**实验报告**

专业：计算机科学与技术

姓名：林海滨

学号：3220103645

日期：2023/11/14

课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： bitmap文件亮度的对数调整操作和直方图均衡操作

**一、实验目的和要求**

通过项目实践，了解bmp图像的直方图概念，并理解如何使用对数调整来平滑整体的亮度。另外，通过使用直方图均衡的操作，理解图像直方图操作的本质以及作用，以及理解应该在什么情况下适合使用直方图均衡。

本实验要求编写一个程序，读入一个bmp图像，首先通过对数操作调整这个图像的亮度，输出一张图片。然后通过直方图均衡操作，改变图像的亮度分布，再输出一张图片。

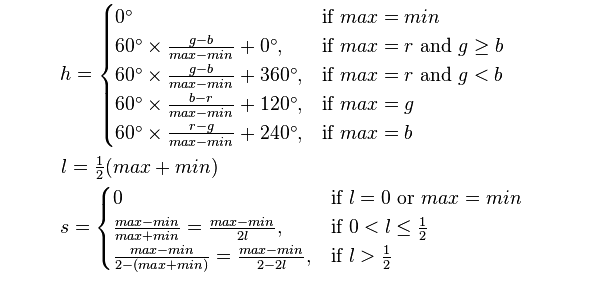
1. **实验内容和原理**
2. HSL色彩空间

之前的实验中，我使用的都是YUV色彩空间，这个色彩空间是简单的RGB通道的线性组合，通过寻找出合适的线性组合，来代表亮度和色调。然而这种色彩空间有一个弊端，也就是它的Y取值范围在[0,255]之间，对一些需要亮度值在[0,1]之间的操作不是非常方便，另外，YUV色彩空间不能直观地反映颜色的形象，比如U和V两者代表色调，但是它们的单独的值不能给我们一个直观的认识。相对的，HSL色彩空间的三个通道分别代表色调，饱和度和亮度，符合我们对色彩的直观认识，并且L也在[0,1]范围内，操作方便。

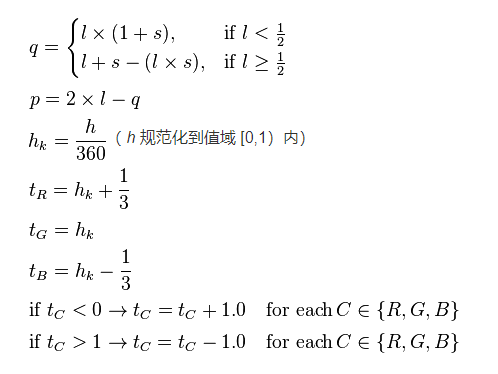
色调H表示颜色在色谱中的位置，以角度度量。在HSV中，H的取值范围通常是0°到360°，对应于色轮上的位置。饱和度S指颜色的纯度和强度。饱和度高表示颜色更纯净、饱满，而低饱和度则呈现灰调或淡色。饱和度值通常以百分比表示，从0%（灰度）到100%（最高饱和度）。亮度L表示颜色的明暗程度。L值通常以百分比表示，从0%（黑色）到100%（白色）。增加L值会使颜色更明亮，减小则使颜色变暗。

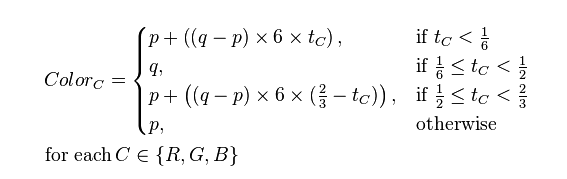
因此，将RGB转为HSL可以说是更优的一种做法。以下是RGB和HSL互相转化的公式：

RGB转HSL：



HSL转RGB：

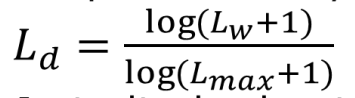




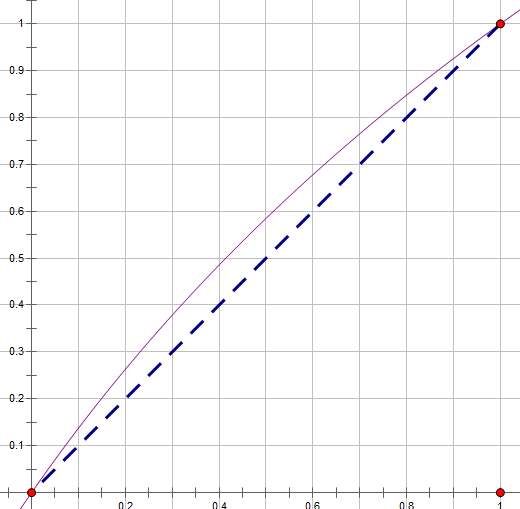
两者相互转化的两个比较重要的前置步骤，找最大最小值和归一化，其中归一化就是把RGB转为[0,1]范围的分量，方便计算。

1. 图像亮度的对数操作处理

对图像的亮度进行对数操作，即对原图片中的像素进行一个对数的映射，表达式如下：



Ld表示的是操作后的亮度，Lw表示的是操作前的亮度，Lmax表示操作前的最大亮度。我们不妨让Lmax就为1，画出这个函数的图像。

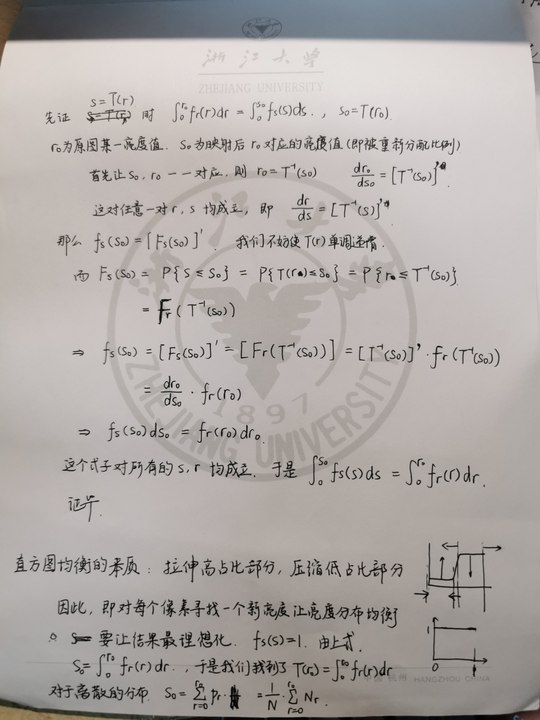


通过和Ld = Lw函数的对比，我们可以发现，实际上Ld在除了两端的亮度，都比Lw要高，也就是亮度被整体地拉高了，并且可以发现，亮度在中部拉升的更明显一些，这样可以让中间部的细节更加突出。

1. 图像亮度的直方图均衡

直方图是图像中各亮度级别的分布图，横轴表示亮度，纵轴表示像素数量。通过直方图，可了解图像的亮度分布，为调整图像的操作，如直方图均衡，提供信息。而直方图均衡本质上是通过对每个像素进行一个亮度的重新调整，从而让调整后的直方图趋向一个平均的状态，即每个亮度都平均地分布。举个例子，如果亮部的像素相比较少，说明图片整体较暗。那么通过直方图均衡，能将一些欠亮的像素变亮，从而显示更多的细节，同时增加了明暗对比。

直方图均衡的证明需要一些概率论知识，证明过程书写如下：



**三、实验步骤与分析**

我将这个程序分成以下步骤：

## **读入一张图片**

*// Step1: read an BMP*

    FILE \*Input = fopen("input.bmp", "rb");

    BMPMetric \*BMP = new BMPMetric();

ReadBMP(Input, BMP); *// In the mean time, HSL is converted*

## **将图片转成HSL颜色空间**

## 在第一步中，我们有一步ReadBMP，在其中包含了转为HSL的函数 Conv\_RGB\_HSL.

void Conv\_RGB\_HSL()

    {

        double Maximum = (MAX(MAX(R(), G()), B())) / 255.0;

        double Minimum = (MIN(MIN(R(), G()), B())) / 255.0;

        double R = R() / 255.0, G = G() / 255.0, B = B() / 255.0;

        if (Maximum == Minimum){  *// special case*

            H() = 0; L() = Maximum; S() = 0;

            return;

        }

*// calculate H*

        if (Maximum == R){

            if (G >= B) H() = 60.0 \* (G - B) / (Maximum - Minimum);

            else H() = 60.0 \* (G - B) / (Maximum - Minimum) + 360;

        }else if (Maximum == G) H() = 60.0 \* (B - R) / (Maximum - Minimum) + 120;

        else H() = 60.0 \* (R - G) / (Maximum - Minimum) + 240;

*// calculate l*

        L() = (Maximum + Minimum) / 2;

*// calculate S*

        S() = (L() < 1 / 2) ? (Maximum - Minimum) / (2 \* L()) : (Maximum - Minimum) / (2 \* (1 - L()));

    }

## **统计明度最大最小值**

    double MaxL = 0, MinL = 1;

    for (int i = 0; i < BMPInfoHeader.biHeight; i++)

        for (int j = 0; j < BMPInfoHeader.biWidth; j++)

        {

            double Lx = (\*BMP)[i][j].L();

            MaxL = MAX(Lx, MaxL);

            MinL = MIN(Lx, MinL);

        }

## **改变图像的Lightness值，转回RGB**

    for (int i = 0; i < BMPInfoHeader.biHeight; i++)

        for (int j = 0; j < BMPInfoHeader.biWidth; j++)

        {

            (\*BMP)[i][j].L() = (log((\*BMP)[i][j].L() + 1) / log(MaxL + 1));

            (\*BMP)[i][j].Conv\_HSL\_RGB();

        }

## **输出一张图片，完成对数操作**

    FILE \*Output1 = fopen("Output1.bmp", "wb+");

    WriteBMP(BMP, Output1);

## **统计明度的分布**

    long long \*LD = new long long[STRIP + 1];

    memset(LD, 0, sizeof(long long[STRIP + 1]));

    for (int i = 0; i < BMPInfoHeader.biHeight; i++)

        for (int j = 0; j < BMPInfoHeader.biWidth; j++)

        {

            Pixel pixel = (\*BMP)[i][j];

            int index = round(pixel.L() \* STRIP);

            LD[index]++;

        }

## **对每一个明度，计算均衡后对应的明度，存成表格**

    double \*L\_map = new double[STRIP + 1];

    long long N = BMPInfoHeader.biWidth \* BMPInfoHeader.biHeight;

    L\_map[0] = (double)LD[0] / N;

    for (int Light = 1; Light <= STRIP; Light++)

    {

        L\_map[Light] = L\_map[Light - 1] + (double)LD[Light] / N;

    }

## **遍历像素，根据明度对应表改变亮度，转回RGB**

    for (int i = 0; i < BMPInfoHeader.biHeight; i++)

        for (int j = 0; j < BMPInfoHeader.biWidth; j++)

        {

            int index = round((\*BMP)[i][j].L() \* STRIP);

            (\*BMP)[i][j].L() = L\_map[index];

            (\*BMP)[i][j].Conv\_HSL\_RGB(); *// turn back to RGB*

        }

## **输出一张图片，完成直方图均衡操作**

    FILE \*Output2 = fopen("Output2.bmp", "wb+");

    WriteBMP(BMP, Output2);

注：输入输出已在实验一中做阐释，此处不再赘述。

1. **实验环境及运行方法**

该程序使用vscode软件编写完成，可以使用MingW进行编译后，在执行程序的文件夹中放入input.bmp文件，输出output1.bmp，output2.bmp两个文件，分别对应对数操作，直方图均衡两种操作。目前经测试可以在windows系统上稳定运行。

**五、实验结果展示**

原图

原图

对数操作后

对数操作后

直方图均衡后

直方图均衡后

**六、心得体会**

本次实验我对实验的架构进行了一些变化，通过包装之前输入输出的函数为一个hpp文件，然后通过include函数进行引用，从而避免了大段的代码复制。因此整个代码也是非常简洁，实现过程也非常快。

实验中我遇到了三个BUG。第一个bug在于我在进行测试的时候，程序报饱和值大于1。通过调试复现，发现是因为浮点数精度问题，在饱和度为1时尾巴还存在一个极小值。因此将报错代码的阈值改为1.0001，解决了这个问题。

第二个BUG在于HSL转换为RGB时我发现色彩丢失，整个图像变为了黑白。我使用断点对相关的数据进行查看，发现在计算R、G、B分量时显示了相同的答案，经过排查发现是浮点常数的表示错误，比如在C语言中，三分之一不能表示为1/3(这实际上是0)，而需要表示为1.0/3,表示这是一个浮点数。

第三个BUG在于测试时，程序报亮度值小于0。这种情况很可能是因为没有赋初值（理论上不会溢出）。经过检查，发现把memset的三个参数位置写错了，正确的写法是memset(ptr,data,size)。而且，在使用int指针时，需要注意，int \*p = new int[k]; 语句虽然让p指向了一个分配好的数组空间，但是p本身指向的仍然是一个单个的int，所以sizeof(p)最终还是一个int，这值得注意。

另外，通过测试，我发现，实际上直方图均衡并不适用于所有的图。当一个图片本身就没有多少暗部时，强行平衡分布，把亮的像素拉暗会导致奇怪的视觉效果，就像下面两张图所显示的一样。因此，对直方图均衡的使用需要判断场合。

