实验二：直方图均衡

学号：SA22225286 姓名：孟寅磊 日期：20220921

实验内容

> 1. 计算灰度图像的归一化直方图
>
> 具体内容：利用OpenCV对图像像素进行操作，计算归一化直方图，并在窗口中以图形的方式显示出来。
>
> 2. 灰度图像直方图均衡处理
>
> 具体内容：通过计算归一化直方图，设计算法实现直方图均衡化处理。
>
> 3. 彩色图像直方图均衡处理
>
> 具体内容：在灰度图直方图均衡处理的基础上实现彩色直方图均衡处理。

实验完成情况

- [x] 计算灰度图像的归一化直方图

令$r_k, k=0,1,2,\cdots,L-1$表示一幅L级灰度数字图像$f(x,y)$的灰度。f的非归一化直方图定义为

$$h(r_k)=n_k, \quad k=0,1,2,\cdots,L-1 \tag{2.1}$$

式中，n_k是f中灰度为r_k的像素的数量。类似的，f的归一化直方图定义为

$$p(r_k)=\frac{h(r_k)}{MN}=\frac{n_k}{MN} \tag{2.2}$$

式中，M和N分别是图像的行数和列数。在本次实验中，我们只需要统计每种灰度级的像素出现的次数，并计算出其概率，就可以得到其归一化直方图。

```cpp
// 计算归一化直方图
Mat display_histogram(Mat &src) {
    // 区间数目
    const int bins = 256;
    // 统计待处理图像的灰度级分布
    vector<int> gray_scales_counter(bins, 0);
    int rows = src.rows;
    int cols = src.cols;
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            ++gray_scales_counter[src.at<uchar>(i, j)];
    // 计算归一化直方图
    int size = rows * cols;
    vector<double> histogram(bins, 0.0);
    for (int i = 0; i < bins; ++i)
        histogram[i] = 1.0 * gray_scales_counter[i] / size;
    // 获得直方图中的最大值
    double max_val = *max_element(histogram.begin(), histogram.end());
    // 绘制直方图
    int hist_h = bins;
```

```
    int factor = 2;
    int hist_w = bins * factor;
    Mat hist_image(hist_h, hist_w, CV_8UC1, Scalar(0));
    for (int i = 0; i < bins; ++i) {
        rectangle(hist_image, Point(i * factor, hist_h - 1),
                  Point((i+1) * factor , hist_h -
cvRound((histogram[i]/max_val) * hist_h)),
                  Scalar(255), FILLED);
    }
    return hist_image;
}
```

- [x] 灰度图像直方图均衡处理

使用图像处理中一个特别重要的变换函数

$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{2.3}$$

可以得到一个由均匀的PDF(概率密度函数)表征的随机变量s。对于离散值，我们用概率与求和来代替概率密度函数与积分，在数字图像中灰度级r_k的出现概率由式(2.2)给出。变换(2.3)的离散形式为

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), \quad k = 0, 1, 2, \cdots, L-1 \tag{2.4}$$

式中，L是灰度图像中可能的灰度级数(8比特图像为256级)。使用(2.4)将输入图像中灰度级为r_k的每个像素映射为输出图像中灰度级为s_k的对应像素，就得到了处理后的输出图像，这称为直方图均衡化。

```
// 灰度图像的直方图均衡化
void histogram_equalization(Mat &src, Mat &dst) {
    dst.create(src.size(), src.type());
    const int GRAY_SCALE = 256;
    int rows = src.rows;
    int cols = src.cols;
    // 统计输入图像的灰度级分布
    vector<int> r(GRAY_SCALE, 0);
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            ++r[src.at<uchar>(i, j)];
    // 计算每个灰度级的概率
    int size = rows * cols;
    vector<double> pdf(GRAY_SCALE, 0.0);
    for (int i = 0; i < GRAY_SCALE; ++i)
        pdf[i] = 1.0 * r[i] / size;
    // 计算累积概率
    vector<double> cdf(GRAY_SCALE, 0.0);
    for (int i = 1; i < GRAY_SCALE; ++i)
        cdf[i] = cdf[i-1] + pdf[i];
    // 计算输出灰度级
    vector<int> s(GRAY_SCALE, 0);
    for (int i = 0; i < GRAY_SCALE; ++i)
        s[i] = saturate_cast<uchar>(GRAY_SCALE * cdf[i] + 0.5);
    // 输出图像
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
```

```
            dst.at<uchar>(i, j) = s[src.at<uchar>(i, j)];
}
```

☑ 彩色图像直方图均衡处理

应用灰度图像直方图均衡化的原理，对彩色图像的每个通道进行直方图均衡化，完成彩色图像的直方图均衡化。

```
// 彩色图像的直方图均衡化
void rgb_histogram_equalization(Mat &src, Mat &dst) {
    dst.create(src.size(), src.type());
    const int N_CHANNELS = 3;
    const int GRAY_SCALE = 256;
    int row = src.rows;
    int col = src.cols;
    int size = row * col;
    vector<vector<int>>    r (N_CHANNELS, vector<int>(GRAY_SCALE, 0));
    vector<vector<int>>    s (N_CHANNELS, vector<int>(GRAY_SCALE, 0));
    vector<vector<double>> p (N_CHANNELS, vector<double>(GRAY_SCALE, 0.0));
    for (int i = 0; i < row; ++i)
        for (int j = 0; j < col; ++j)
            for (int k = 0; k < N_CHANNELS; ++k)
                ++r[k][src.at<Vec3b>(i, j)[k]];
    for (int i = 0; i < N_CHANNELS; ++i)
        for (int j = 0; j < GRAY_SCALE; ++j)
            p[i][j] = 1.0 * r[i][j] / size;
    vector<vector<double>> prefix_sum (N_CHANNELS, vector<double>
(GRAY_SCALE, 0.0));
    prefix_sum[0][0] = p[0][0];
    prefix_sum[1][0] = p[1][0];
    prefix_sum[2][0] = p[2][0];
    for (int i = 0; i < N_CHANNELS; ++i)
        for (int j = 1; j < GRAY_SCALE; ++j)
            prefix_sum[i][j] = prefix_sum[i][j-1] + p[i][j];
    for (int i = 0; i < N_CHANNELS; ++i)
        for (int j = 0; j < GRAY_SCALE; ++j)
            s[i][j] = saturate_cast<uchar>(GRAY_SCALE * prefix_sum[i][j] +
0.5);
    for (int i = 0; i < row; ++i)
        for (int j = 0; j < col; j++)
            for (int k = 0; k < N_CHANNELS; ++k)
                dst.at<Vec3b>(i, j)[k] = s[k][src.at<Vec3b>(i, j)[k]];
}
```

实验结果

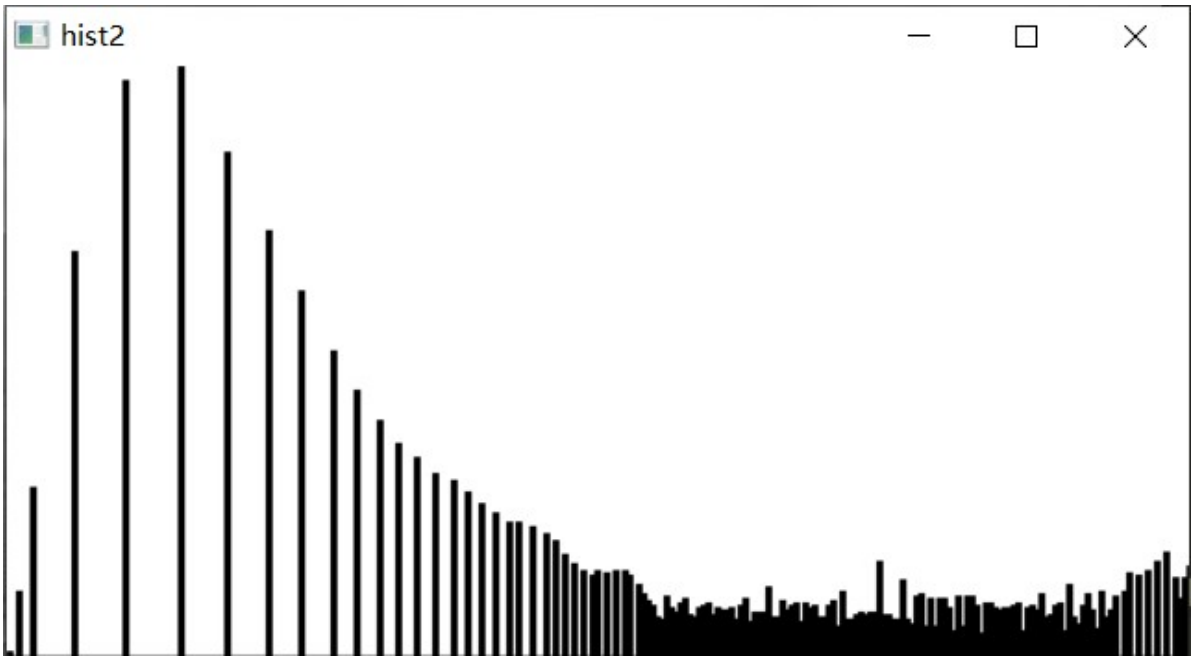
> 1. 计算灰度图像的归一化直方图

> 所用的灰度图像

> 该灰度图像的归一化直方图

> 2. 灰度图像直方图均衡处理

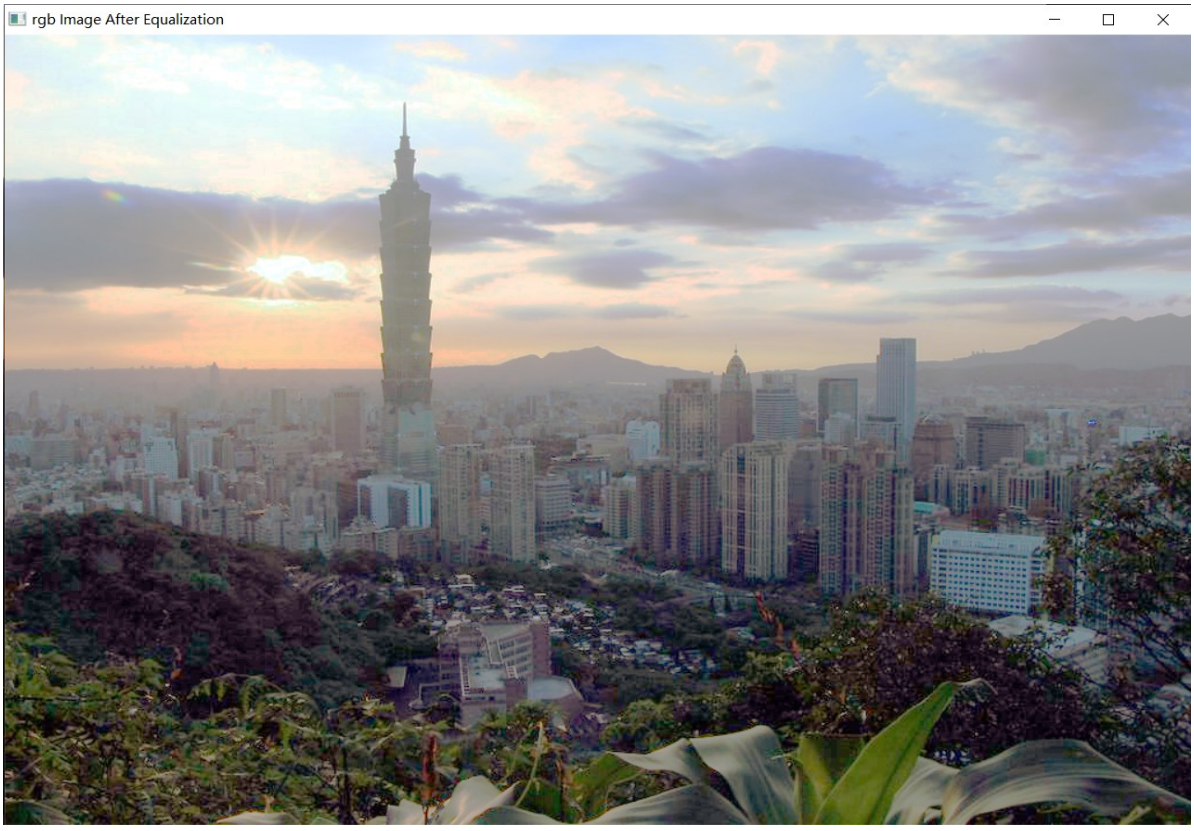
> 处理后的图像的归一化直方图



> 3. 彩色图像的直方图均衡处理

> 处理前图像

> 处理后图像



完整的源代码见附件`lab2.cpp`。