

# 计算机视觉附加题 1

陈铭涛

16340024

## 1. 用 CImg 编写灰度图像直方图均衡化

图像直方图是图像灰度级的函数，表示了图像中各灰度级出现的频率。通过对图像直方图均衡化的操作，可以增强图像的全局对比度。其主要步骤如下：

1. 从输入图像中求出各灰度级出现的概率密度函数 pdf，即其灰度直方图
2. 根据 pdf 使用累积分布函数做变换求出 CDF
3. 使用  $h(v) = \text{int}(cdf(v) * (v_{max} - 1) + 0.5)$  求得新的灰度值，其中  $v_{max}$  表示梯度值的最大值（灰度图中一般为 256）。

对于 RGB 彩色图像，可以使用分别对 RGB 三个通道进行均衡化的方法进行图像直方图均衡化，然而这种做法可能会破坏图像的色彩平衡，导致图像出现失真或其他现象，这是由于颜色通道间的比例分布改变了。改良的方法是使用另外的色彩空间进行均衡化。在这次的程序中实现了使用 YCbCr 色彩空间进行彩色图像均衡化，因为使用 YCbCr 时只对 Y 通道（流明）进行均衡化，不会对图像的色彩平衡造成过多的影响，输出的图片效果可以优于对 RGB 三通道分别进行均衡化的效果。

## 程序简述

程序中包含了一个 HistoEqual 类，用于对目标图片进行处理，构造函数接收文件名并读取目标文件，类中各函数及用处如下：

```
CImg<unsigned char> runWithGreyScale(int nb_level);
```

对输入图像的灰度图进行均衡化，返回均衡化后的图像

```
CImg<unsigned char> runWithRGB(int nb_level);
```

对输入图像的 RGB 三通道分别进行均衡化，返回均衡化后的图像

```
CImg<unsigned char> runWithYCbCr(int nb_level);
```

将输入图像转为 YCbCr 色彩空间后进行均衡化再转换为 RGB 图像，返回处理后的图像

```
CImg<unsigned char> getRGBHistogram();
```

返回输入 RGB 图像的直方图

```
CImg<unsigned char> getGreyHistogram();
```

返回输入图像灰度图的直方图

```
static void equalizeImageChannel(CImg<unsigned char>& input, int channel,
int nb_level);
```

对传入图像参数的对应通道进行均衡化，nb\_level 参数为灰度级的数量，一般为 256，若为其他数值则每个灰度级值区间大小为  $256/nb\_level$ 。

```
static vector<float> getChannelCDF(const CImg<unsigned char>& img, int
channel);
```

对传入图像的对应通道求 CDF，返回一个长度为 256 的 vector。

## 程序测试

main.cpp 文件包含了用于测试的主程序，调用 HistoEqual 类根据输入参数进行图像的处理。使用方法如下：

```
./HistogramEqualization -i filename [-l|--level histogram level] [-rgb] [-ycbcr]
```

其中各个参数使用如下：

- -i 为必须指定的参数，后接输入的文件路径
- -l 或 --level 为梯度级的数量，若未指定时默认为 256
- -rgb 和 -ycbcr 用于让程序使用 rgb 模式或 YCbCr 模式进行处理，两者

可同时指定，若都未指定时则程序将对灰度图进行处理。

程序完成图像处理时，若为灰度图模式则将会显示处理前后的图像直方图（使用 CImg 的 get\_histogram 实现）。生成输出图像后程序会将原图与生成的所有图像并列显示用于对比，并将生成的图像对应地存入 greyEqualized.bmp, YCbCrEqualized.bmp 或 RGBEqualized.bmp 文件中。

## 程序编译

程序通过 CMake 进行编译，在 Linux, macOS 和 Windows 下的编译方式分别如下：

```
# macOS, GNU/Linux
cmake .
make

# Windows
cmake -G "MinGW Makefiles" .
mingw32-make
```

在代码目录下的 bin 文件夹中已经包含了 Linux, macOS 和 Windows 下的可执行文件。

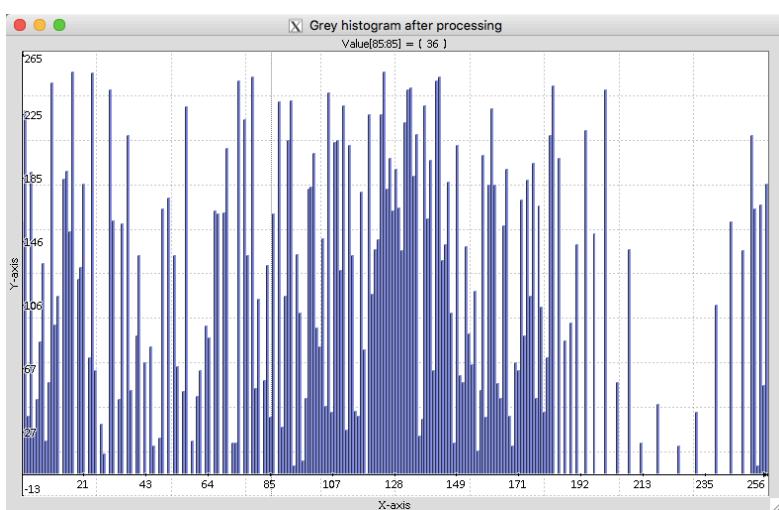
## 程序运行结果

本次使用了不同光照条件下的数张图片进行测试。程序运行目录为代码目录下的 testcases 文件夹，其在灰度图和彩色模式下的程序输出分别如下：

1. 处理前：



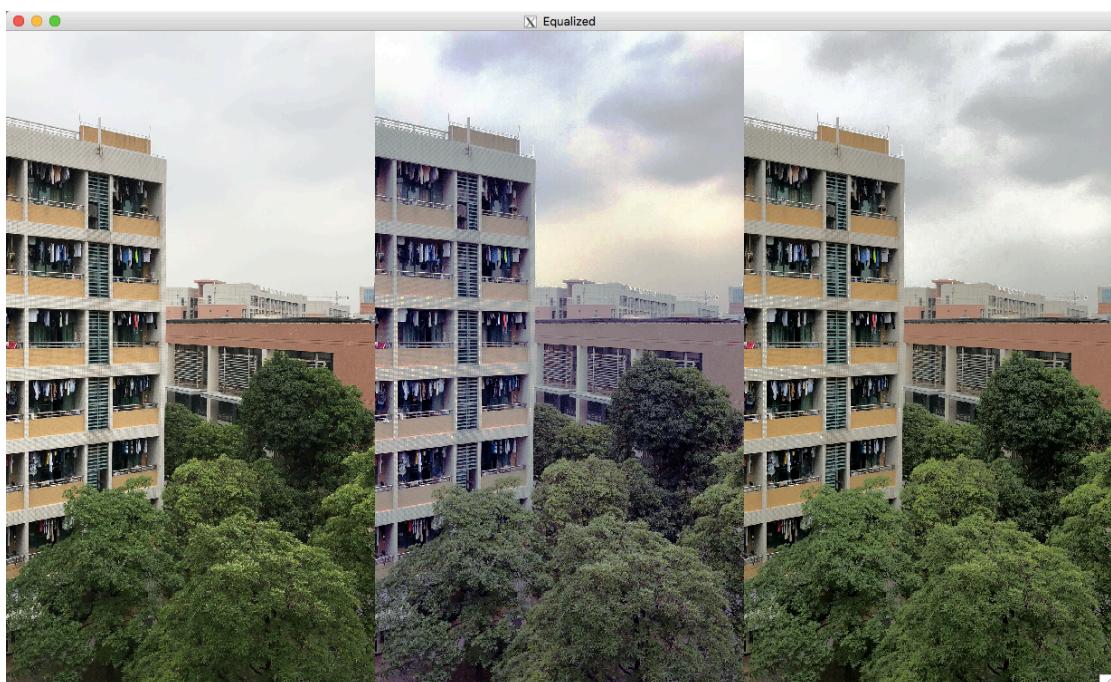
处理后：



灰度图对比：

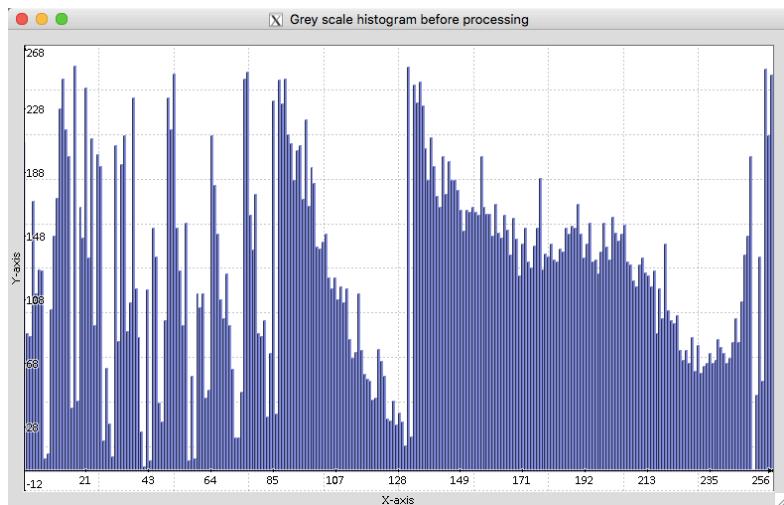


彩色图对比：

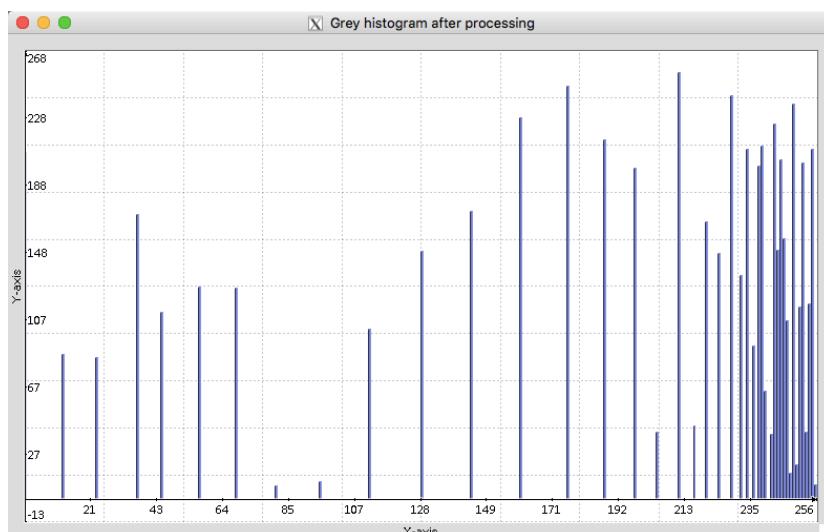


左为原图，中为 RGB 模式输出，右为 YCbCr 模式输出。可以看出 RGB 方  
法进行均衡化处理较之 YCbCr 方法会有颜色失真的情况出现。

## 2. 处理前：



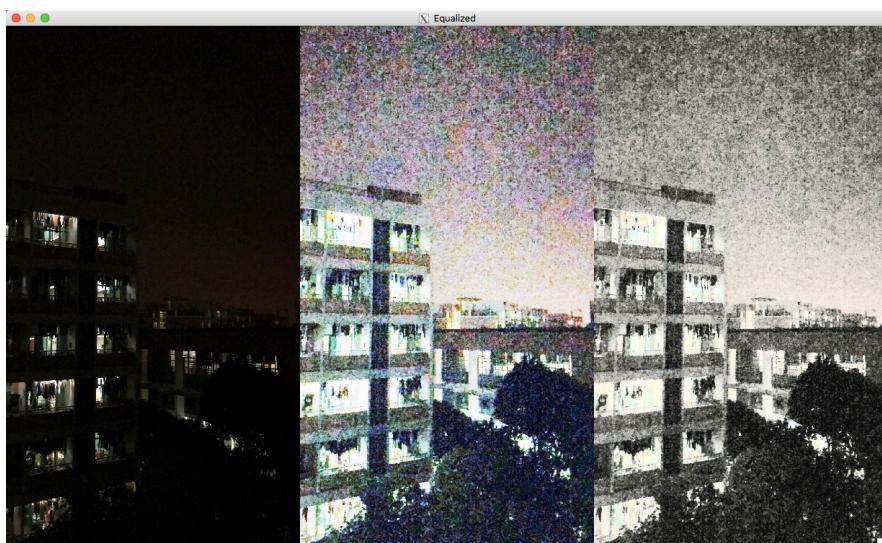
处理后：



灰度图：



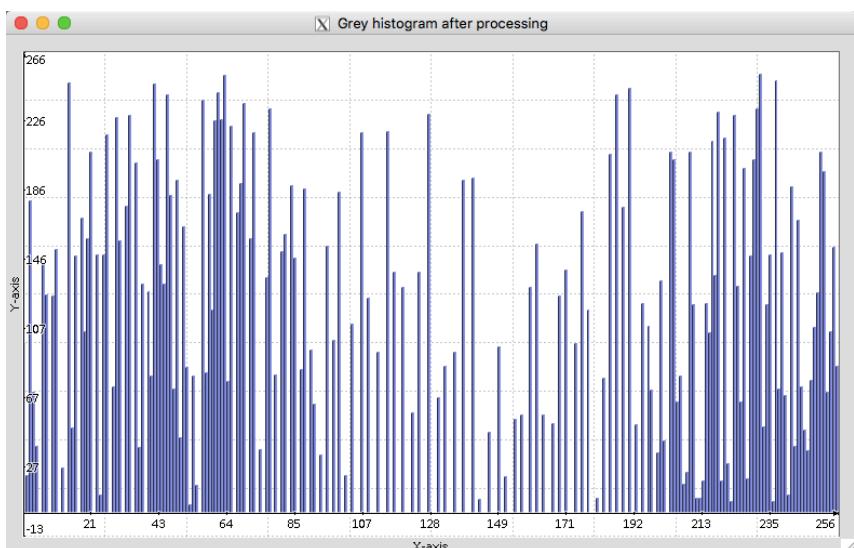
彩色图：



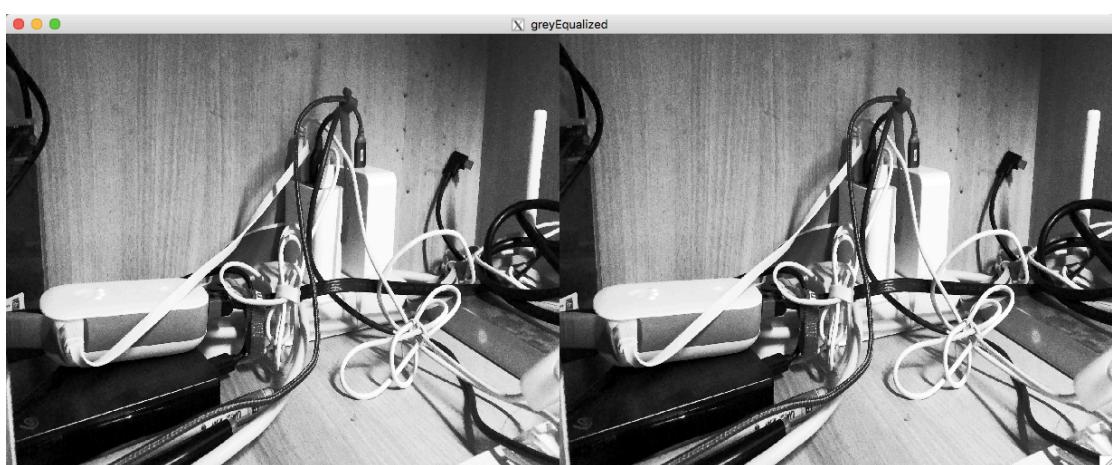
3. 处理前：



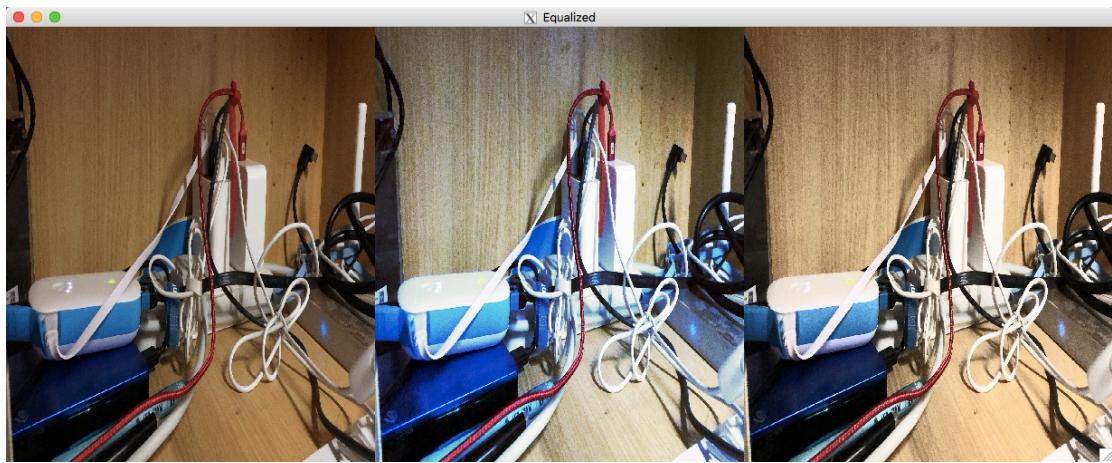
处理后：



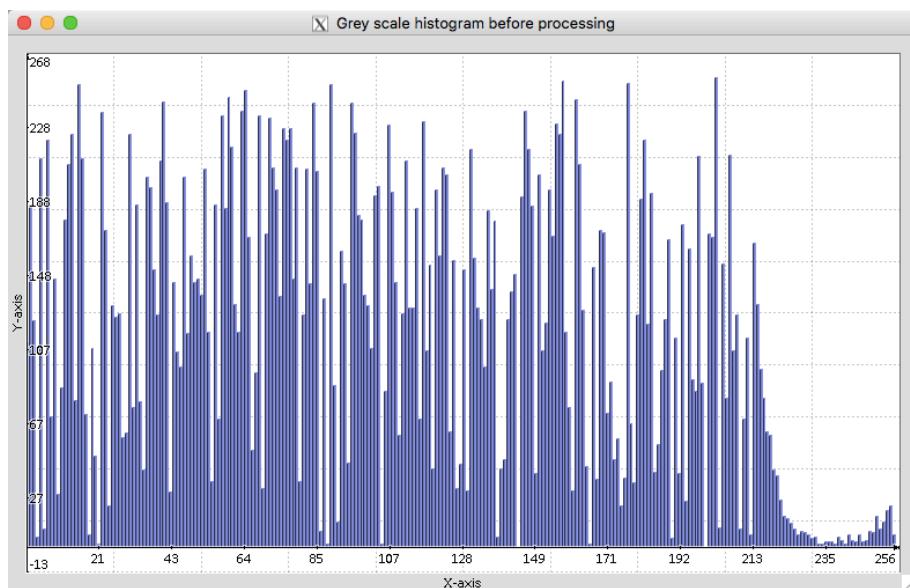
灰度图：



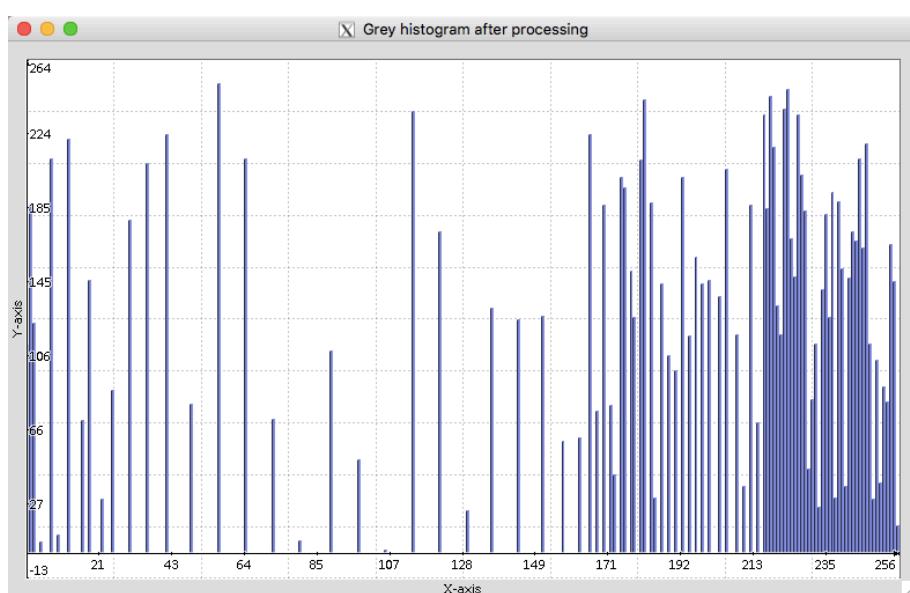
彩色图：



4. 处理前：



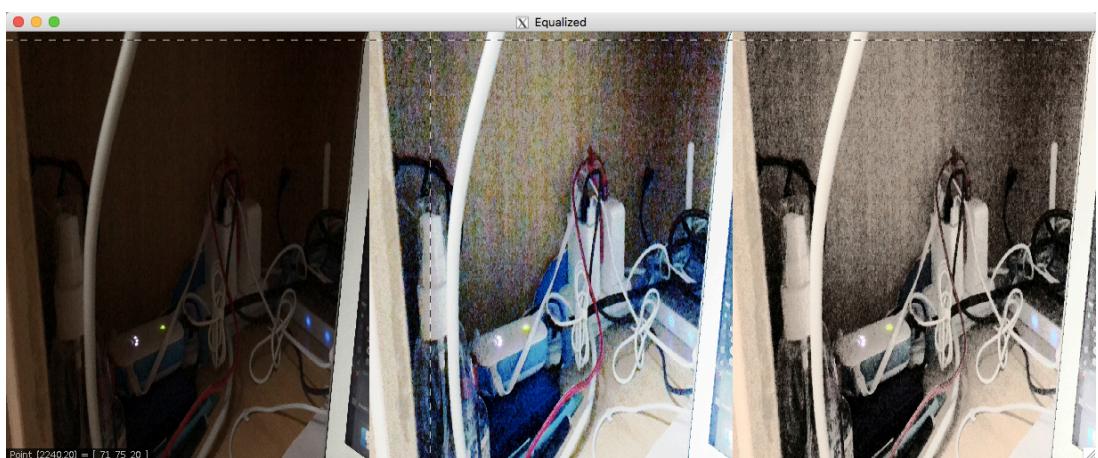
处理后：



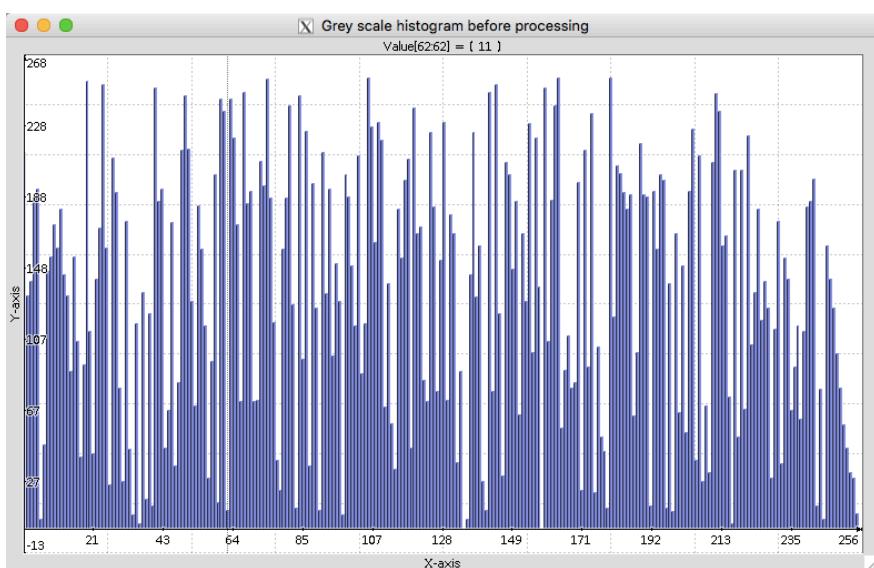
灰度图：



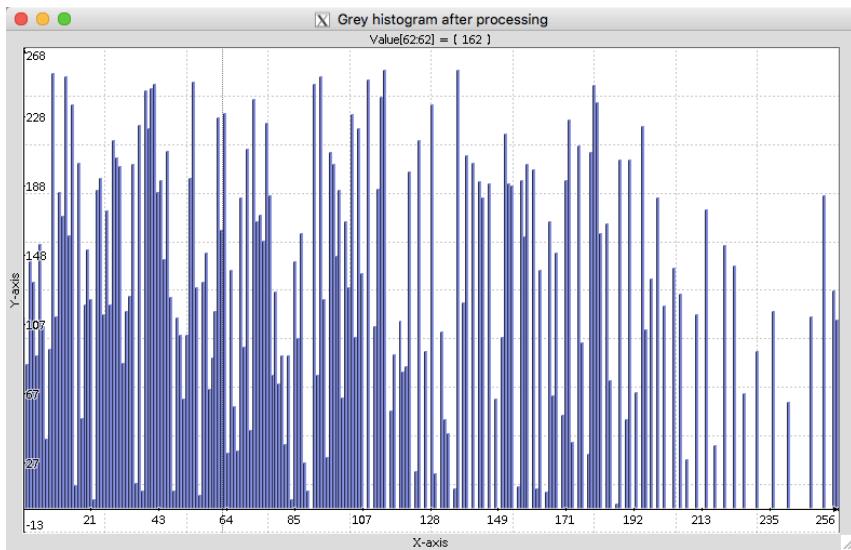
彩色图：



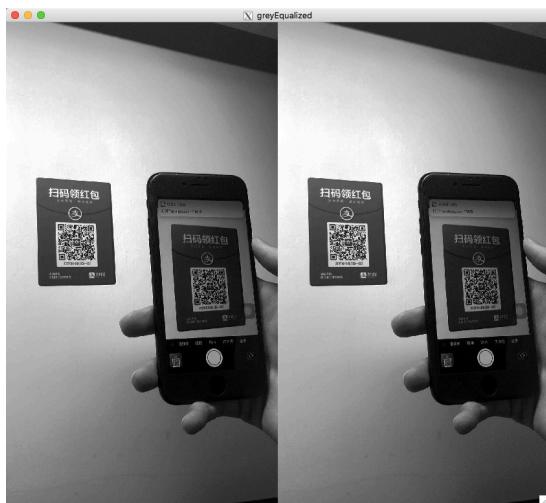
## 5. 处理前：



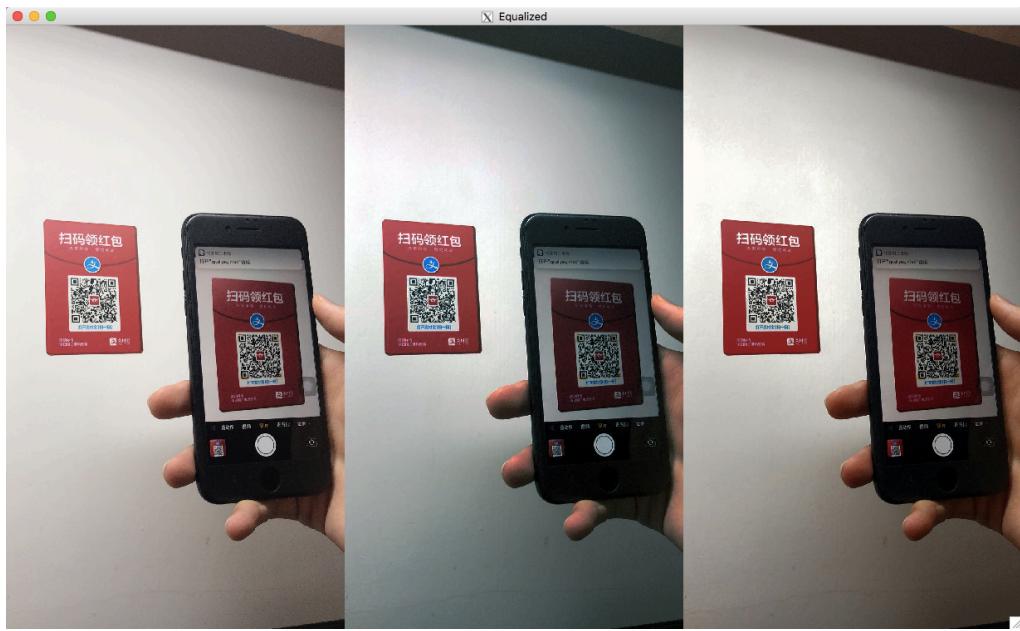
处理后：



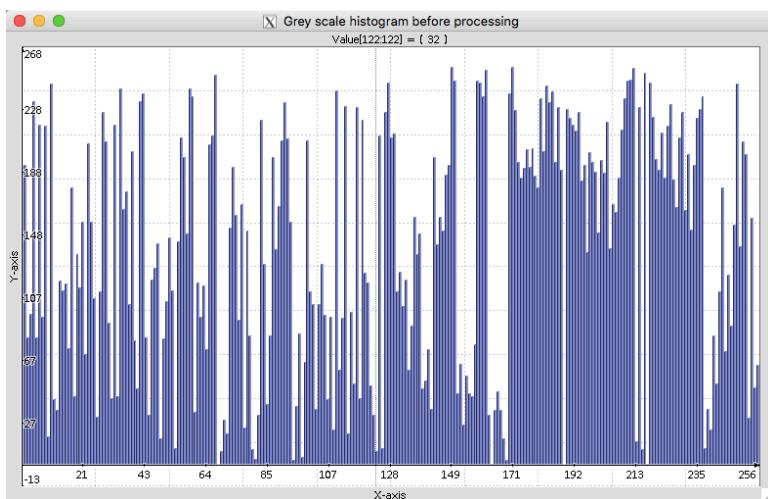
灰度图：



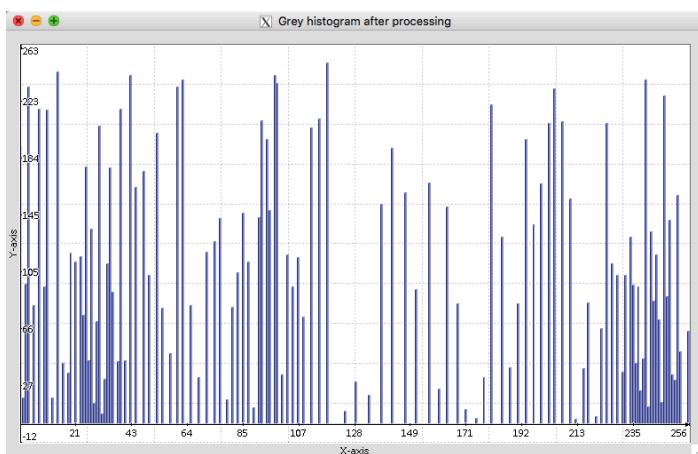
彩色图：



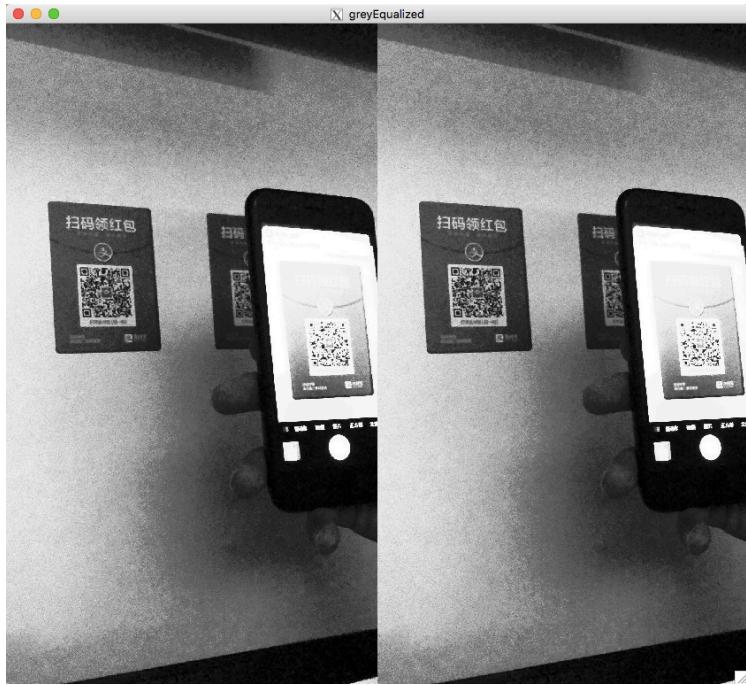
## 6. 处理前：



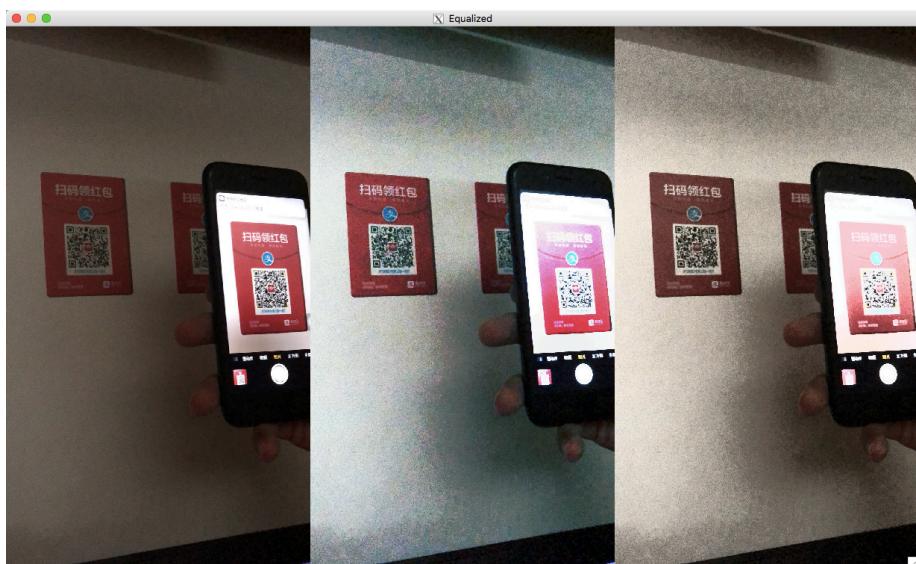
处理后：



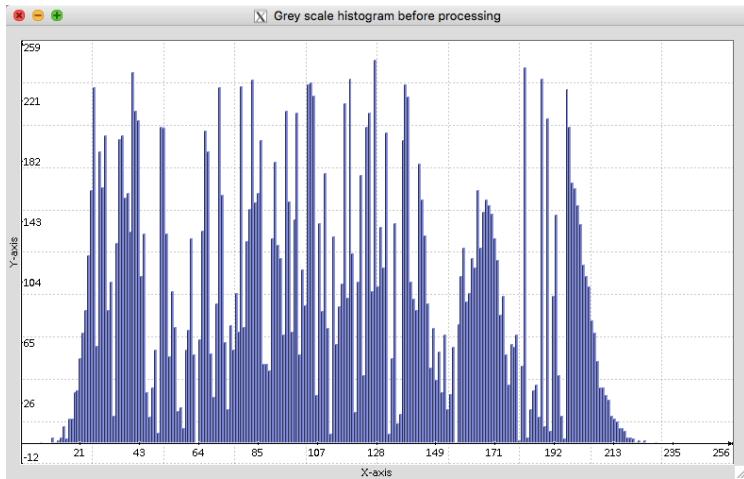
灰度图：



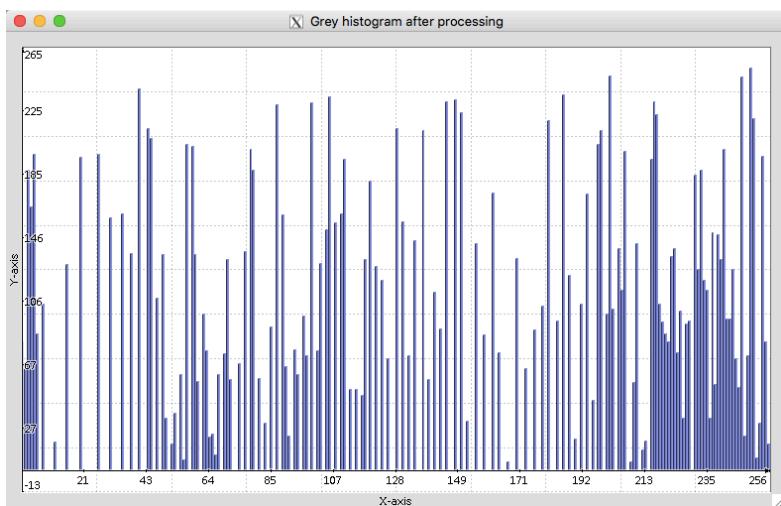
彩色图：



7. 处理前：



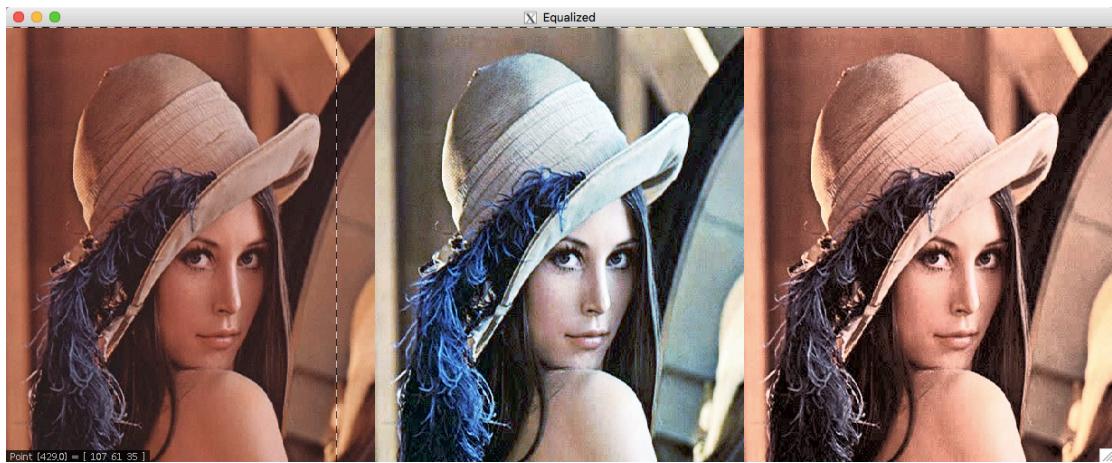
处理后：



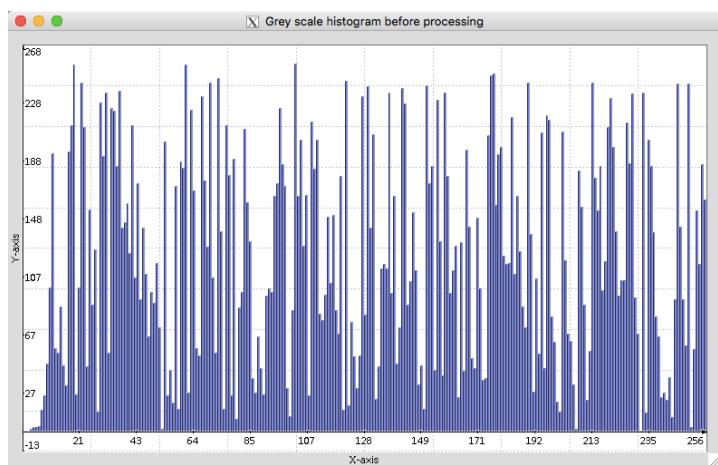
灰度图：



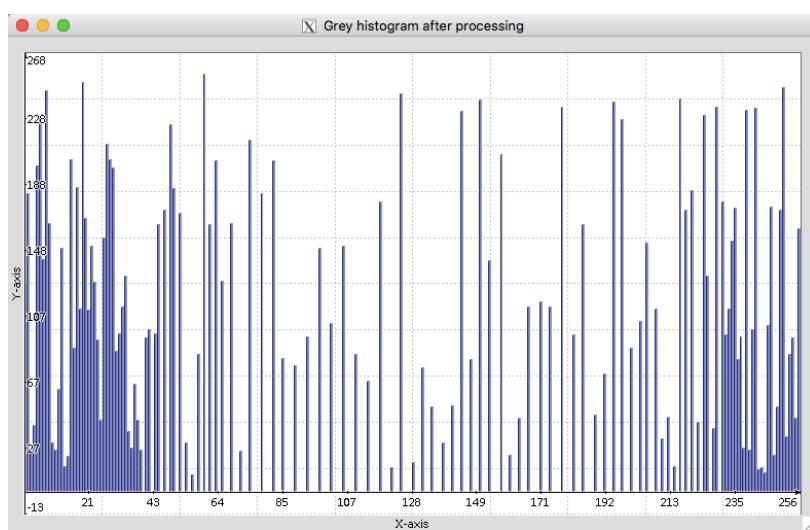
彩色图：



8. 处理前：



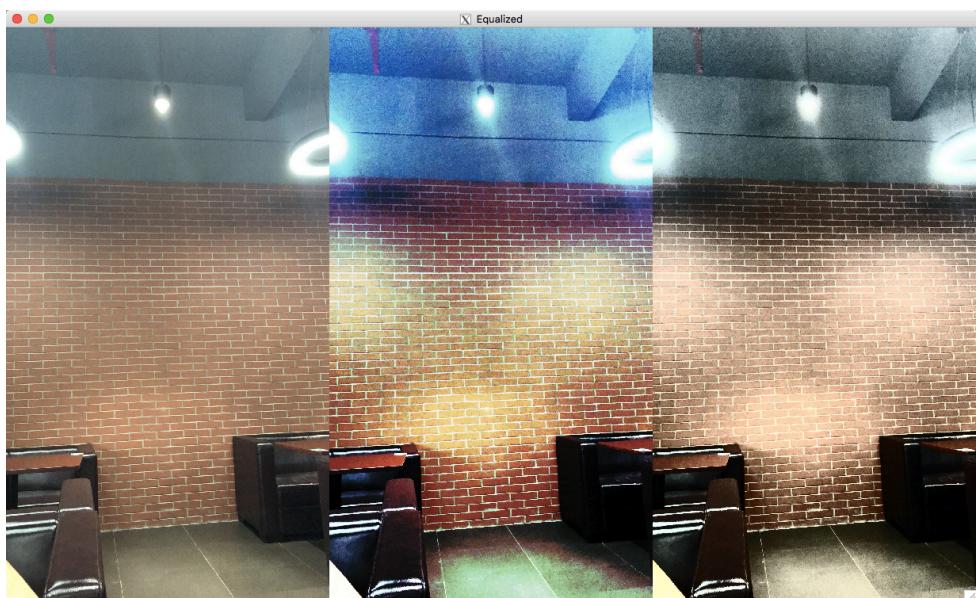
处理后：



灰度图：



彩色图：



## 2. 实现颜色转换

颜色转换的意思为将原图的色彩特征转为参考图的色彩特征。由于 RGB 色彩空间三个通道有较强的相关性，对像素色彩的修改将

较为困难，因此在这里使用了  $l\alpha\beta$  色彩空间，先将 RGB 图像转为 lab 空间的图像，进行处理后再转回 RGB 的图像。

RGB 向 Lab 转换的方式如下：

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3811 & 0.5783 & 0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1288 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$L = \log L, M = \log M, S = \log S$$

$$\begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}$$

Lab 向 RGB 转换的方式如下：

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix}$$

$$L = 10^L, M = 10^M, S = 10^S$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4679 & -3.5873 & 0.1193 \\ -1.2186 & 2.3809 & -0.1624 \\ 0.0497 & -0.2439 & 1.2045 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}$$

计算出原图和参考图的 lab 空间后，对各通道求出均值和标准差，然后对原图的点减去均值：

$$l' = l - \langle l \rangle$$

$$\alpha' = \alpha - \langle \alpha \rangle$$

$$\beta' = \beta - \langle \beta \rangle$$

然后根据标准差比例对点值进行调整：

$$l' = \frac{\sigma_t^l}{\sigma_s^l} l'$$

$$\alpha' = \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha'$$

$$\beta' = \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta'$$

然后，将所求得的结果加上参考图的均值：

$$l' = l' - \langle l_s \rangle$$

$$\alpha' = \alpha' - \langle \alpha_s \rangle$$

$$\beta' = \beta' - \langle \beta_s \rangle$$

最后将求得的 lab 转回 RGB 空间即为所得。

## 程序简述

提交的程序包含了一个主要的 ColorTransfer 类，其构造函数接收原图和参考

图的文件路径并读取图像，其他函数及用途如下：

```
static CImg<double> RGBtoLab(const CImg<unsigned char>& img);
```

将输入的 RGB 图像转为 Lab 空间的图像，返回转换后的图像

```
static CImg<unsigned char> LabtoRGB(const CImg<double>& img);
```

将输入的 Lab 图像转为 RGB 空间的图像，返回转换后的图像

```
static array<double, 3> getMean(const CImg<double>& img);
```

对输入的图像求均值，返回的数组为图像三个通道的均值

```
static array<double, 3> getStdDev(const CImg<double>& img, const
array<double, 3>& mean);
```

对输入的图像求标准差，需输入一个均值的数组。

```
CImg<unsigned char> run();
```

执行颜色转换的主要流程，返回处理完后的图像。

## 程序测试

测试的主程序位于 main.cpp, 用法如下：

```
./ColorTransfer -s filename -t filename
```

其中参数-s 后接原图文件路径, -t 后接参考图文件路径。

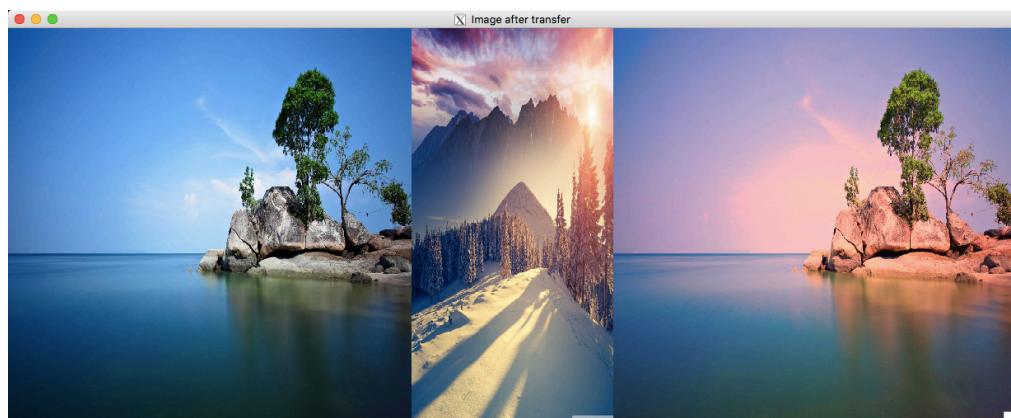
程序运行将会把输出的图片结果保存在 transfer.bmp 文件中, 并将原图, 参考图与输出图并列输出。

程序编译方式与第一题编译方式相同。

## 测试结果

testcases 文件夹中有 10 张用于测试的图片, 每两张为一组, 测试结果如下：

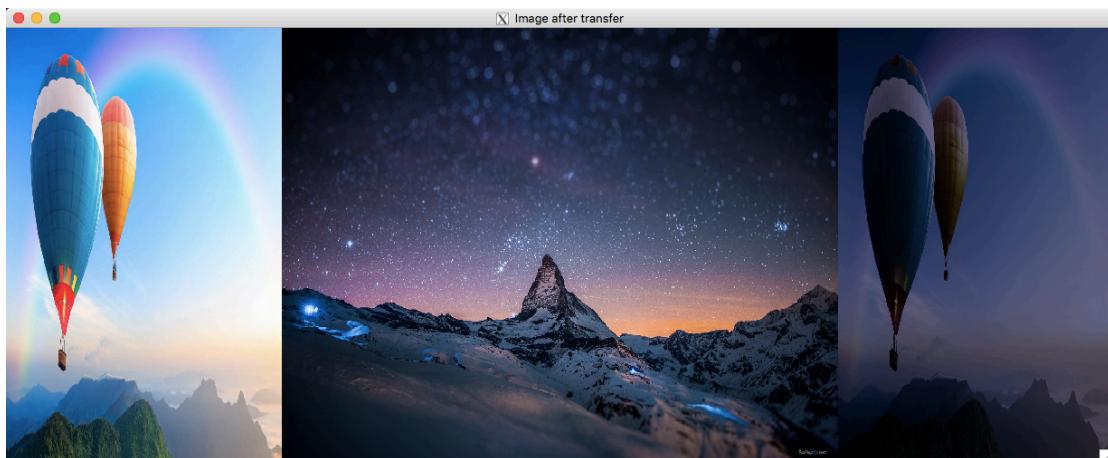
1. 原图 : 1, 参考 : 2.



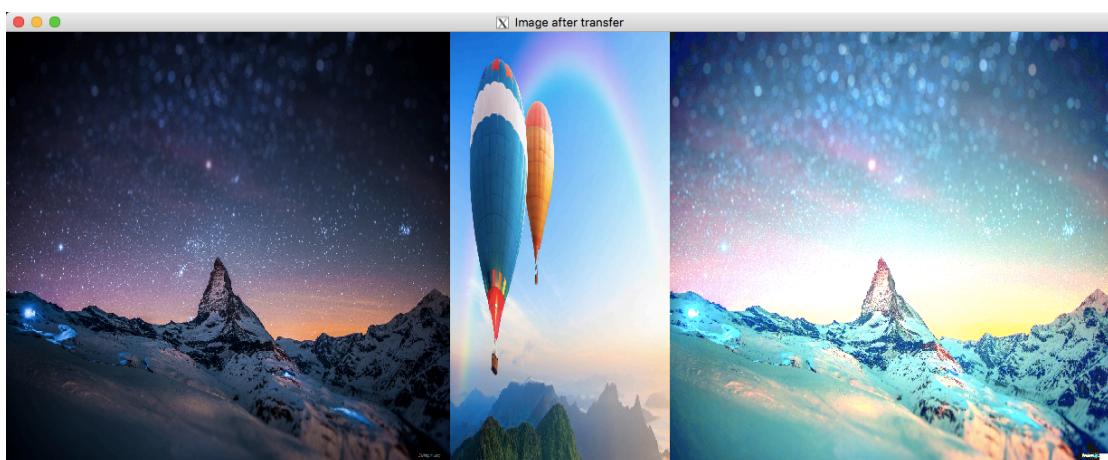
原图 : 2, 参考 : 1



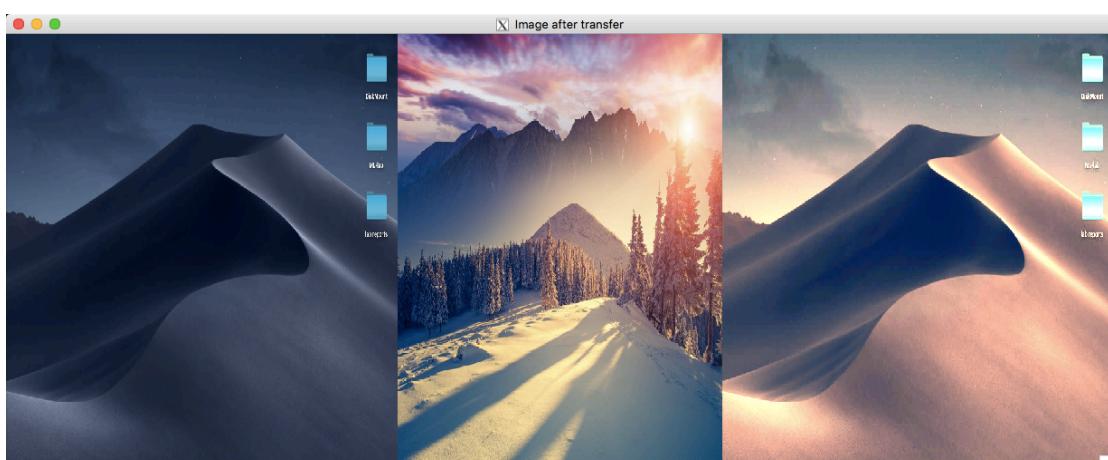
2. 原图 : 3, 参考 : 4



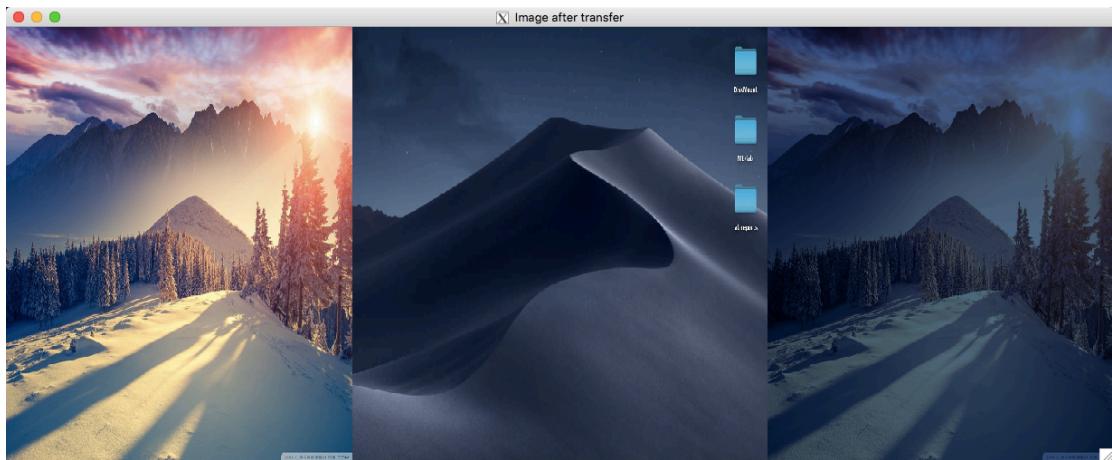
原图 : 4, 参考 : 3



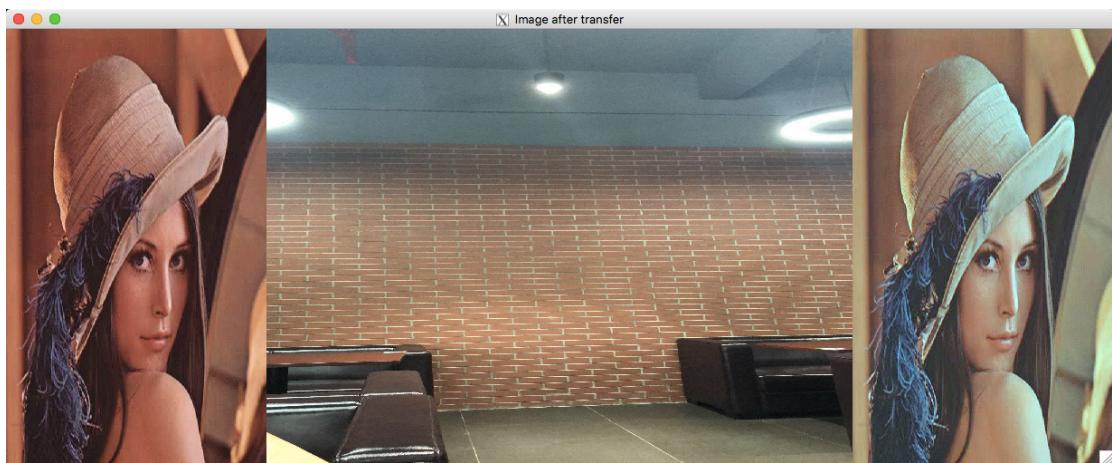
3. 原图 : 5, 参考 : 6



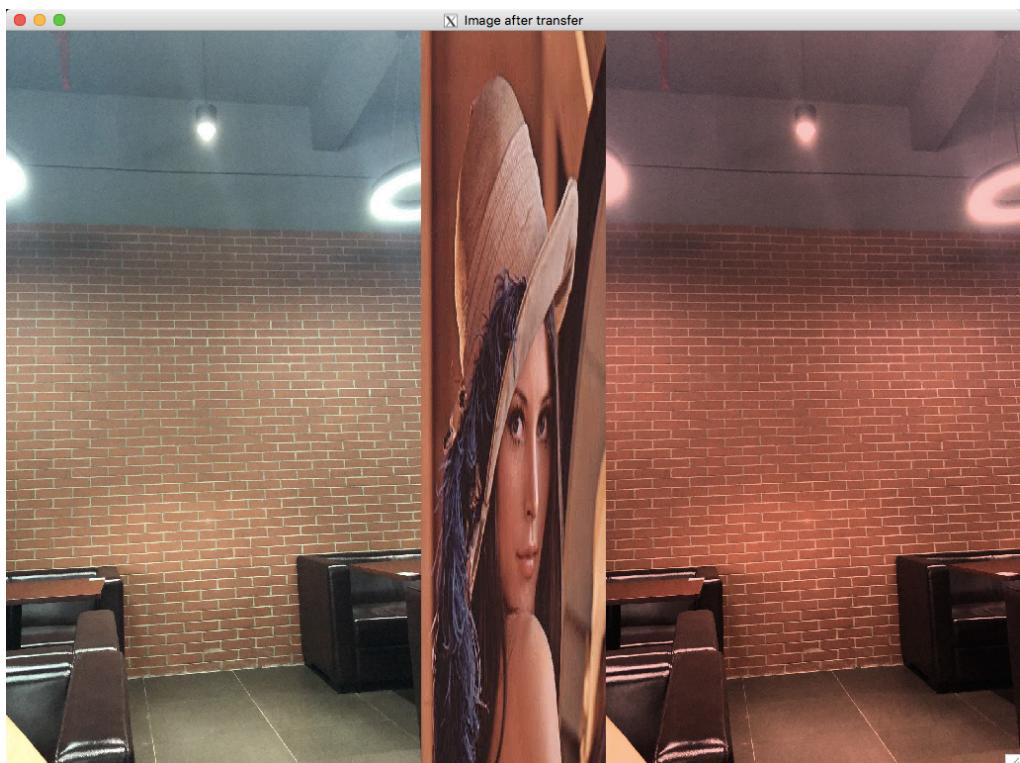
原图 : 6, 参考 5



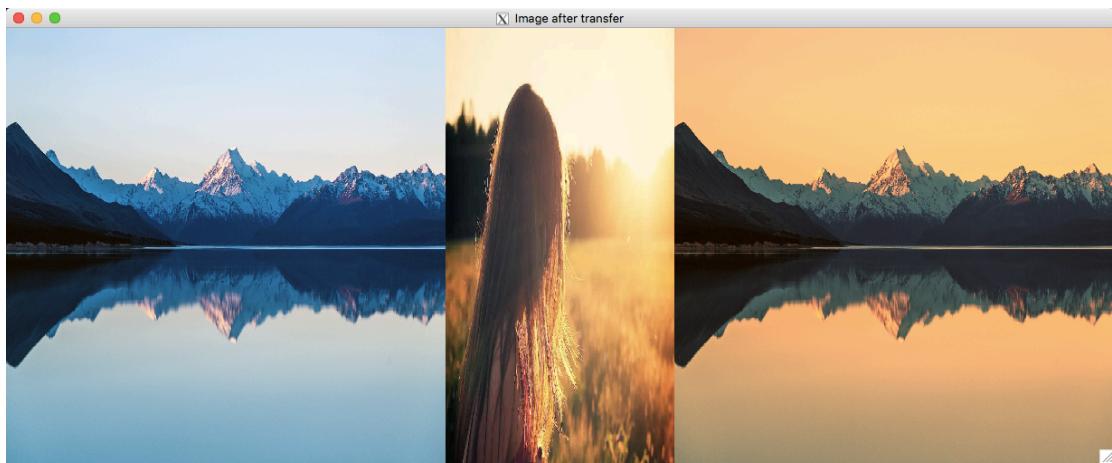
4. 原图 : 7, 参考 : 8



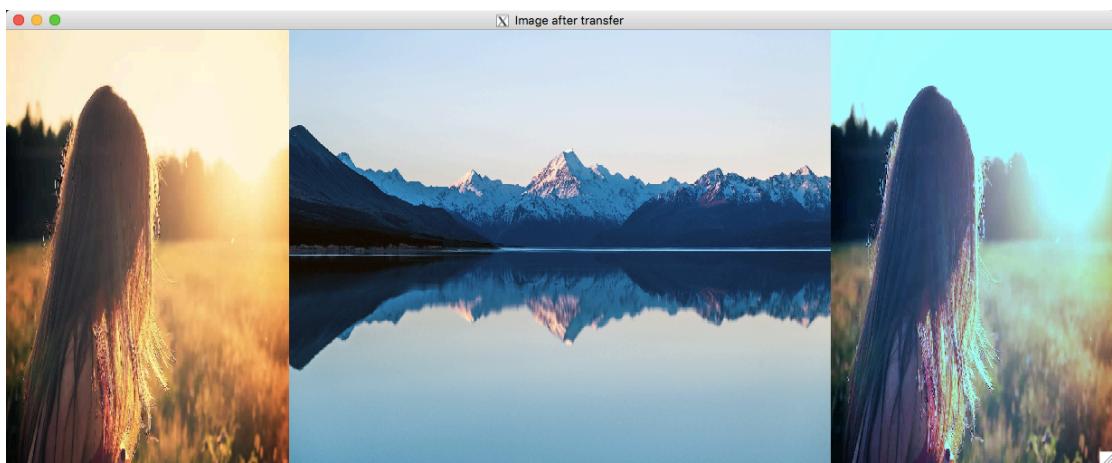
原图 : 8, 参考 : 7



5. 原图 : 9, 参考 : 10



原图 : 10, 参考 : 9



从测试中可以看出，当原图与参考图的构成区别较大时，输出的结果效果可能较差，如图 3 和 4 的处理。因此该颜色转换的方法的处理质量对输入图像构图的相似度有着一定的要求。在 *Color Transfer between Images* 一文中，提出了使用 swatches 来进行改良，比如对图像中的天空和草地分别进行选择与计算，然后对于原图像素分别进行计算并根据距离进行合并。通过这种方法可以使输出的图像结果更加合理。