专业：计算机科学与技术
姓名：龙永奇
学号：3220105907
日期：2023/11/11

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：图像增强的对数运算和直方图均衡化

一、实验目的和要求

1. 通过对数运算增强图像可视化
2. 直方图均衡化

二、实验内容和原理

1. 增强图像可视化

为增强图像可视化，可对图像像素进行如下对数操作：

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

其中 L_d 为显示出来的亮度，即所要求的值，L_w 为真实世界的亮度，即当前值，L_{max} 是场景中的最亮值，即最大值。此函数映射可确保无论场景动态范围如何变化，其最大值都能映射到(1，1，1)，即白色，并且其他值变化较为平滑。

2. 灰度直方图

灰度图在之前的作业中已经学习并实现。

灰度值直方图，顾名思义，是一种统计图，给定了图像中不同灰度级像素所占总像素数的比例。

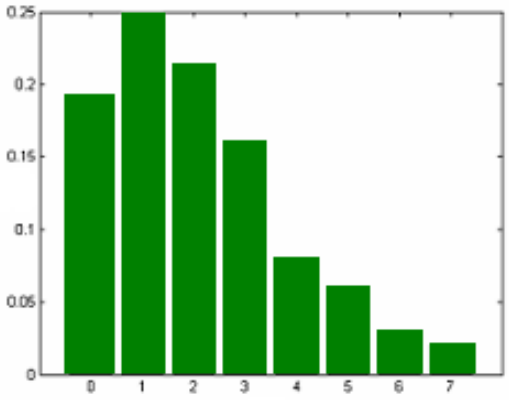
其表示方法如下：

对于给定的 [0, L-1] 级灰度，灰度直方图的离散形式可表达为：

$$h(r_k) = n_k$$

其中 r_k 为第 k 级灰度，n_k 是 r_k 的像素个数，通过概率密度函数对直方图进行归一化：

$$P(r_k) = \frac{n_k}{n}$$



由概率论相关知识可得，直方图数据满足以下关系：

$$\sum_{k=0}^{L-1} P(r_k) = 1$$

3. 直方图均衡化

直方图有以下特征：

- 基于空间域的处理技术基础，反映图像的灰度分布，但不能传达图像中视觉信息的结构变化
- 特定图像的唯一性，不同的图像可能共享相同的直方图
- 直方图可以用于图像增强、压缩和分割，是图像处理的实用手段

直方图有以下作用：

- 使用轮廓线确定物体边界时，通过直方图可以更好的选择边界阈值并进行阈值化处理
- 可通过直方图求出简单物体的面积和综合光密度
- 对物体和背景有较强对比度的景物分割有明显作用

直方图均衡化原理：

直方图均衡化通过改变图像的灰度分布，使得图像中的像素灰度值更加均匀分布，从而增强图像的对比度。

在原始图像中，灰度分布可能集中在较窄的范围内，导致图像缺乏清晰度。例如，过曝光的图像可能使灰度级集中在高亮度范围内，而曝光不足则会使灰度级集中在低亮度范围内。通过直方图均衡化使得图像中**主要灰度值**的像素得到**更广泛的展开**，而**次要灰度值**则被**归并**。从而增加像素之间灰度值差异的动态范围，提升了图像的对比度。

直方图均衡化过程：

为实现直方图均衡化，我们需要找到一个映射 T，使得非均匀分布的直方图在映射 T 变换后变为均匀直方图：

$$s = T(r)$$

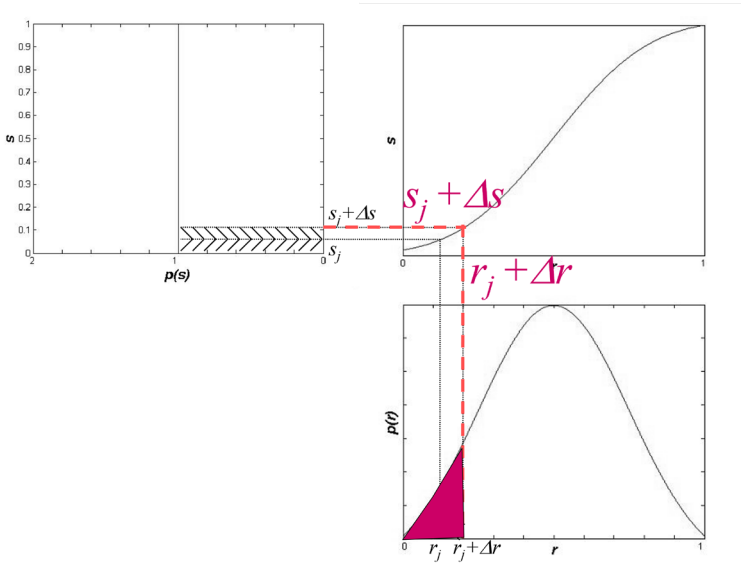
- **连续情况下**

设 r 和 s 表示变换前后的灰度级，P(r)和 P(s)对应 r 和 s 的概率 $0 \le r \quad s \le 1$

T(r) 是单调递增函数，$0 \le r \le 1$ 且 $0 \le T(r) \le 1$ 其逆变换 $r = T^{-1}(s)$ 亦是一个单调递增函数。由于灰度变换不会改变位置和像素数：

$$\int_0^r P(r)dr = \int_0^s P(s)ds = \int_0^s 1ds = s = T(r)$$

由此可得，s = T(r)是原始图像中 [0, r] 之间的直方图曲线覆盖面积：



- **离散情况下：**

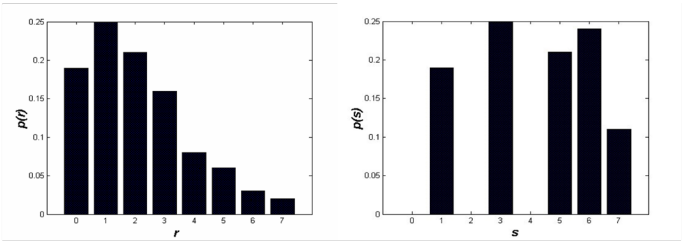
假设图像的总像素数为 n，分为 L 个灰度级，n_k 是灰度级为 k 的像素数，那么 k 灰度级的概率为：

$$P(r_k) = \frac{n_k}{n}(0 \le n_k \le 1, k = 0, 1, 2, ..., L-1)$$

可得离散灰度直方图均衡转化公式为：

$$s_k = T(r_k) = \sum_{i=0}^{k} P(r_i) = \sum_{i=0}^{k} \frac{n_i}{n} = \frac{1}{n}\sum_{i=0}^{k} n_i$$

即 s_k 是由 $[0, r_k]$ 之间的所有像素数相加除以总像素数得到：



(a)Before histogram equalization (b) After histogram equalization

三、实验步骤与分析

1. 对数化增强

首先找出最大灰度值 max，再对每一个像素进行 YUV 转化后对 Y 通道进行对数增强，分别储存灰度图和彩色图（还要进行一次 RGB 转换）源代码如下：

```
void Log(int BMPheight, int BMPwidth){
    int row_byte = (3 * BMPwidth + 3) / 4 * 4;
    int byte_row = (BMPwidth + 3) / 4 * 4;
    int now_position,new_position;
    double max=0;
    // 遍历像素找出最大灰度值 max
    for (int i = 0; i < BMPheight; i++){
        for (int j = 0; j < BMPwidth; j++){
            now_position = row_byte * i + 3 * j;
            double B = zjz.pix_val[now_position];
            double G = zjz.pix_val[now_position + 1];
            double R = zjz.pix_val[now_position + 2];
            double Y, U, V;
            RGB_YUV(R, G, B, &Y, &U, &V);
            if (max < Y) max = Y;
        }
    }
    // 遍历每一个像素并在转换成 YUV 空间后对 Y 进行对数增强
    for(int i=0; i<BMPheight;i++){
        for(int j=0;j<BMPwidth;j++){
            now_position = row_byte * i + 3 * j;
            new_position = byte_row * i + j;
            double B = zjz.pix_val[now_position];
            double G = zjz.pix_val[now_position + 1];
            double R = zjz.pix_val[now_position + 2];
            double Y, U, V;
            RGB_YUV(R, G, B, &Y, &U, &V);
            Y = new(log(Y+1)/log(max+1)*255); // 在这里对数增强
            zjz1.pix_val[new_position]=Y;
            YUV_RGB(Y,U,V,&R,&G,&B);
            zjz2.pix_val[now_position]=new(B);
            zjz2.pix_val[now_position+1]=new(G);
            zjz2.pix_val[now_position+2]=new(R);
        }
    }
}
```

2. **计算各通道像素数**

其中四个数组分别储存 Y,R,G,B 四个通道的每一值对应的像素数，源代码如下：

```
// 首先计算 Y 和 RGB 每一通道对应的像素个数，像素总数为 BMPheight*BMPwidth
for(int i=0; i<BMPheight;i++){
    for(int j=0;j<BMPwidth;j++){
        now_position = row_byte*i+3*j;
        double B = zjz.pix_val[now_position];
        double G = zjz.pix_val[now_position + 1];
        double R = zjz.pix_val[now_position + 2];
        double Y, U, V;
        RGB_YUV(R, G, B, &Y, &U, &V);
        pix_R[(unsigned char)new(R)]+=1.0; // R 通道
        pix_G[(unsigned char)new(G)]+=1.0; // G 通道
        pix_B[(unsigned char)new(B)]+=1.0; // B 通道
        pix_Y[(unsigned char)new(Y)]+=1.0; // Y 通道
    }
}
```

3. **对各通道进行直方图均衡化变换**

对 i = 0 情况特殊化，源代码如下：

```
for(int i=0;i<256;i++){
    if(i==0){ // i=0 时和其他情况区分开
        sum_pix_Y[i]=pix_Y[i]/(BMPheight*BMPwidth);
        sum_pix_R[i]=pix_R[i]/(BMPheight*BMPwidth);
        sum_pix_G[i]=pix_G[i]/(BMPheight*BMPwidth);
        sum_pix_B[i]=pix_B[i]/(BMPheight*BMPwidth);
    }else{ // 0 到 i 之间所有像素数相加除以总像素数
        sum_pix_Y[i]=sum_pix_Y[i-1]+pix_Y[i]/(BMPheight*BMPwidth);
        sum_pix_R[i]=sum_pix_R[i-1]+pix_R[i]/(BMPheight*BMPwidth);
        sum_pix_G[i]=sum_pix_G[i-1]+pix_G[i]/(BMPheight*BMPwidth);
        sum_pix_B[i]=sum_pix_B[i-1]+pix_B[i]/(BMPheight*BMPwidth);
    }
}
```

4. **进行直方图均衡化（离散）**

本次实现了三种直方图均衡化算法：

- 在 Y 分量上进行均衡化并输出彩色图，源代码如下：

```
Y=zjz3.pix_val[new_position]=sum_pix_Y[(unsigned char)new(Y)]*255;
```

- 在 RGB 三个分量上分别均衡化，输出彩色图，源代码如下：

```
zjz4.pix_val[now_position]=sum_pix_B[(unsigned char)new(B)]*255;
zjz4.pix_val[now_position+1]=sum_pix_G[(unsigned char)new(G)]*255;
zjz4.pix_val[now_position+2]=sum_pix_R[(unsigned char)new(R)]*255;
```

➢ 在 Y 分量进行均衡化，再转换成 RGB 空间，输出彩色图，源代码如下：

```
Y=sum_pix_Y[(unsigned char)new(Y)]*255;
YUV_RGB(Y,U,V,&R,&G,&B);
zjz5.pix_val[now_position]=new(B);
zjz5.pix_val[now_position+1]=new(G);
zjz5.pix_val[now_position+2]=new(R);
```

四、实验环境及运行方法

1. 实验环境

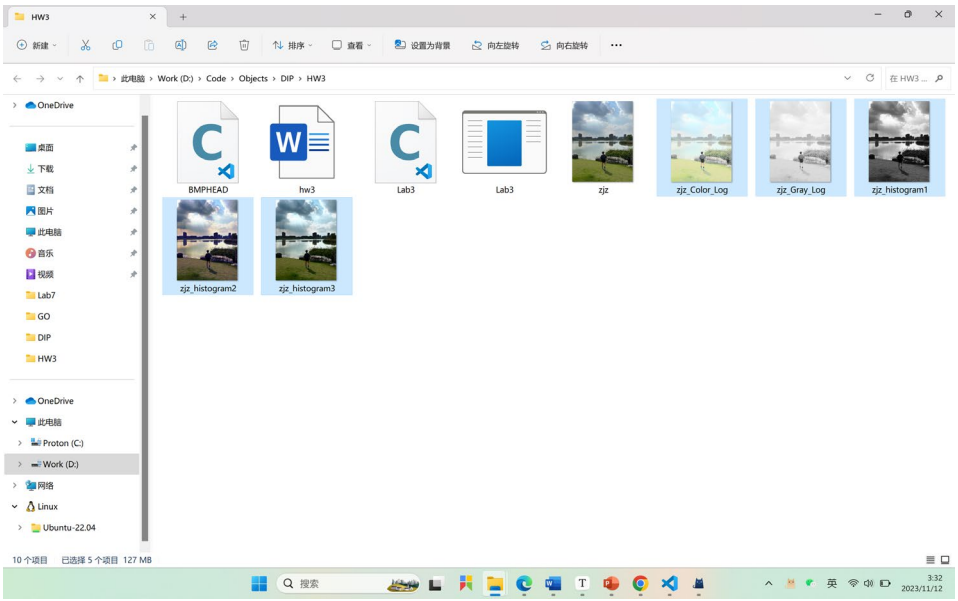
系统 Windows11 编译器 gcc 10.3.0x86_mingw32

2. 运行方法

在文件夹中，Lab3.c 为源文件，运行代码，如果出现“憨憨在这里:)”说明读取图片文件成功，否则会出现“憨憨不见了: (” 随后程序会分别输出：

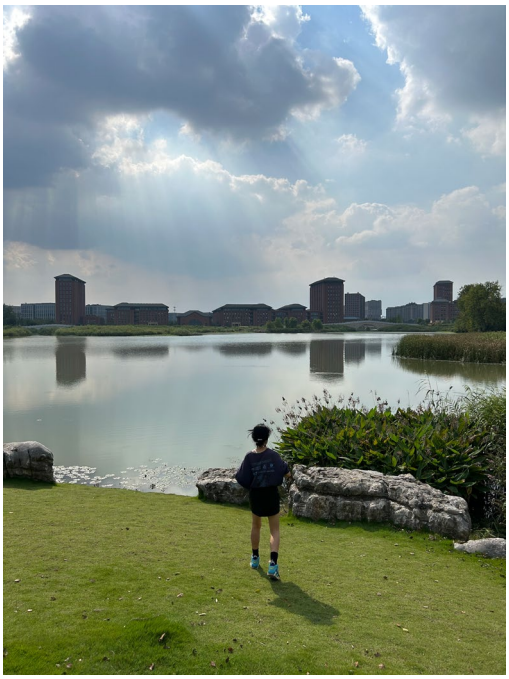
➢ 对数增强灰度图像：zjz.Gray_Log.bmp，彩色图像图像：zjz_Color_Log.bmp

➢ 直方图均衡化算法 1：zjz_histogram1.bmp，算法 2：zjz_histogram2.bmp，算法 3：zjz_histogram3.bmp



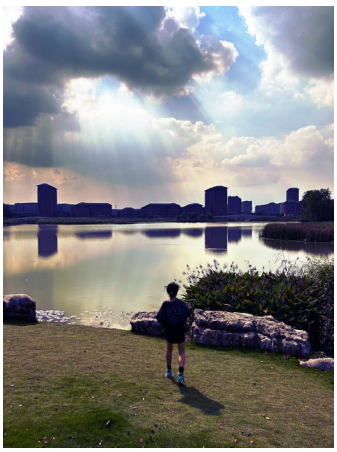
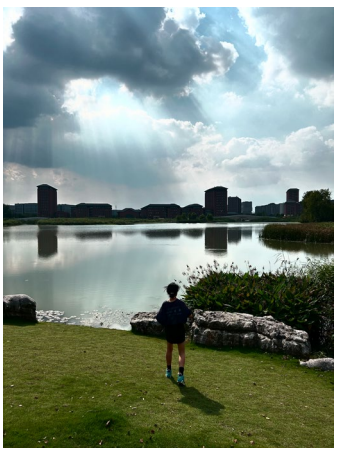
五、实验结果展示

1. 原图像



2. 对数增强灰度图像与彩色图像

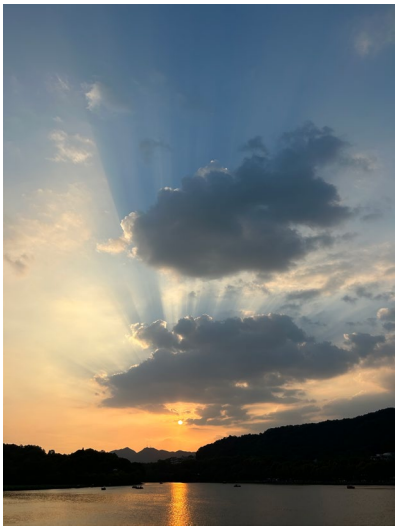

3. 直方图均衡化算法三种图像

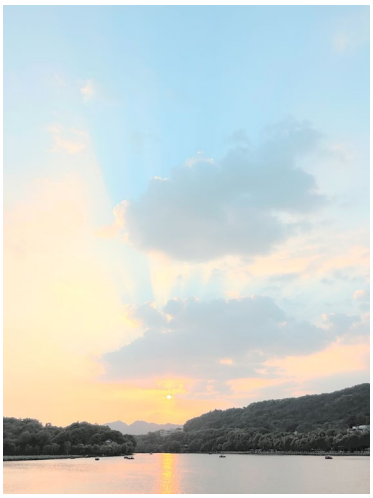




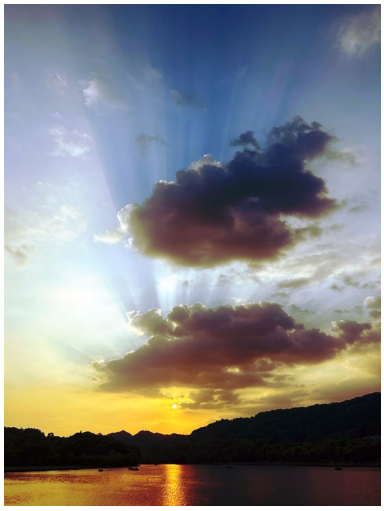
4. 此外还准备了两组测试样例：

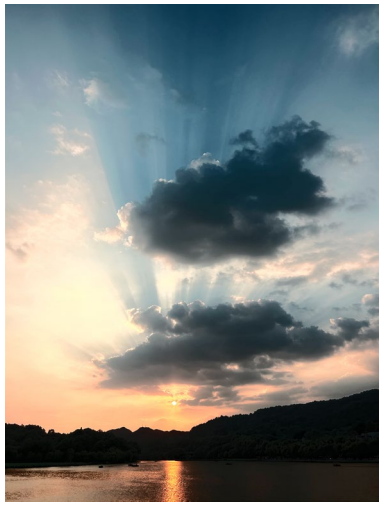
- 测试样例一：

- 测试样例二：









六、心得体会

本次实验通过对对数增强以及直方图均衡两种算法的学习与实践，完成了对图像可视化增强的操作。

由三个实验结果可见，对数增强会使得图像被贴上一层膜，在第三组样例中可见，相对于明亮的天空和水面，亮度较低的山脉细节得以显现，而直方图均衡化给人的视觉效果是去掉膜后的对数增强，并且不同的两种算法会产生不同的色调，其原因就是改变 RGB 和改变 Y 导致图像不同分量被改变，对色调造成一定影响，但相比于对数增强，此算法对亮度较低的山脉处细节显现效果有限。