

# 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：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。

## 二、实验内容和原理

### 1. 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  $0^{\circ}$  到  $360^{\circ}$ ，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:

$$h = \begin{cases} 0^\circ & \text{if } \max = \min \\ 60^\circ \times \frac{g-b}{\max-\min} + 0^\circ, & \text{if } \max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{\max-\min} + 360^\circ, & \text{if } \max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{\max-\min} + 120^\circ, & \text{if } \max = g \\ 60^\circ \times \frac{r-g}{\max-\min} + 240^\circ, & \text{if } \max = b \end{cases}$$
$$l = \frac{1}{2}(\max + \min)$$
$$s = \begin{cases} 0 & \text{if } l = 0 \text{ or } \max = \min \\ \frac{\max-\min}{\max+\min} = \frac{\max-\min}{2l}, & \text{if } 0 < l \leq \frac{1}{2} \\ \frac{\max-\min}{2-(\max+\min)} = \frac{\max-\min}{2-2l}, & \text{if } l > \frac{1}{2} \end{cases}$$

HSL 转 RGB:

$$q = \begin{cases} l \times (1 + s), & \text{if } l < \frac{1}{2} \\ l + s - (l \times s), & \text{if } l \geq \frac{1}{2} \end{cases}$$
$$p = 2 \times l - q$$
$$h_k = \frac{h}{360} \quad (h \text{ 规范化到值域 } [0,1) \text{ 内})$$
$$t_R = h_k + \frac{1}{3}$$
$$t_G = h_k$$
$$t_B = h_k - \frac{1}{3}$$

if  $t_C < 0 \rightarrow t_C = t_C + 1.0$  for each  $C \in \{R, G, B\}$   
if  $t_C > 1 \rightarrow t_C = t_C - 1.0$  for each  $C \in \{R, G, B\}$

$$Color_C = \begin{cases} p + ((q - p) \times 6 \times t_C), & \text{if } t_C < \frac{1}{6} \\ q, & \text{if } \frac{1}{6} \leq t_C < \frac{1}{2} \\ p + ((q - p) \times 6 \times (\frac{2}{3} - t_C)), & \text{if } \frac{1}{2} \leq t_C < \frac{2}{3} \\ p, & \text{otherwise} \end{cases}$$

for each  $C \in \{R, G, B\}$

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。

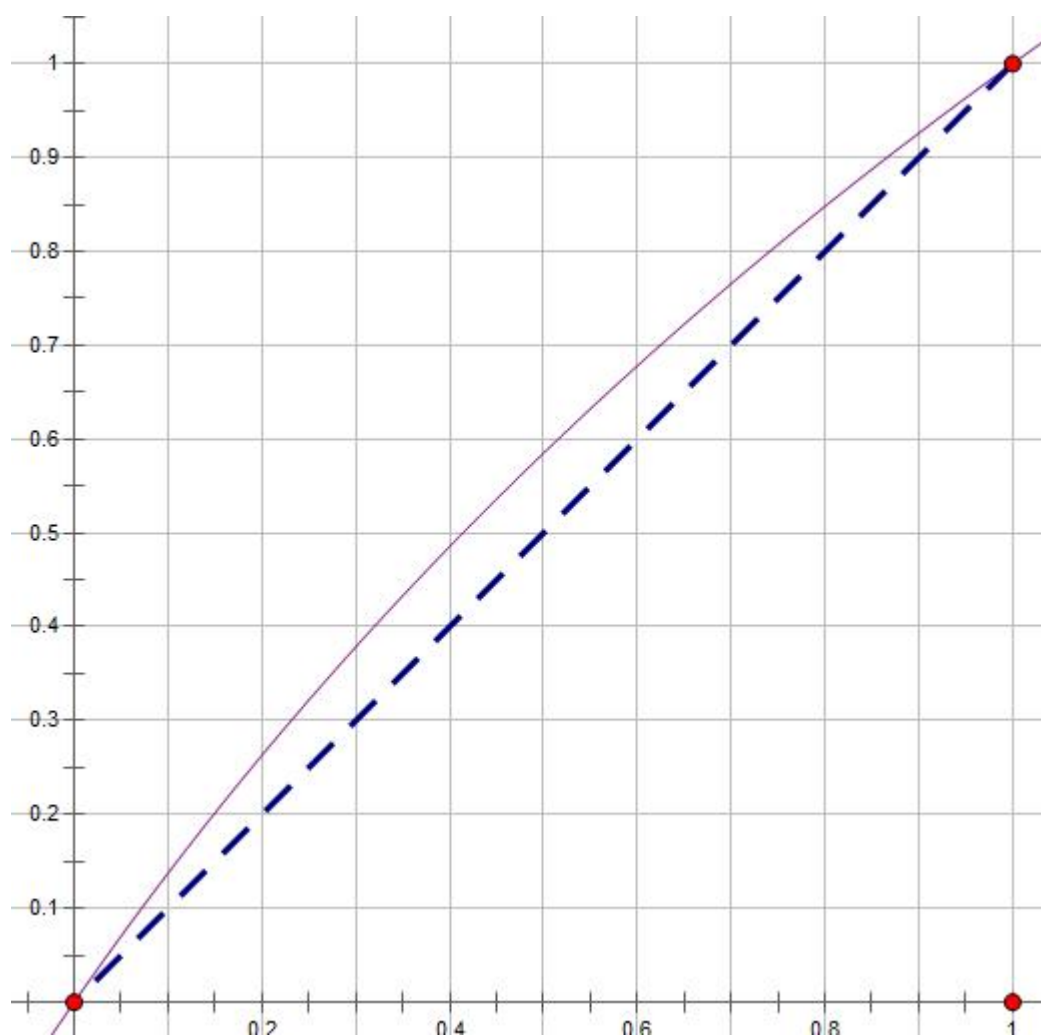
## 2. 图像亮度的对数操作处理

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：

$$L_d = \frac{\log(L_w + 1)}{\log(L_{max} + 1)}$$

$L_d$  表示的是操作后的亮度， $L_w$  表示的是操作前的亮度， $L_{max}$  表示操作前的最大亮度。

我们不妨让  $L_{max}$  就为 1，画出这个函数的图像。



通过和  $L_d = L_w$  函数的对比，我们可以发现，实际上  $L_d$  在除了两端的亮度，都比  $L_w$  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，这样可以使中间部的细节更加突出。

### 3. 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：

浙江大學  
ZHEJIANG UNIVERSITY

先证  $s = T(r)$  时  $\int_0^{r_0} f_r(r) dr = \int_0^{s_0} f_s(s) ds$  ,  $s_0 = T(r_0)$

$r_0$  为原图某一亮度值.  $s_0$  为映射后  $r_0$  对应的亮度值 (即被重新分配比例)

首先让  $s_0, r_0$  一一对应, 则  $r_0 = T^{-1}(s_0)$   $\frac{dr_0}{ds_0} = [T^{-1}(s_0)]'$

这对任意一对  $r, s$  均成立, 即  $\frac{dr}{ds} = [T^{-1}(s)]'$

那么  $f_s(s_0) = [F_s(s_0)]'$  我们不妨使  $T(r)$  单调递增.

而  $F_s(s_0) = P\{s \leq s_0\} = P\{T(r_0) \leq s_0\} = P\{r_0 \leq T^{-1}(s_0)\}$

$= F_r(T^{-1}(s_0))$

$\Rightarrow f_s(s_0) = [F_s(s_0)]' = [F_r(T^{-1}(s_0))]' = [T^{-1}(s_0)]' \cdot f_r(T^{-1}(s_0))$

$= \frac{dr_0}{ds_0} \cdot f_r(r_0)$

$\Rightarrow f_s(s_0) ds_0 = f_r(r_0) dr_0$

这个式子对所有的  $s, r$  均成立, 于是  $\int_0^{s_0} f_s(s) ds = \int_0^{r_0} f_r(r) dr$ .

证毕.


直方图均衡的本质: 拉伸高占比部分, 压缩低占比部分

因此, 即对每个像素寻找一个新亮度让亮度分布均衡

o 要让结果最理想化.  $f_s(s) = 1$ . 由上式.

$s_0 = \int_0^{r_0} f_r(r) dr$ , 于是我们找到了  $T(r) = \int_0^r f_r(r) dr$

对于离散分布  $s_0 = \sum_{r=0}^{255} p_r \cdot \frac{1}{N} = \frac{1}{N} \sum_{r=0}^{255} N_r$



---

### 三、实验步骤与分析

我将这个程序分成以下步骤：

#### 1. 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
```

#### 2. 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()));
}
```

---

### 3. 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);
    }
```

### 4. 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();
    }
```

### 5. 输出一张图片，完成对数操作

```
FILE *Output1 = fopen("Output1.bmp", "wb+");
WriteBMP(BMP, Output1);
```

### 6. 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]++;
    }
```

## 7. 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;
}
```

## 8. 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
    }
```

## 9. 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。

## 三、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



对数操作后



直方图均衡后



原图



对数操作后



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检查，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。

