专业：计算机科学与技术
姓名：
学号：
日期：2021/10/29

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：灰度图直方图均衡化和对数处理

一、实验目的和要求

1. 直方图均衡化
2. 对数处理

二、实验内容和原理

1. 直方图均衡化

对于灰度级范围为$[0,\ L-1]$的数字图像，其直方图可以表示为一个离散函数$h(r_k)=n_k$，其中r_k是第k级灰度值，n_k是图像中灰度值为r_k的像素个数，也就是说图像的灰度直方图表征的是该图像的灰度分布。进行归一化之后可以表示为$p(r_k)=\frac{n_k}{MN}$，也就是说$p(r_k)$表示灰度级r_k在图像中出现概率的估计。

如果一张图像的灰度值几乎覆盖整个灰度的取值范围，并且除了个别灰度值比较突出，整个灰度值分布近似与均匀分布，那么这幅图像就具有较大的灰度动态范围和较高的对比度。所以我们想找到一个变换函数能够达到上述的效果。

对于连续型灰度图像，我们可以构造出一个重要的变换函数为

$$s=T(r)=(L-1)\int_0^r p_r(w)dx$$

满足$T(r)$在$[0,L-1]$严格单调递增；且在$0\le r\le L-1$时，$0\le T(r)\le L-1$。

而对于离散型的函数，我们可以构造出下面这个式子：

Histogram equalization—find T (discrete)

设一幅图像的像素总数为n，分L个灰度级，n_k为第k个灰度级出现的像素数，则第k个灰度级出现的概率为：

$$P(r_k)=n_k/n \quad (0\le r_k\le 1,\ k=0,1,2,...,L-1)$$

离散灰度直方图均衡化的转换公式为：

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

通过这个变换就可以实现灰度图像的直方图均衡化。

2. 对数处理

对数处理目的是提高图像的能见度，使用如下的公式进行对数处理

- In order to enhance the image's visibility, adjust the pixel value by a logarithmic operator.
- $L_d = \frac{\log(L_w+1)}{\log(L_{max}+1)}$
- L_d is display luminance, L_w is the real luminance, L_{max} is the maximal luminance value in the image.
- This mapping function make sure that: no matter the dynamic range of the scene, the maximal luminance value will be 1 (white), and other values changes smoothly.

注意：进行对数处理之后，得到的L_d是 0 到 1 的一个数，所以需要乘以一个比例系数，这个比例系数是自己任意取的，笔者代码中使用的是 255。

三、实验步骤与分析

0. 优化图像的读取和写入

在之前的实验中笔者已经实现了图像的读取和写入，但是代码的结构并不清晰，笔者在这次实验中继续优化代码，使得结构更加清晰，并且目前支持读入 8 位图片和 24 为图片。

下面是笔者的数据结构：

```
typedef unsigned char       BYTE;
typedef unsigned short      WORD;
typedef unsigned int        DWORD;
typedef unsigned int        LONG;

typedef struct tagBITMAPFILEHEADER {
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;

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;
```

```
typedef struct tagRGBQUAD {
        BYTE    rgbBlue;
        BYTE    rgbGreen;
        BYTE    rgbRed;
        BYTE    rgbReserved;
} RGBQUAD;

typedef struct {
        BYTE R;
        BYTE G;
        BYTE B;
} Pixel;

typedef struct {
        BITMAPFILEHEADER *bmpFileHeader;
        BITMAPINFOHEADER *bmpInfo;
        RGBQUAD *bmpRgbQuad;
        BYTE *bmpData;
} Image;
```

具体的实现和之前实验的实验方法类似，只是将文件头、信息头、调色板和图像信息都放

在 Image 结构体中，使得代码结构更加清晰。

1. 直方图均衡化

直方图均衡化主要是根据上面的离散型公式。我们主要使用了 4 个数组，分别是存储各灰度值像素数量的数组 grayNum、存储各个灰度值像素比例的 grayRatio、存储灰度值累积比例的 grayAccumulateRatio、存储各灰度值经过直方图均衡化的新的值。因此我们首先初始化各个数组，并计算所有的像素值。

```
int i, j;
Image *bmpImageHE = Copy(bmpImage);
int N = bmpImageHE→bmpInfo→biWidth * bmpImageHE→bmpInfo→biHeight;
int grayNum[256] = {0}; // save the num of pixels of every grayscale
double grayRatio[256] = {0}; // save the ratio of grayNum
double grayAccumulateRatio[256] = {0}; // save the accumulate
int grayHE[256] = {0}; // save transformed grayscale after HistogramEqualization
```

然后我们统计各个灰度值下像素的数量。

```
// calculate the num of pixels of every grayscale
for (i = 0; i < bmpImageHE→bmpInfo→biHeight; i++)
    for (j = 0; j < bmpImageHE→bmpInfo→biWidth; j++)
        grayNum[bmpImageHE→bmpData[i * bmpImageHE→bmpInfo→biWidth + j]]++;
```

由此计算每个像素值下像素占总的像素的比例。

```
// calculate the ratio
for (i = 0; i < 256; i++)
    grayRatio[i] = (double)grayNum[i] / N;
```

再通过上面的公式计算累积灰度比例。

```
// calculate the accumulate ratio
grayAccumulateRatio[0] = grayRatio[0];
for (i = 1; i < 256; i++)
    grayAccumulateRatio[i] = grayAccumulateRatio[i - 1] + grayRatio[i];
```

接下来就对每一个灰度值进行转换。转换的公式是

$$f(x) = GrayAccumulateRatio(x) * 255 + 0.5$$

上式x表示是 0 到 255 的各个整数灰度值。

```
// calculate the transformed grayscale after histogram equalization
for (i = 0; i < 256; i++)
    grayHE[i] = grayAccumulateRatio[i] * 255 + 0.5;
```

最后就根据上面得到的转换表再进行对整个图像的灰度的更新。

```
// update all the pixels
for (i = 0; i < bmpImageHE->bmpInfo->biHeight; i++)
    for (j = 0; j < bmpImageHE->bmpInfo->biWidth; j++)
        bmpImageHE->bmpData[i * bmpImageHE->bmpInfo->biWidth + j] = grayHE[bmpImageHE->bmpData[i * bmpImageHE->bmpInfo->biWidth + j]];
```

2. 对数图像处理

对数图像处理主要是使用在原理中提到的公式进行图像优化。

```
// Find the maximum luminance
for (i = 0; i < bmpImageLog->bmpInfo->biHeight; i++) {
    for (j = 0; j < bmpImageLog->bmpInfo->biWidth; j++) {
        if (bmpImageLog->bmpData[i * bmpImageLog->bmpInfo->biWidth + j] > maxL)
            maxL = bmpImageLog->bmpData[i * bmpImageLog->bmpInfo->biWidth + j];
    }
}
```

首先是找到最大的灰度值。

```
// calculate the new luminance
for (i = 0; i < bmpImageLog->bmpInfo->biHeight; i++) {
    for (j = 0; j < bmpImageLog->bmpInfo->biWidth; j++) {
            currentL = bmpImageLog->bmpData[i * bmpImageLog->bmpInfo->biWidth + j];
            newL = log((double)currentL + 1) / log((double)maxL + 1) * coefficient;
            bmpImageLog->bmpData[i * bmpImageLog->bmpInfo->biWidth + j] = newL;
    }
}
```

然后如上，使用之前的公式对每个像素进行优化。注意上面的 coefficient 是自己设置的超参数，笔者在 main.c 中使用的是 255。

四、实验环境及运行方法

编译环境：

gcc 6.3.0、Windows 11 Insider Preview 22483.1011

测试方法：

在命令行中输入 gcc image.c main.c，然后运行.\a.exe。本输入图片为 8 位灰度 BMP 文件，文件名为 gray.bmp（放在源代码同一级目录中），输出的图片有对数处理之后的 8 位灰度图 outputLog.bmp 和经过直方图均衡化后的 8 位灰度图 outputHisteq.bmp。要修改对数处理的系数，可以更改 main.c 第 11 行的 255 值。

五、实验结果展示

8 位灰度图	
经过对数处理	

经过直方图均衡化	

六、心得体会

在本次实验中我对于灰度图的操作有了更深的认识，对数处理和直方图均衡化是两种非常有用的图像处理技巧，实现这两种算法让我感到非常有成就感。这次我继续优化了代码结构使得之后可以不断复用。