`：位图用到调色盘上的颜色索引数，如果所有颜色都用到了，其值为 0
- `biClrImportant`：位图用到调色盘上的有重要影响的颜色索引数，如果所有颜色都重要，其值为 0

3. **Palette（调色盘）**：
   - 大小：N*4 Bytes
   - 调色盘的每一项的大小为 4 字节，每个字节分别对应蓝色（`rgbBlue`）、绿色（`rgbGreen`）、红色（`rgbRed`）和固定的 0 值（`rgbReserved = 0`）
   - 更确切的叫法应该叫做**颜色查找表**(LUT, lookup table)，因为我们会将用到的颜色存放在调色盘内，如果某个像素需要使用其中一种颜色，那么这个像素点会用这个颜色在调色盘的索引值来表示该颜色，而不是 RGB 值，从而节省了存储空间。
4. **Image Data（图像数据）**：
   - 大小取决于图像大小和颜色深度
   - 注意：每一行的字节数必须是 4 的倍数，若不是则需要补齐（最简单的做法是在末尾补 0）

2.2 RGB 与 YUV 之间的转换

2.2.1 RGB 色彩空间

RBG 颜色模型是三维直角坐标颜色系统中的一个单位正方体。

- 在正方体的主对角线上，各原色的量相等，产生由暗到亮的白色，即灰度
- （0，0，0）为黑，（1，1，1）为白，正方体的其他 6 个角点分别为红、黄、绿、青、蓝和品红
- RGB 颜色模型构成的颜色空间是 CIE 原色空间的一个真子集
- RGB 颜色模型通常用于彩色阴极射线管和彩色光栅图形显示器（计算机和电视机采用）

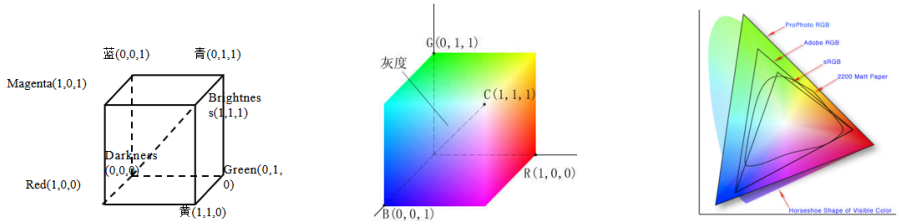

Figure 1: RGB 颜色模型

2.2.2 YUV 色彩空间

在现代彩色电视系统中，通常采用三管彩色摄像机或彩色 CCD（电荷耦合器件）摄像机，它把摄得的彩色图像信号，经分色，分别放大校正得到 RGB，再经过矩阵变换电路得到亮度信号 Y 和两个色差信号 R－Y、B－Y，最后发送端将亮度和色差三个信号分别进行编码，用同一信道发送出去。

采用 YUV 颜色空间的重要性是它的亮度信号 Y 和色度信号 U、V 是分离的。如果只有 Y 信号分量而没有 U、V 分量，那么这样表示的图就是黑白灰度图。彩色电视采用 YUV 空间正是为了用亮度信号 Y 解决彩色电视机与黑白电视机的兼容问题，使黑白电视机也能接收彩色信号。

2.2.3 转换

可以用线性代数中矩阵乘法的知识来实现 RGB 与 YUV 之间的转换，具体公式如下所示：

- RGB $\rightarrow$ YUV:

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

- YUV $\rightarrow$ RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.4075 \\ 1 & -0.3455 & -0.7169 \\ 1 & 1.779 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

三、 实验步骤与分析

3.1 BMP 文件结构体

```
typedef unsigned char BYTE;    // 字节
typedef unsigned short WORD;   // 字长 = 2 字节
typedef unsigned int DWORD;    // 双字长 = 4 字节
typedef unsigned int LONG;
#pragma pack(1)
```

```
// 图像文件头
typedef struct tagBITMAPFILEHEADER {
    BYTE  bfType[2];
    DWORD bfSize;
    WORD  bfReversed1;
    WORD  bfReserved2;
    DWORD bfOffBits;
} BITMAPFILEHEADER, * PBITMAPFILEHEADER;

// 图像信息头
typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER, * PBITMAPINFOHEADER;

// 调色盘
typedef struct tagRGBQUAD{
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;

// BMP 文件 = 前面的结构体 + 图像数据
typedef struct tagBMPFILESTRUCT {
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    RGBQUAD aColors[COLORNUM];
    BYTE *aBitmapBits;
} * BMPFILE;
```

· 根据 2.1 节描述的 BMP 文件格式，我们可以构造出对应的结构体。整个 BMP 文件的结构体共有 4 个字段，前 2 个字段是**文件头**和**信息头**，它们也是结构体；第 3 个

字段是**调色盘**，是一个结构体数组；第 4 个字段存储图像数据，根据 `biSizeImage` 来分配存储空间。

- 由于默认的结构体对齐方式会使字段之间产生空位，而实际的 BMP 文件并没有空位，因此需要设置 `#pragma pack(1)` 使字段连续排列，从而避免读取错误。

3.2 读取文件

```
// 读取 BMP 文件
BMPFILE ReadBMPFile() {
    FILE * fp;
    BMPFILE image;
    LONG w, h;
    DWORD imageSize;

    // 打开文件
    fp = fopen(BMPFILEPATH, "rb+");
    if (!fp) {
        printf("Fail to open the file!\n");
        exit(1);
    }
    printf("Successfully open the file!\n");

    image = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 读取文件头 & 错误处理
    fread(&(image->bmfh), sizeof(BITMAPFILEHEADER), 1, fp);
     if (image->bmfh.bfType[0] != 'B' || image->bmfh.bfType[1] != 'M') {
        printf("The file is not a BMP file!\n");
        exit(1);
    }
    if (image->bmfh.bfReversed1 || image->bmfh.bfReserved2) {
        printf("Wrong reversed value in BMP file!\n");
        exit(1);
    }
    printf("Size: %d(bit)\n", image->bmfh.bfSize);

    // 读取信息头 & 错误处理 & 打印图像信息
    fread(&(image->bmih), sizeof(BITMAPINFOHEADER), 1, fp);
    if (image->bmih.biCompression) {
        printf("The program don't take the compreesed BMP file into account!\n");
        exit(1);
```

```
    }
    if (image->bmih.biBitCount != RGBNUM * 8) {
        printf("Wrong bit numbers per pixel!\n");
        exit(1);
    }
    w = image->bmih.biWidth;
    h = image->bmih.biHeight;
    // 计算宽度上的位数，注意位图数据每一行的字节数必须是 4 的倍数，若不是则需要补齐
    ColorBitWidth = (w * RGBNUM + RGBNUM) / 4 * 4;
    printf("ColorBitWidth: %d\n", ColorBitWidth);
    imageSize = image->bmih.biSizeImage;
    if (!imageSize) {    // 这个值虽然表示图像大小，但通常为 0，因此需要手动计算
            image->bmih.biSizeImage = image->bmfh.bfSize - image->bmfh.bfOffBits;
        imageSize = image->bmih.biSizeImage;
    }
    printf("Width: %d\n", w);
    printf("Height: %d\n", h);
    printf("Image Size: %d\n", imageSize);

    // 由于彩色图没有调色盘，因此不需要读取
    // fread(&(image->aColors), COLORNUM * sizeof(RGBQUAD), 1, fp);

    // 读取图像数据
    fseek(fp, image->bmfh.bfOffBits, SEEK_SET);
    image->aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * imageSize);
    fread(image->aBitmapBits, imageSize * sizeof(BYTE), 1, fp);

    fclose(fp);
    return image;
}
```

- 使用 `fread()` 完成对 BMP 图片的读取操作，读取模式为 `"rb+"`（二进制文件 + 可能需要调整图像大小对应的字段 `bmih.biSizeImage`）
- `bmih.biSizeImage` 的值可能为 0，所以需要手动处理这种情况（用总的文件大小减去偏移量）
- 注意图像数据的大小一定是 4 的倍数，因为每个像素点占 4 字节空间（RGB 各 1B + 保留位占 1B），如果不是的话需要补齐

3.3 生成灰度图

```
// 生成灰度图
BMPFILE GenerateGrayScale(BMPFILE bf) {
    BMPFILE gImg;
    RGB rgb;
    YUV yuv;
    BYTE val;
    int i, x, y;
    int pos;

    // 分配空间
    rgb = (RGB)malloc(sizeof(struct rgb));
    yuv = (YUV)malloc(sizeof(struct yuv));
    gImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(gImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(gImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 灰度图需要加上调色盘的大小
        gImg->bmfh.bfOffBits = (int)sizeof(BITMAPFILEHEADER) +
(int)sizeof(BITMAPINFOHEADER) + COLORNUM * (int)sizeof(RGBQUAD);
    gImg->bmih.biBitCount = 8;   // 灰度图单个像素只有 8 位（[0, 255]）
    // 计算宽度上的位数，同彩色图需要补齐至 4 的倍数
    GrayBitWidth = (gImg->bmih.biWidth + RGBNUM) / 4 * 4;
    // 灰度图的图像大小和文件大小也需要重新计算
    gImg->bmih.biSizeImage = GrayBitWidth * gImg->bmih.biHeight;
        gImg->bmfh.bfSize = gImg->bmfh.bfOffBits + gImg-
>bmih.biSizeImage;
        gImg->aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * gImg-
>bmih.biSizeImage);

    // 灰度图用到调色盘，因此需要赋值，且 RGB 的值是一致的
    for (i = 0; i < COLORNUM; i++) {
        gImg->aColors[i].rgbBlue = gImg->aColors[i].rgbGreen = gImg-
>aColors[i].rgbRed = i;
        gImg->aColors[i].rgbReserved = 0;
    }

    // RGB -> YUV，然后将 Y 作为灰度图的像素值
    for (y = 0; y < bf->bmih.biHeight; y++)
        for (x = 0; x < bf->bmih.biWidth; x++) {
```

```
            pos = y * ColorBitWidth + x * 3;
            rgb->B = bf->aBitmapBits[pos];
            rgb->G = bf->aBitmapBits[pos + 1];
            rgb->R = bf->aBitmapBits[pos + 2];
            yuv = rgb2yuv(rgb);
            val = rearrangeComp(yuv->Y);
            gImg->aBitmapBits[y * GrayBitWidth + x] = val;
        }

    return gImg;
}
```

- 关于调色盘的使用: 在彩色图中是没有调色盘这个字段的, 而在灰度图中用到了调色盘, 因此需要在原来图像信息头和数据之间插入调色盘的字段; 且由于每个像素点的取值为[0, 255]，所以调色盘的每个元素的 RGB 与它们的索引值相等
- 在遍历原图的图像数据时一定要注意位置问题: 原图的像素点是 3 字节的, 且顺序为 B、G、R，要根据图像的宽和高计算对应的 RGB 值，再转化为 YUV 色彩空间
- 转换过程中要注意越界问题，不要 Y 值低于 0 或高于 255，如果超过边界的话就让它的值等于边界值

3.4 改变原图亮度

```
// 改变图片的亮度
BMPFILE ChangeLuminance(BMPFILE bf) {
    BMPFILE lImg;
    RGB rgb;
    YUV yuv;
    BYTE val;
    int x, y;
    int pos;
    double time;

    // 分配空间
    rgb = (RGB)malloc(sizeof(struct rgb));
    yuv = (YUV)malloc(sizeof(struct yuv));
    lImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(lImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(lImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

       lImg->aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * lImg-
>bmih.biSizeImage);
```

```
    // 询问用户想要的亮度倍率
    printf("How many times do you want to adjust the luminance of
the image(double value): ");
    scanf("%lf", &time);
    if (time ⩽ 0) {
        printf("You should input a positive value!\n");
        exit(1);
    }
    sprintf(Time, "%.2f", time);

    // RGB → YUV, 通过改变 Y 来改变亮度, 然后再变回 RGB, 注意越界处理
    for (y = 0; y < bf→bmih.biHeight; y++)
        for (x = 0; x < bf→bmih.biWidth; x++) {
            pos = y * ColorBitWidth + x * 3;
            rgb = (RGB)malloc(sizeof(struct rgb));
            yuv = (YUV)malloc(sizeof(struct yuv));
            rgb→B = bf→aBitmapBits[pos];
            rgb→G = bf→aBitmapBits[pos + 1];
            rgb→R = bf→aBitmapBits[pos + 2];
            yuv = rgb2yuv(rgb);
            yuv→Y *= time;
            // 防止越界
            yuv→Y = rearrangeComp(yuv→Y);
            rgb = yuv2rgb(yuv);
            lImg→aBitmapBits[pos] = rearrangeComp(rgb→B);
            lImg→aBitmapBits[pos + 1] = rearrangeComp(rgb→G);
            lImg→aBitmapBits[pos + 2] = rearrangeComp(rgb→R);
        }

    return lImg;
}
```

- 在做亮度调整之前，会先询问用户的意见，用户需给出他们想要得到的亮度倍率
- 这里涉及到 RGB → YUV → RGB 的过程，且 Y 的值可能会放大多倍，因此需要同时考虑 RGB 和 YUV 值的越界问题

3.5 写入文件

```
// 将修改过的图片写入新的文件内
void WriteBMPFile(BMPFILE bf, int choice) {
    FILE * fp;
    std::string fname, path;
```

```
    unsigned long fisize;

    // 根据不同情况构建不同的文件名
    if (choice == 1) {
        fname = GRAYBMPFILE;
    } else {
       fname = std::string(LUMINANCEBMPFILE) + "_" + std::string(Time)
+ ".bmp";
    }

    // 打开新文件
    path = DIR + fname;
    fp = fopen(path.c_str(), "wb");

    if (!fp) {
        printf("Fail to create the bmp file!\n");
        exit(1);
    }

    // 先写入文件头和信息头，其中灰度图还需要写入调色盘
    fisize = sizeof(BITMAPFILEHEADER) + sizeof(tagBITMAPINFOHEADER);
    if (choice == 1) {
        fwrite(bf, fisize + COLORNUM * sizeof(RGBQUAD), 1, fp);
    } else {
        fwrite(bf, fisize, 1, fp);
    }
    // 后写入图像数据
      fwrite(bf->aBitmapBits, bf->bmih.biSizeImage * sizeof(BYTE),
1, fp);

    printf("Finish the mode %d conversion successfully!\n", choice);
    fclose(fp);
}
```

- 写入文件的流程会根据不同需求而有所调整
  - 生成灰度图：额外输出调色盘字段
  - 改变亮度：名称上会显示亮度倍率

3.6 主程序

```
int main() {
    int choice;
    BMPFILE oldImg, newImg;
```

```
    // 第 1 步：询问选择
    choice = AskforChoice();

    oldImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 第 2 步：读取 BMP 文件
    oldImg = ReadBMPFile();

    // 第 3 步：根据选择调整 BMP 文件
    if (choice == 1) {
        newImg = GenerateGrayScale(oldImg);
    } else {
        newImg = ChangeLuminance(oldImg);
    }

    // 第 4 步：写入新的 BMP 文件
    WriteBMPFile(newImg, choice);

    free(oldImg);
    free(newImg);
    return 0;
}
```

主程序完整地实现了上面的方法：从询问、读取、转换，最后到写入，整个过程比较简洁明了。

3.7 辅助函数

```
// RGB → YUV
YUV rgb2yuv(RGB model) {
    YUV yuv_model = (YUV)malloc(sizeof(struct yuv));

     yuv_model→Y = 0.299 * model→R + 0.587 * model→G + 0.114 *
model→B;
     yuv_model→U = -0.147 * model→R - 0.289 * model→G + 0.435 *
model→B;
     yuv_model→V = 0.615 * model→R - 0.515 * model→G - 0.100 *
model→B;

    return yuv_model;
}
```

```
// YUV → RGB
RGB yuv2rgb(YUV model) {
    RGB rgb_model = (RGB)malloc(sizeof(struct rgb));

    rgb_model→R = model→Y + 1.4075 * model→V;
    rgb_model→G = model→Y - 0.3455 * model→U - 0.7169 * model→V;
    rgb_model→B = model→Y + 1.779 * model→U;

    return rgb_model;
}

// 越界处理（超过 255 则赋值为 255，低于 0 则赋予 0）
BYTE rearrangeComp(double c) {
    if (c > COLORNUM - 1) {
        return COLORNUM - 1;
    } else if (c < 0) {
        return 0;
    }
    return (BYTE)c;
}
```

这些是我用到的其他函数，分别实现了 RGB 与 YUV 之间的转换，以及越界的处理。

四、 实验环境及运行方法

4.1 实验环境

我的开发环境如下：

- WSL2 + Ubuntu 24.04 LTS
- gcc version 13.2.0 (Ubuntu 13.2.0-23ubuntu4)
- GNU Make 4.3

4.2 运行方法

若要运行主程序，需要遵循以下步骤：

1. 将目录切换至 `./代码/build`
2. 执行以下命令：

```
# 编译代码
$ make

# 运行可执行文件
$ ./lab1
```

```
Welcome to the BMP file conversion program!
Here are some choice you can select:
Mode 1: Get a grayscale bmp file.
Mode 2: Get a luminance-changed bmp file.
Please input your choice(1 or 2):
```

此时可以选择生成灰度图（输入 1）或者获取亮度改变后的图像（输入 2），下面以输入 2 为例：

```
Please input your choice(1 or 2): 2

Valid choice!
Please wait a minute for the generation of the new file...
Successfully open the file!
Size: 14257206(bit)
ColorBitWidth: 8736
Width: 2912
Height: 1632
Image Size: 14257152

How many times do you want to adjust the luminance of the
image(positive double value): 1.5

# 输入指定的亮度倍率

Finish the mode 2 conversion successfully!
```

3. 来到 `./代码/tests` 目录，此时可以看到得到的新图

五、 实验结果展示

> **注意**
>
> 本报告采用 typst 书写，但是 typst 不支持插入 BMP 图像文件，因此这里展示的是它们的 PNG 形式，对应的 BMP 形式见目录 `./代码/tests`。

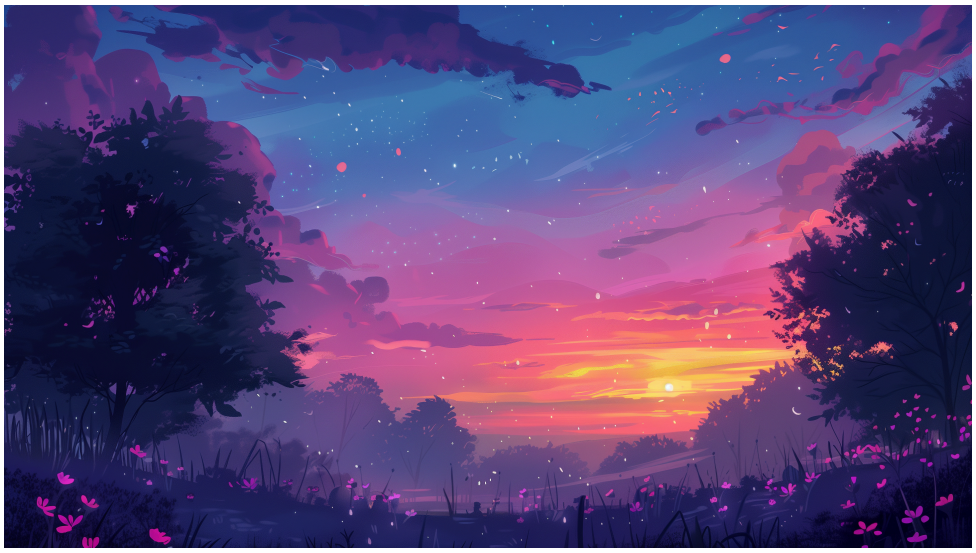

Figure 2: 原图（origin.bmp）

Figure 3: 灰度图（grayscale.bmp）

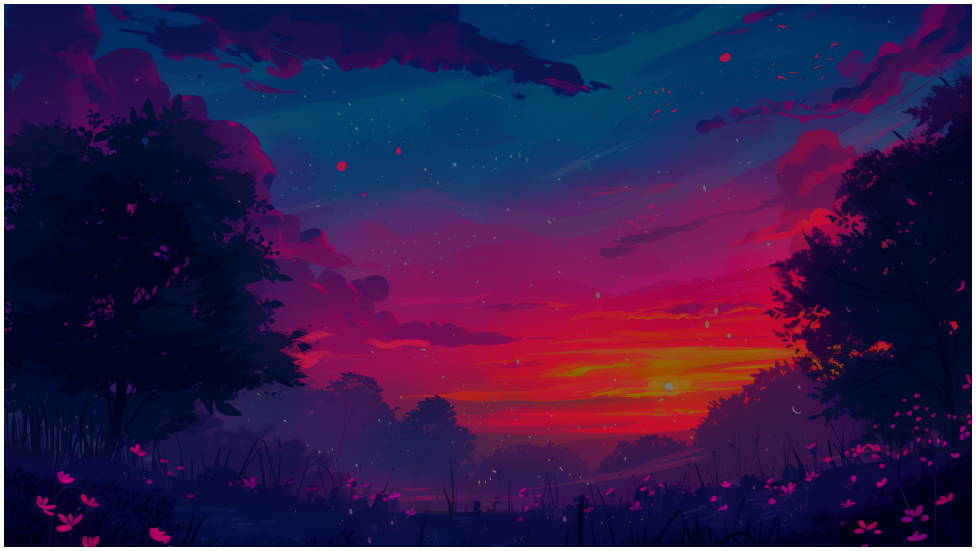

Figure 4: 亮度为原图 0.5 倍（change_luminance_0.50.bmp）

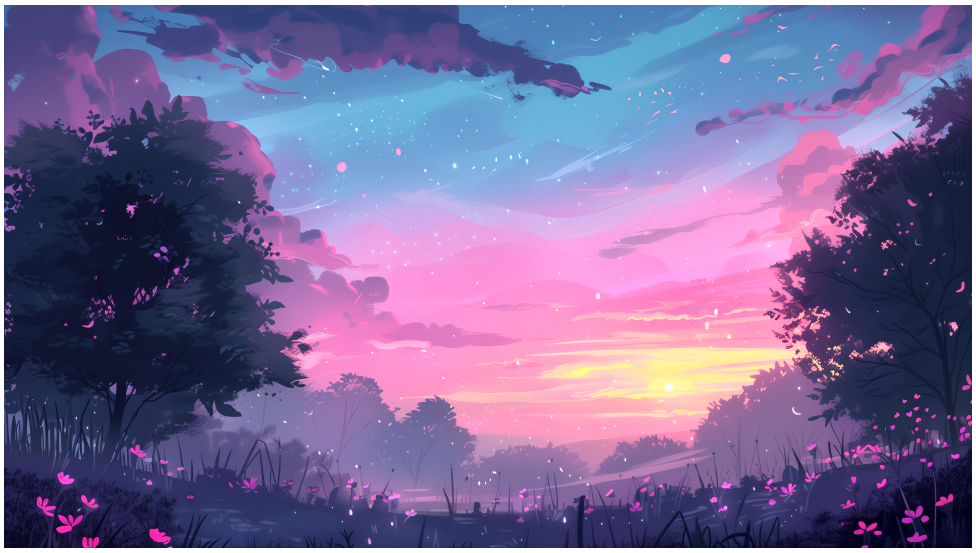

Figure 5: 亮度为原图 1.5 倍（change_luminance_1.50.bmp）

六、 心得体会

由于这是我第一次接触这类实验，所以刚上来肯定有点不适应，不知道从何下手。虽然代码文件是 C++ 文件，但是代码主体上我还是用 C 语言书写的，因为我更熟悉 C 语言文件读写的函数 `fread()` 和 `fwrite()`。代码的逻辑比较简单直接，但是要注意很多细节上的问题，比如如何定位到图像数据上的像素点以获取它的 RGB 值，RGB 与 YUV 转换的过程中产生的越界问题等等。下面列出我遇到的一些小阻碍：

- 定位图像像素点的公式搞错了（`x` 少乘了个 `3`），导致灰度图为原图左侧的一小块内容，并且还被拉伸了：

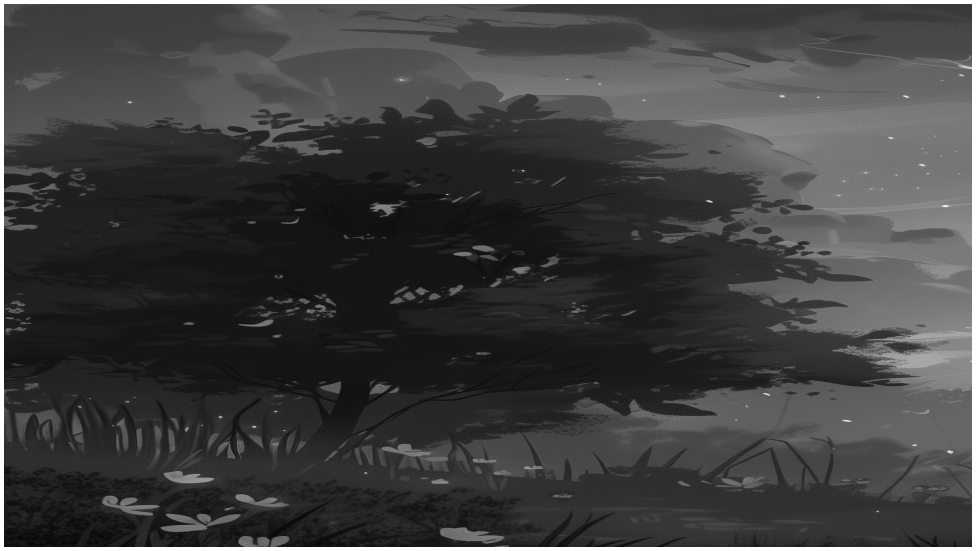

Figure 6: 失败品 1（failure/1.bmp）

- YUV → RGB 的过程中我又忘记处理越界问题，所以在改变亮度之后生成了一张气氛十分阴森的图片：

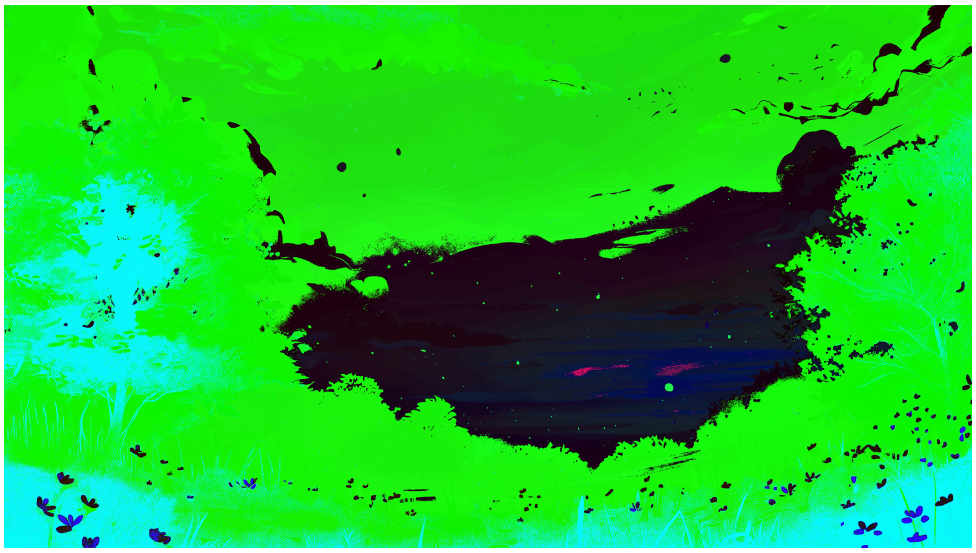

Figure 7: 失败品 2（failure/2.bmp）

- 我本来想先将 RGB 转 YUV 的公式复制给 YUV 转 RGB 的函数，再在上面的基础上修改，结果写着写着就忘记改了，也就是说 RGB → YUV → RGB 的过程变成了 RGB → YUV → YUV，因此图片变得乌漆嘛黑的：

Figure 8: 失败品 3（failure/3.bmp）

虽然一路上遇到过上面的磕磕绊绊，但好歹我也算摸爬滚打过来了，看到代码能够顺利运行，我不仅如释重负，也同时有一种说不出的满足感。总之这次实验让我深入学习和掌握了 BMP 文件格式，以及 RGB 和 YUV 之间的转换，使我受益良多。