实验名称：用 CImg 编写灰度图像直方图均衡化和颜色转换

实验要求：

所有的图像读写、数据处理只使用 CImg 库

实验步骤：

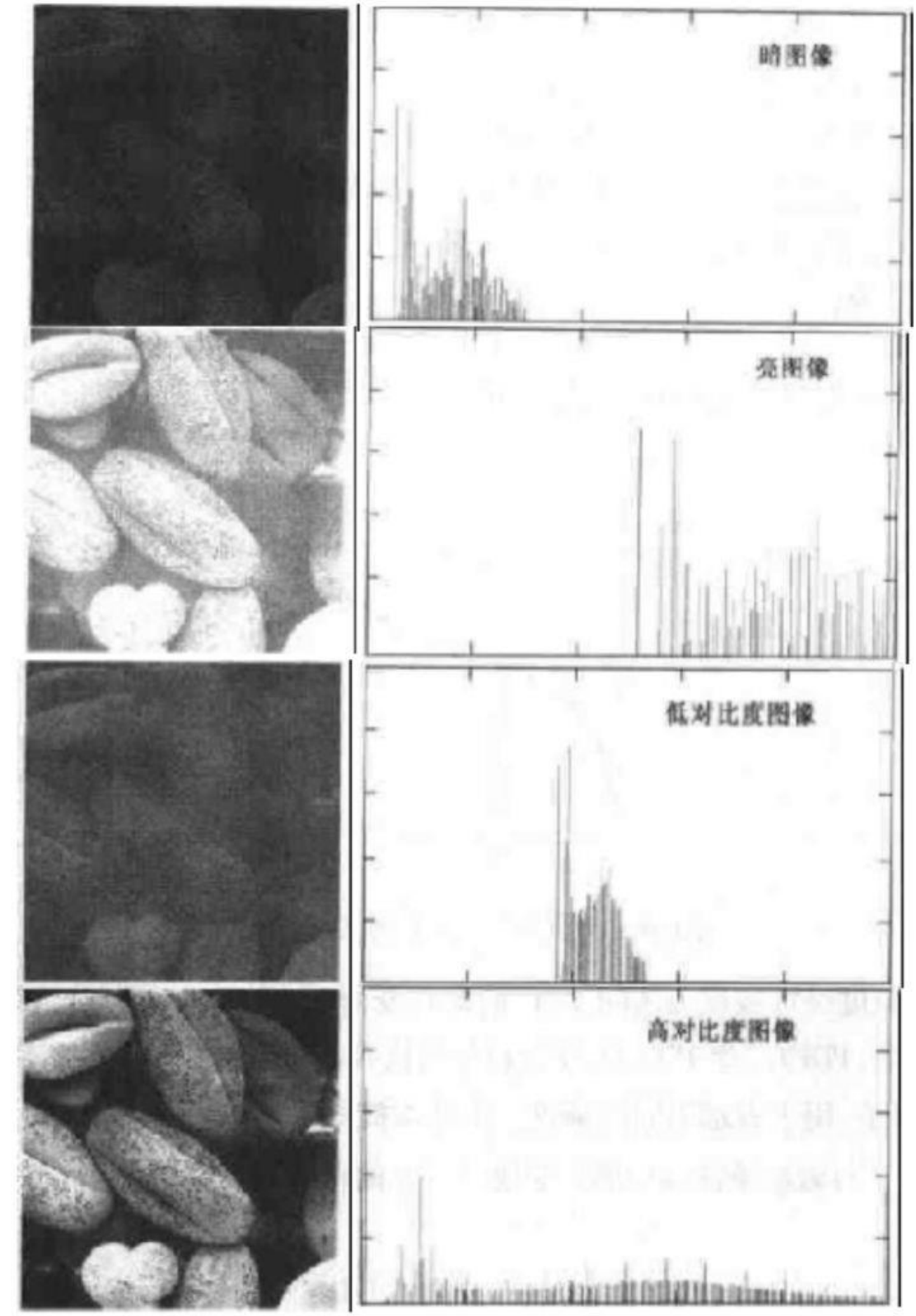
1. 用 CImg 编写图像直方图均衡化，手机拍摄不少于五张的不同光照情况下的图像，并分别针对灰度图像和彩色图像进行直方图均衡化，最后进行分析。
2. 实现颜色转换，测试图象集合不低于 5 对图像，最后进行分析。

实验过程

直方图均衡化-灰度图像

得到一张图像后，经常会发现这张图像存在各种问题，如亮度较暗或较亮，对比度较低或者较高，这都是不能让我们接受的，此时可以观察一下这张图像的直方图，对于灰度图像来说，直方图表示的是图上每个灰度级（0~255）占所有像素的比例，所以说，一幅图像只能对应一个直方图，但是一个直方图可以对应多幅图像，这也就很好的说明了直方图和图像的像素位置没有直接的关系，它只是帮助统计了在一个灰度级出现的像素个数的多少（或占比）。

如下图所示：



观察上述四幅图可以发现：

- 较暗的图中，直方图的组成集中在灰度级低的一侧
- 较亮的图中，直方图的组成集中在灰度级高的一侧
- 低对比度的图中，直方图的组成较窄，集中在一处
- 高对比度的图中，直方图的组成覆盖了很宽的范围，像素分布没有均匀

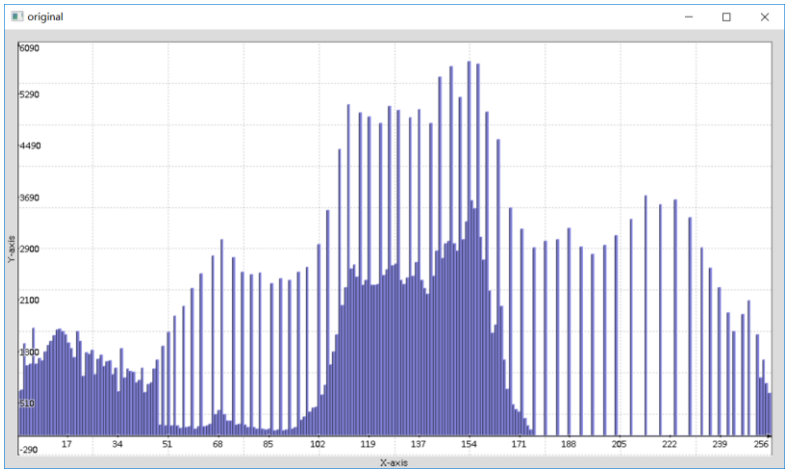
我们的目的就是使得图像的直方图尽可能覆盖多的灰度级而且分布均匀，这就是直方图均衡化的作用了。

具体做法就是统计每个灰度级对应像素的数量，再除以总像素点数得到比例（或者说概率分布），最后再将比例映射回 0~255 范围，并给原图赋值。

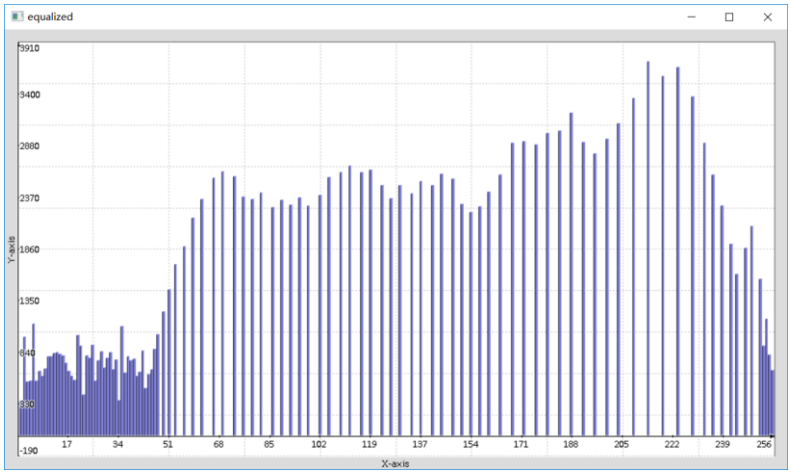
以下是图像测试（左边为灰度图像，右边为均衡化后结果）：

1.

与左边的灰度图像相比，右边的图像明显对比度增强，能够更好地观察到云层的细节。打印直方图观察，原图：

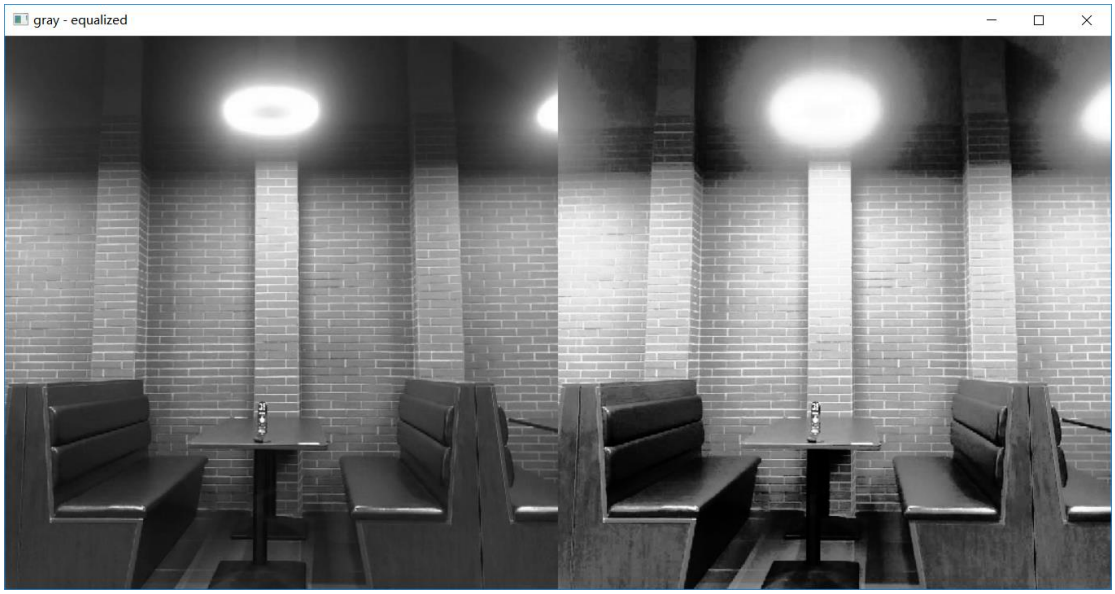


处理后：



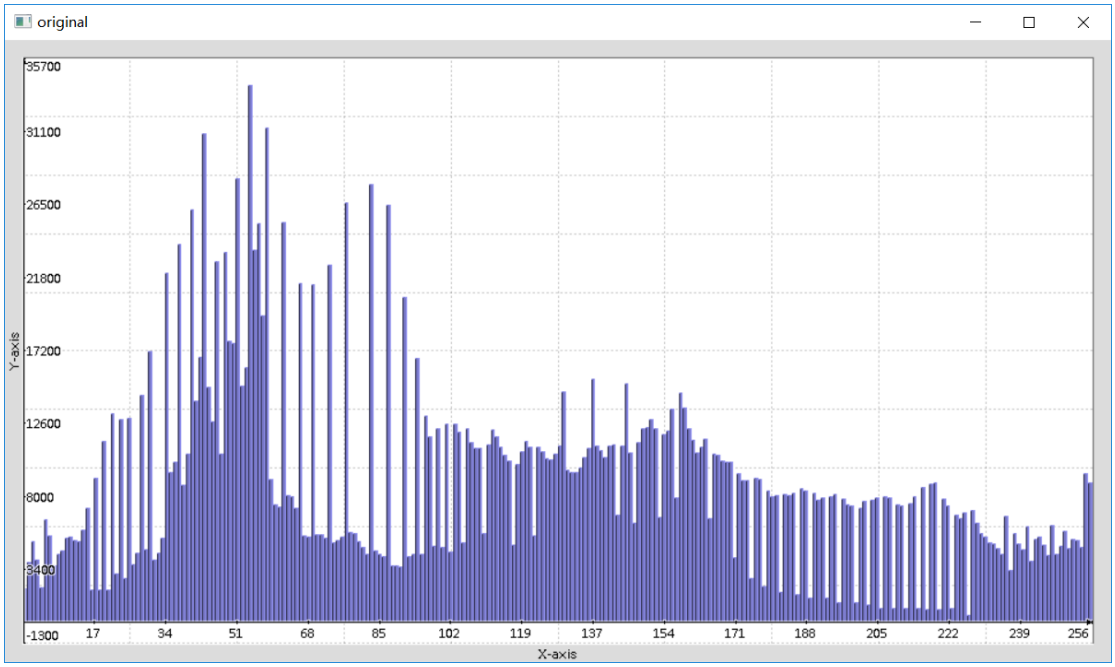
可见相比于原图来说，均衡后的图像筛除了在直方图中间密集存在的那部分灰度级，这也使得整幅图像的直方图分布更加均匀。

2.

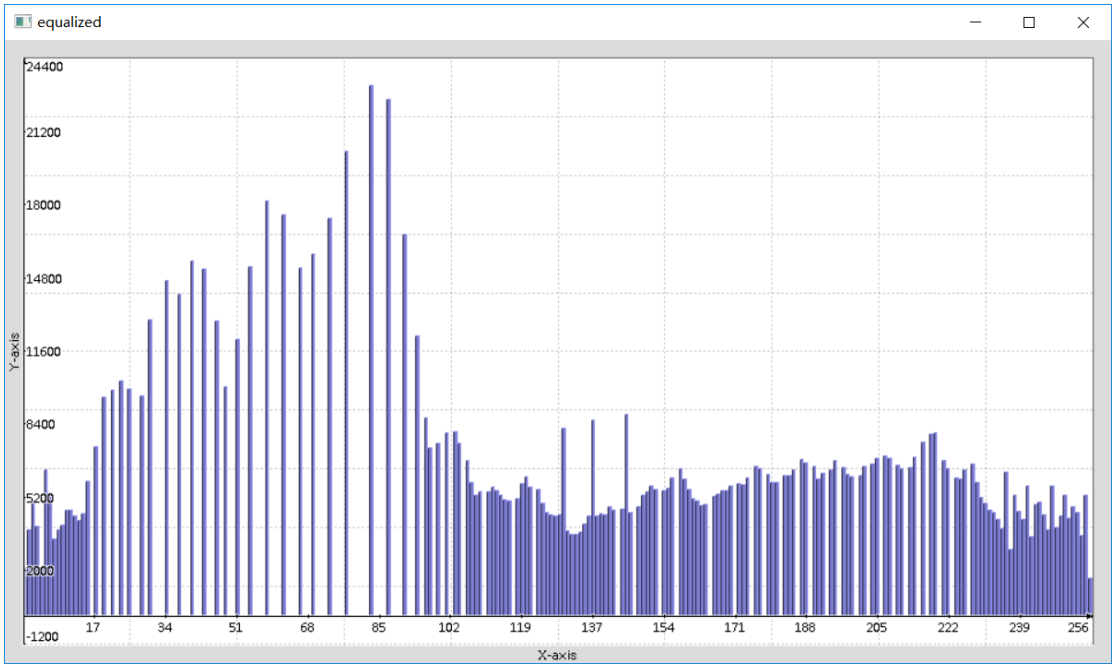


与左边的图像相比，右边的图像中，座椅以及桌子下面这些较暗处明显变亮了一些，细节的表现力更好，但是光源处光晕更重。

打印直方图观察，原图：



处理后：

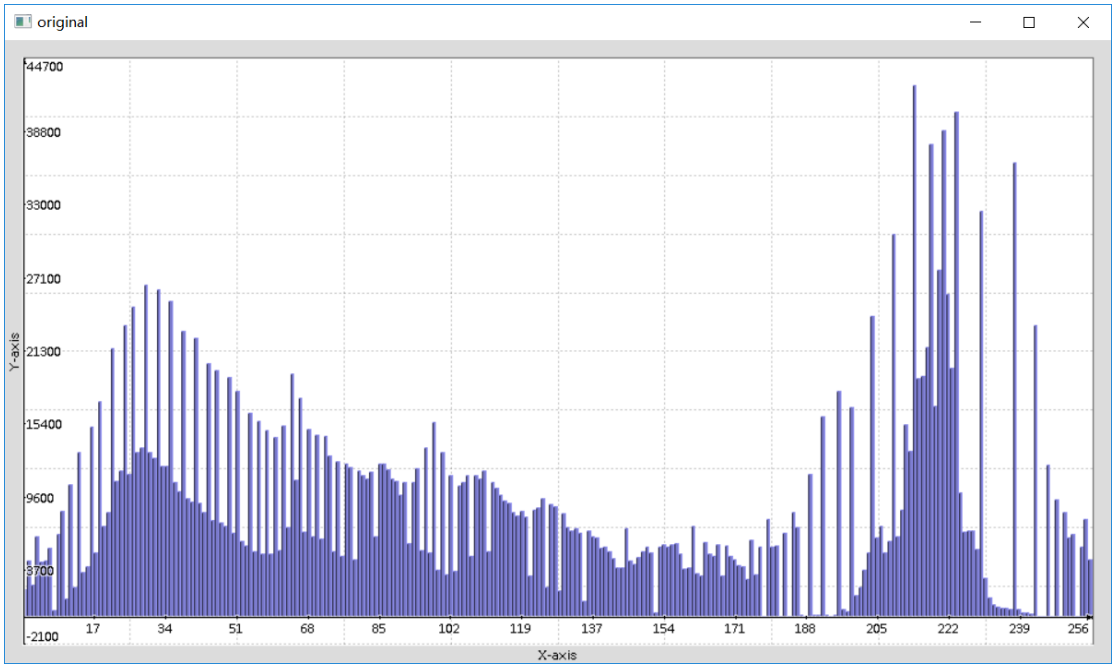


与原直方图相比，处理后的直方图中，灰度级分布也更加均匀，主要是筛除了一些分布过于密集的灰度级。

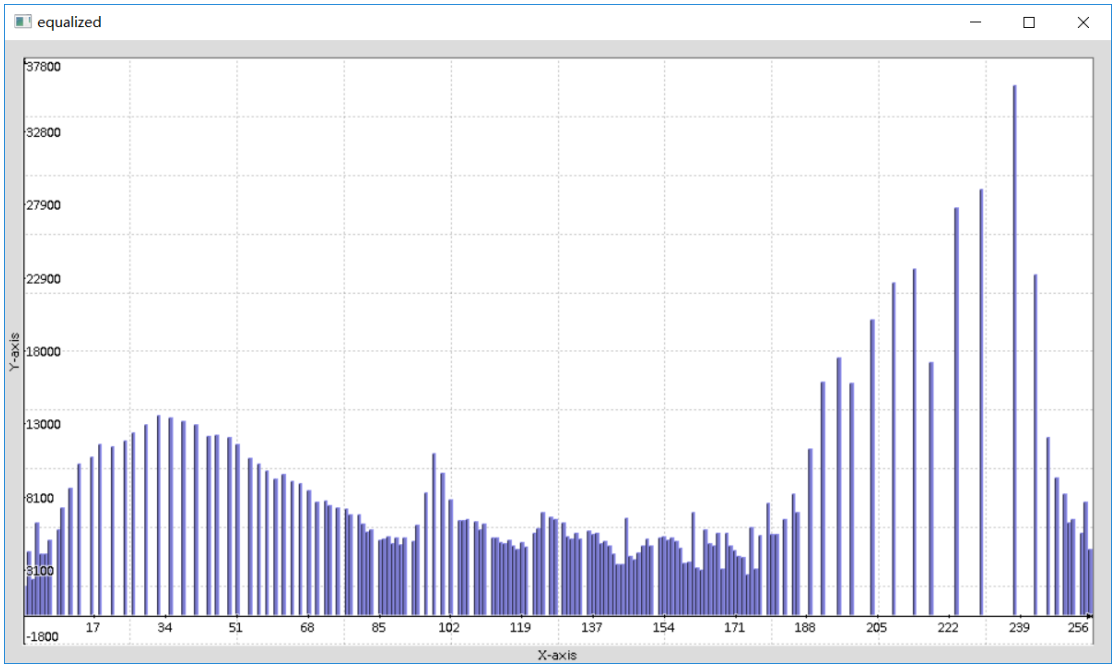
3.

与左边的图像相比，右边的图像可以更清晰的观察到建筑、树木的阴暗处，细节更加丰富。

打印直方图观察，原图：

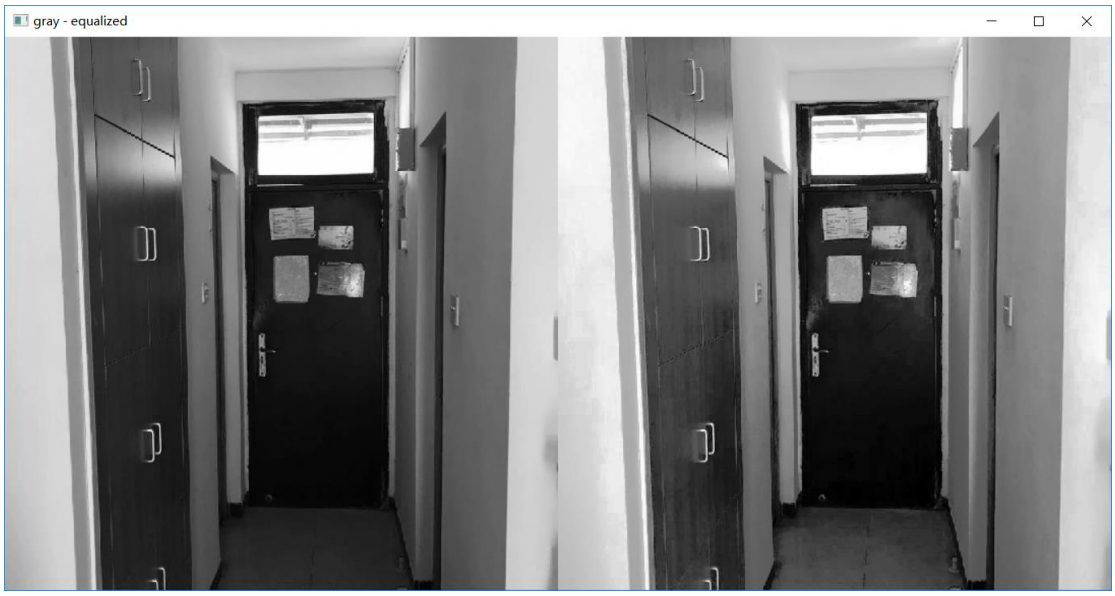


处理后:



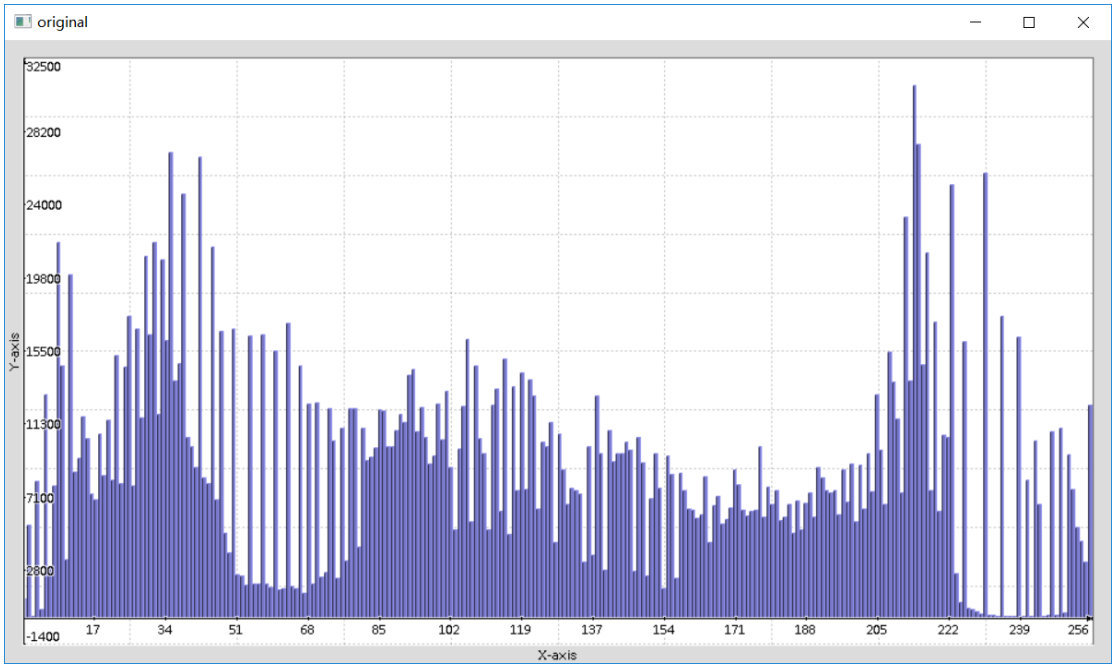
这副图像可以很清晰的看到，原直方图中在灰度级较小和较大的两个部分都比较密集，而处理后得到的直方图则没有这一问题。

4.

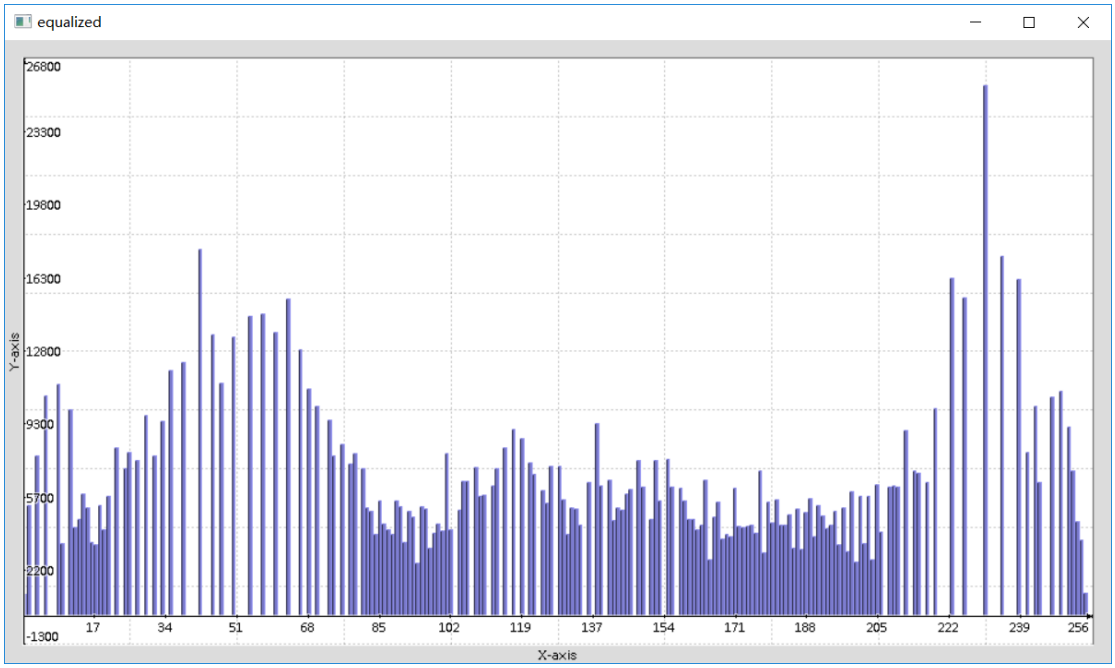


原图中，走廊的地板与柜子的下半部分基本就是一片黑，很难看见细节，而处理后可以很清晰的看见地板砖缝和柜子的细节。

打印直方图观察，原图：



处理后：

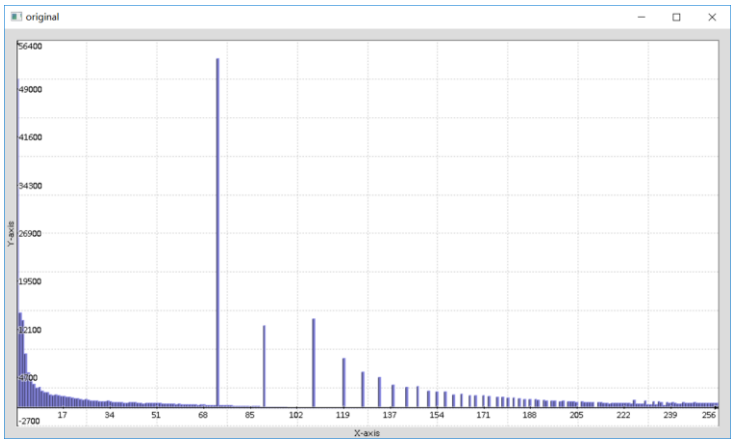


依旧是成功的将直方图变得更加均匀了。

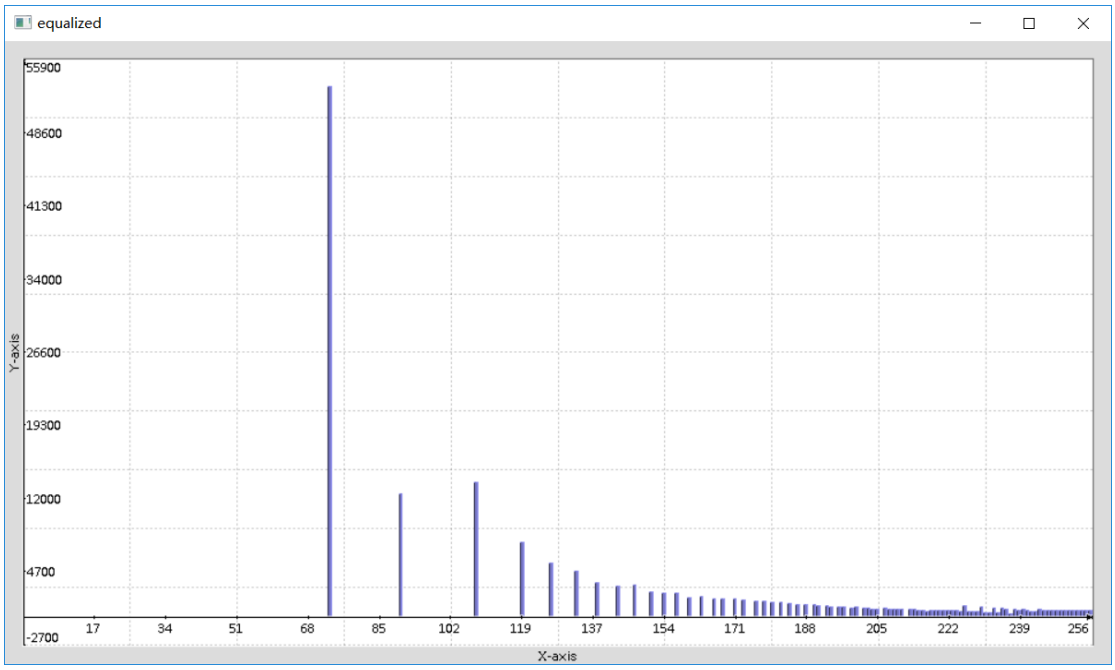
5.

这是效果最明显的一张，原图是一张非常暗的洞穴内的图片，在处理后原来根本看不见的地方也变得清晰可见了。

打印直方图观察，原图：



处理后：



可见那些密集分布在暗处的灰度级被省去了。

经过测试可见，对于灰度图做直方图均衡化能够很好的提升暗处的可见度，对于图像细节处的观察有很大的帮助，但是如果图像有一个很亮的光源时，在这个光源周围可能会产生一些扭曲或者噪音。但总体来说，直方图均衡化对于图像的优化作用是不错的。

直方图均衡化-彩色图像

对彩色图像进行直方图均衡化可以借鉴灰度图的处理——当我们对灰度图做直方图均衡化时，是对灰度图的每个灰度级进行统计、均衡，那么对于彩色图像呢？首先想到的是对 RGB 三个通道分别作一次均衡，但这样做完会得到这样的结果：

可以看到这样得到的结果虽说某些细节更加清晰了，但是图像的颜色却完全失真了，有时可能还会出现一些奇异点。这里选择先将图像从 RGB 空间转到 HSV 空间，只对 HSV 空间中的亮度进行均衡化（这就与灰度级的处理有些相似），最后再将结果转回 RGB 空间进行显示。

处理灰度图时可以根据灰度级直方图来分析处理的好坏，那么对于彩色图像如何进行分析呢？这里通过 HSV 空间的亮度直方图来分析，因为这里我们就是在对亮度进行均衡化。

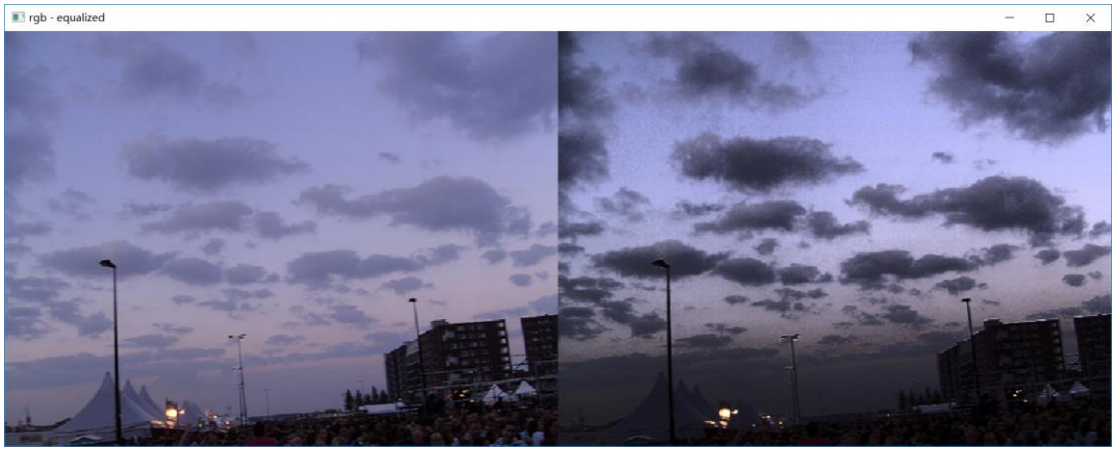
这里还有一点需要注意的就是在对灰度图进行分析时，因为灰度级是从 0~255，所以当计算完每个灰度级的分布比例后要将其映射回 0~255，而这里因为在 cimg 中 HSV 空间的亮度值是 0~1 的，所以计算完分布比例就不再需要映射了，但是为了显示方便，在累加时要先将它的值乘以 255，如：

```
//处理灰度图时
cimg_forXY(equalized, x, y){  //累加
    count[equalized(x, y)] += 1;
}
. . .
for(auto &c : count){ //映射回去
    c = 255 * c + 0.5;
}

//处理彩色图像时
cimg_forXY(equalized, x, y){  //累加
    count[equalized(x, y, 2) * 255] += 1;
}
. . .
//不需映射
```

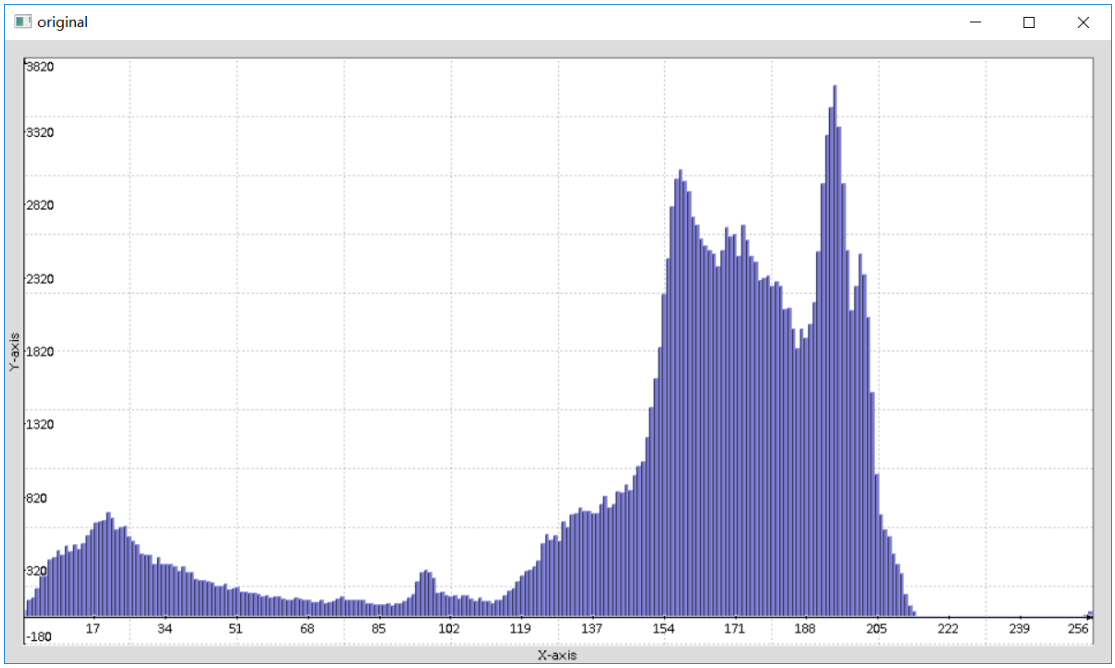
以下是图像测试，左边为原图，右边为均衡化结果：

1.

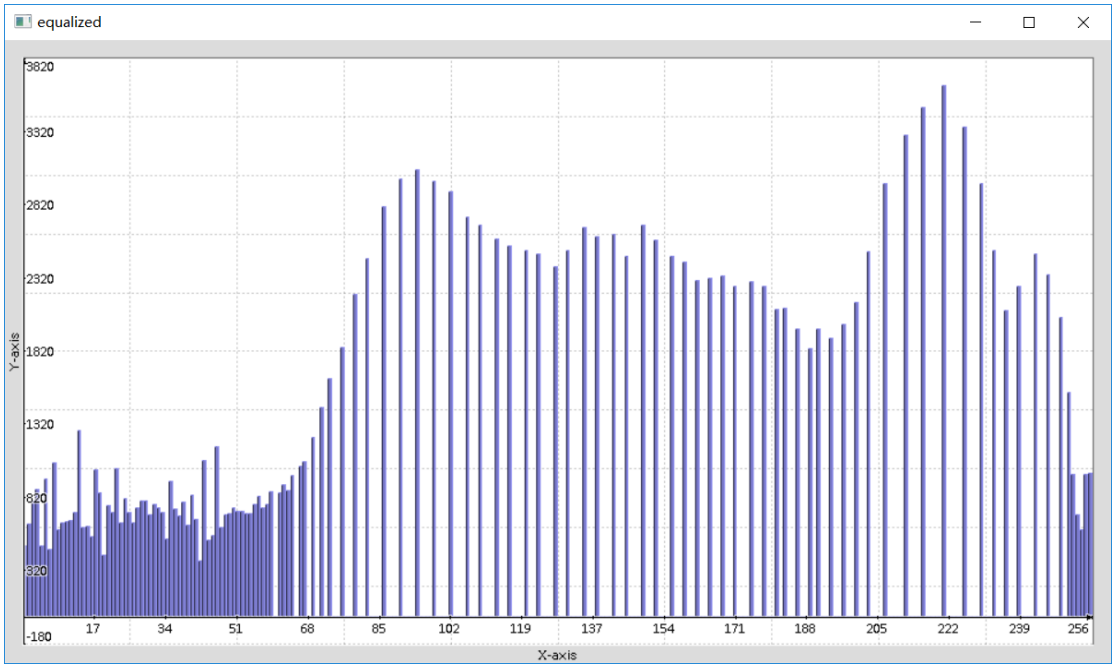


明显能看出图像的对比度增加，不过有些地方变得有些过暗。

打印亮度直方图观察，原图：

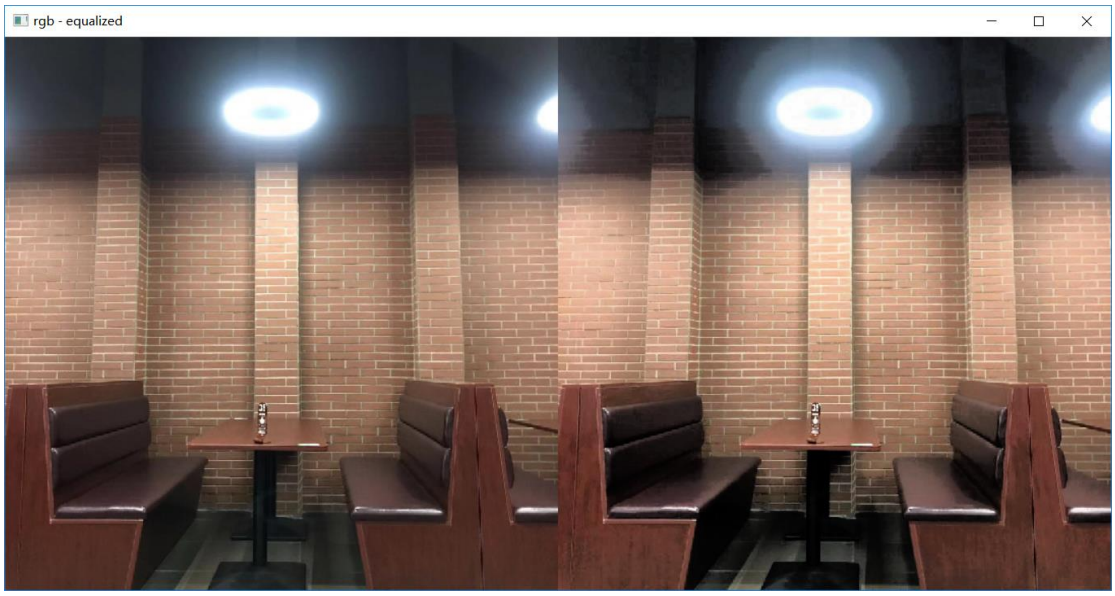


处理后:



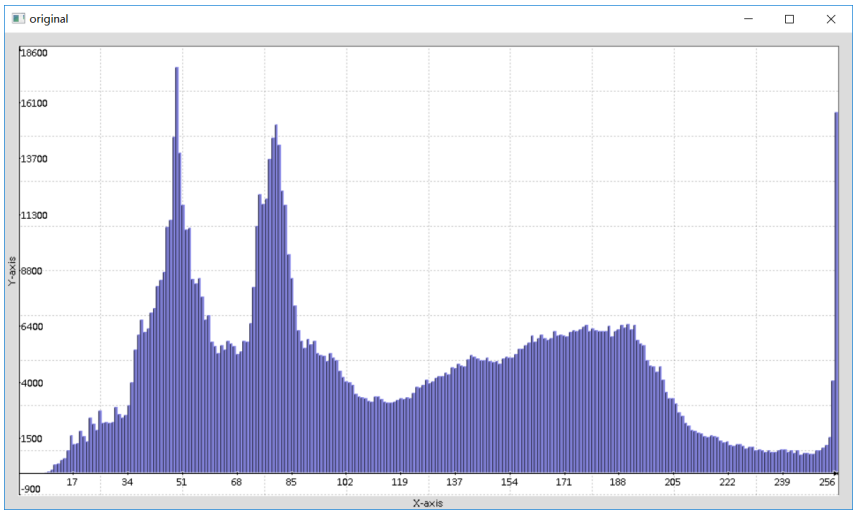
可以明显看出，亮度的分布从原本的集中在中间偏右与最左变为均匀分布，稍有缺陷的地方就在于过暗的部分有些多。

2.

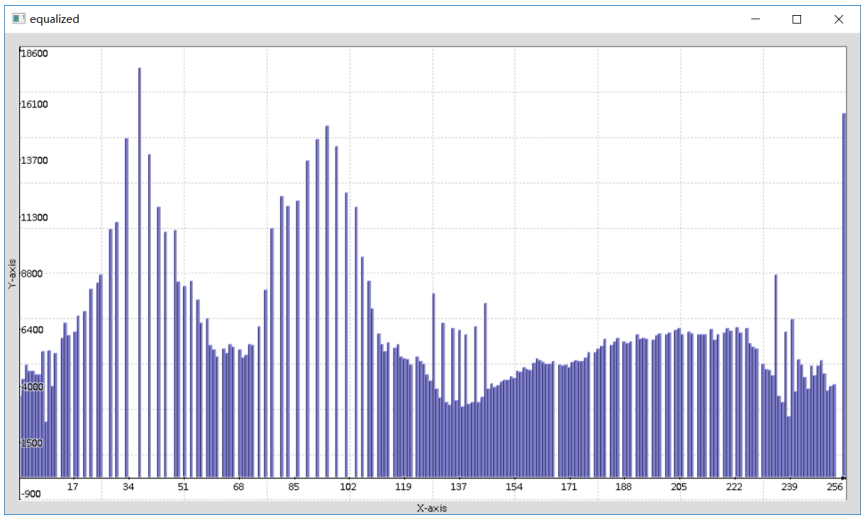


这张图的区别也体现在了对比度的加深，可以看到桌子下方以及座椅一些地方的明暗对比更深。同时，处理灰度图时的问题依然存在：唯一光源周围的光晕。

打印亮度直方图观察，原图：

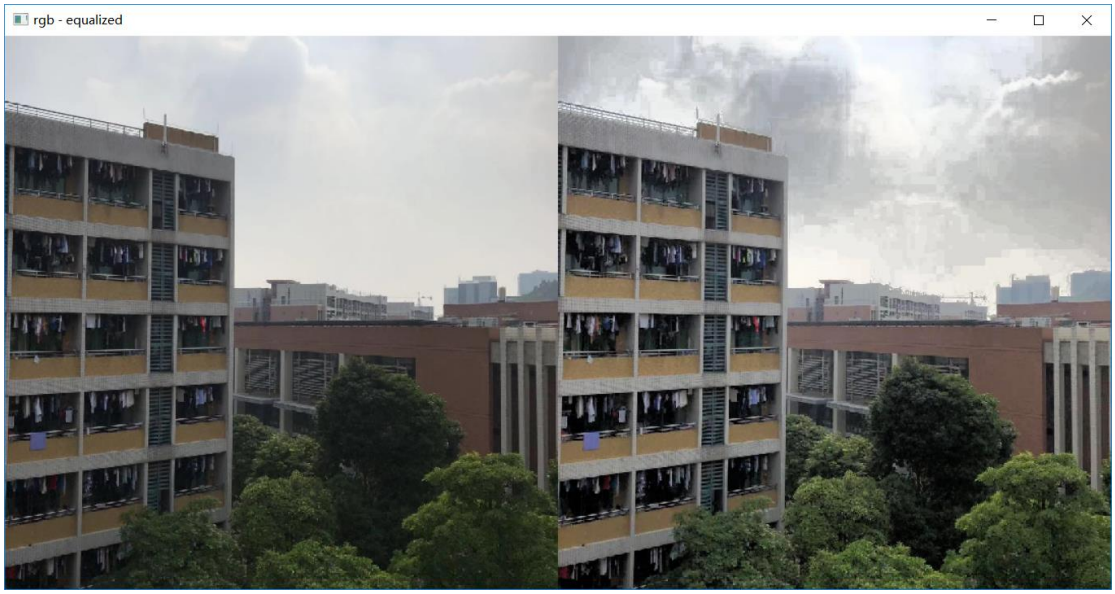


处理后：



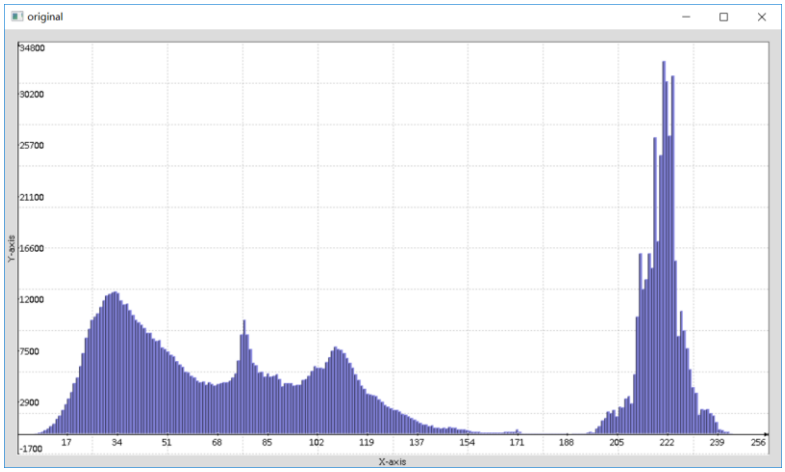
从直方图可见，原图的亮度呈一个多峰型，同时分布极为密集，而且无法覆盖所有亮度级。而处理之后亮度的分布更加均匀，基本扩展到了全亮度级。

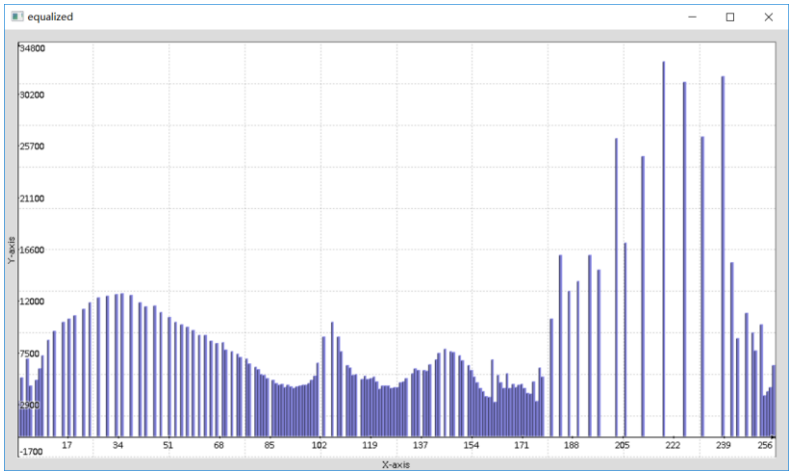
3.



这张图片的处理效果要比前两张图更好，在提升对比度的同时使得暗处更亮了，细节更加容易辨认。

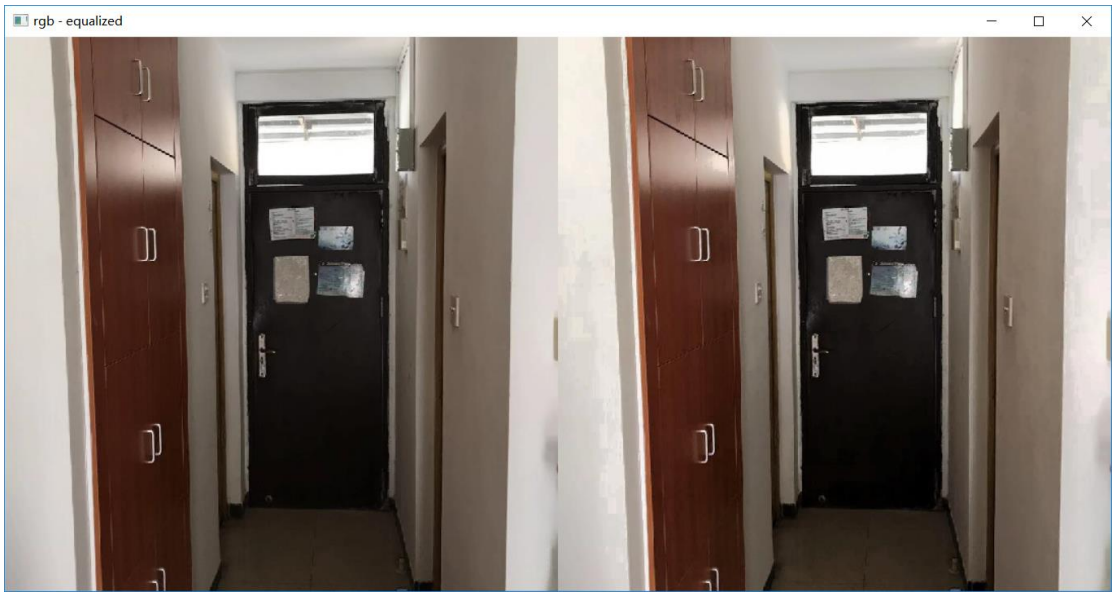
打印亮度直方图观察，原图：



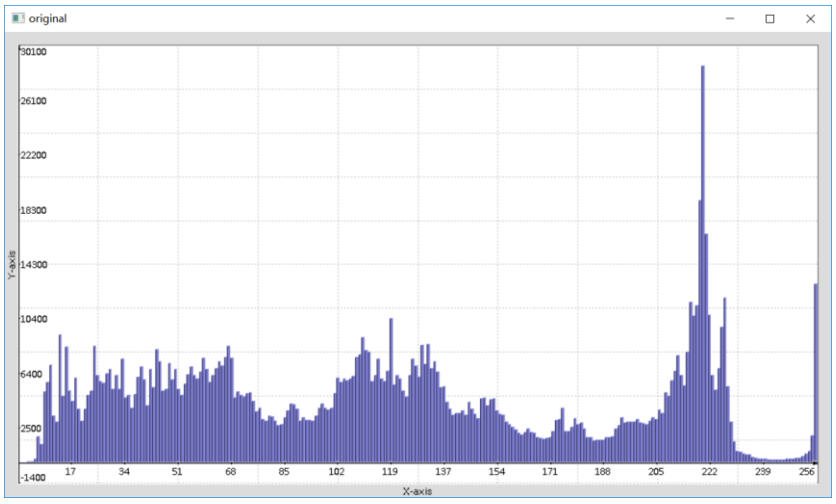


还是同样的，均衡化效果非常好。

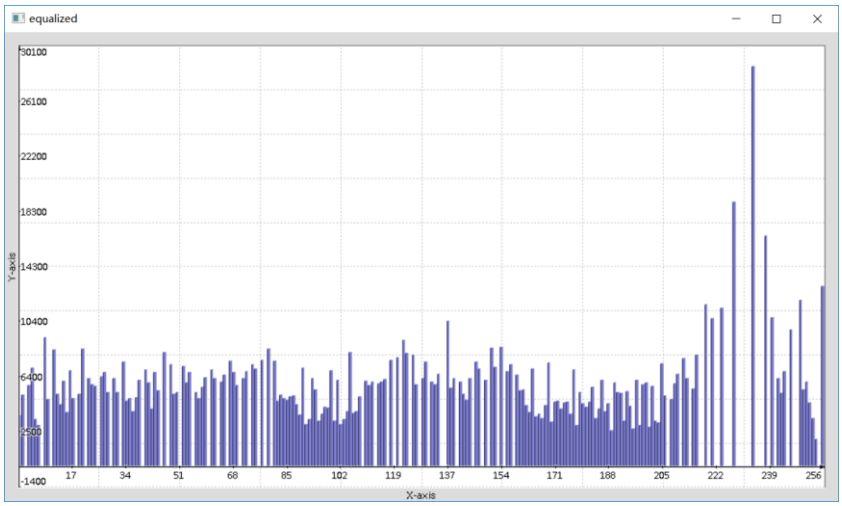
4.



这张图在处理前后变化不大，因为原图就是在一种亮度充足的情况下拍摄的照片，借助亮度直方图观察一下，原图：

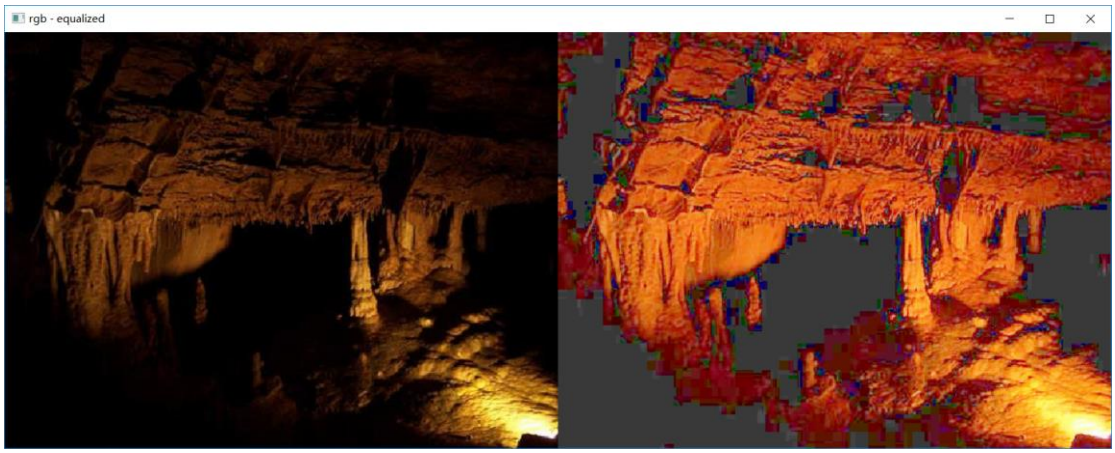


处理后：



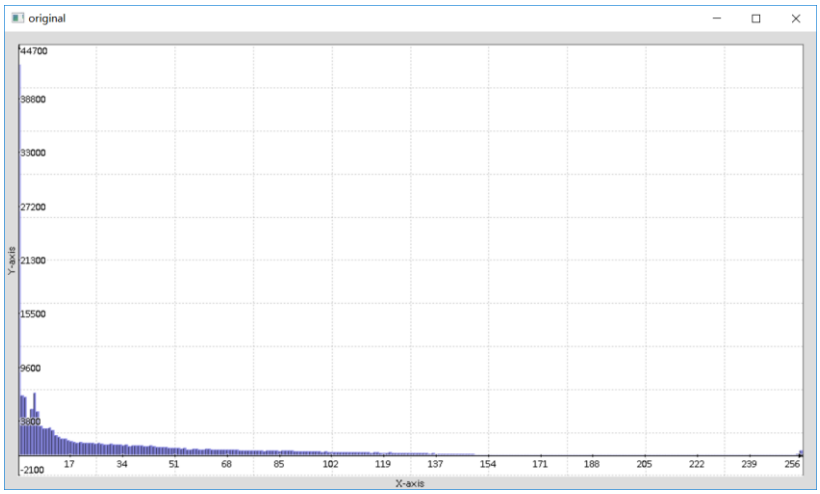
可见虽然肉眼看不太出来（原图就比较均匀），但是直方图均衡化还是有发挥作用的，整张图的亮度分布变得更加均匀，对一些非常密集的部分做了筛除。

5.

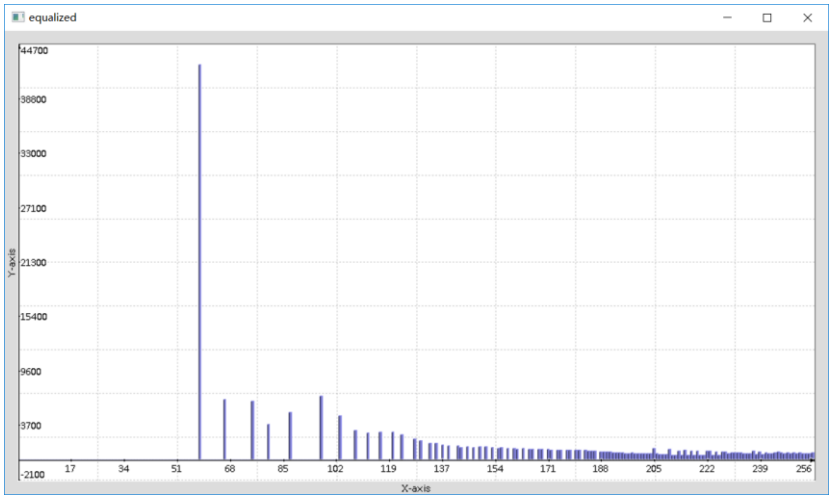


这张图就体现出直方图均衡化最强大的功能了，它可以使得一些过暗的地方的细节重新显示出来，就比如这张图中，原图基本是看不清洞穴的细节了，处理后却能够较好的复原大部分。

借助亮度直方图观察，原图：



处理后：



从上面两张图可以看出，这幅图像实际已属于最糟糕的那一类了：整体的亮度都在极暗

处密集分布，而处理后还是能够较好的均衡化。

经过测试，对于彩色图像做直方图均衡化（实际就是对亮度均衡），能够较好的提升图片的对比度，提取暗处的细节，有时也会使得图片中暗处更暗一些（图一图二），但总的来说还是较好的优化图像的方法，特别是对于那些很暗的图像。

颜色转换

颜色转换相对于直方图均衡化来说要更加复杂，它涉及到了许多颜色空间的转换：RGB 到 XYZ，XYZ 到 LMS，LMS 到 lab，在 lab 空间上做颜色转换最为重要的一部分：Color constancy correction，它包含对于平均值与标准差的计算。最后再将 lab 空间得到的结果转换到 RGB 空间进行显示。

首先对原图和提供转换颜色的图都做 RGB→lab：

```
    //RGB → LMS
    cimg_forXY(img, x, y){
        double L = 0.3811 * img(x, y, 0) + 0.5783 * img(x, y, 1) + 0.0402
* img(x, y, 2);
        double M = 0.1967 * img(x, y, 0) + 0.7244 * img(x, y, 1) + 0.0782
* img(x, y, 2);
        double S = 0.0241 * img(x, y, 0) + 0.1228 * img(x, y, 1) + 0.8444
* img(x, y, 2);
        if (L == 0) L = 1;
        if (M == 0) M = 1;
        if (S == 0) S = 1;
        temp(x, y, 0) = log(L);
        temp(x, y, 1) = log(M);
        temp(x, y, 2) = log(S);
    }
    //LMS → lab
    cimg_forXY(img, x, y){
        result(x, y, 0) = 1/sqrt(3) * temp(x, y, 0) + 1/sqrt(3) * temp(x,
y, 1) + 1/sqrt(3) * temp(x, y, 2);
        result(x, y, 1) = 1/sqrt(6) * temp(x, y, 0) + 1/sqrt(6) * temp(x,
y, 1) - 2/sqrt(6) * temp(x, y, 2);
        result(x, y, 2) = 1/sqrt(2) * temp(x, y, 0) - 1/sqrt(2) * temp(x,
y, 1);
    }
```

注意在 RGB → LMS 的部分，对 L、M、S 取对数时，要先检查它们是否等于 0，如果等于 0 那么应该先将它们置 1 再取对数，最初我没有进行这一步就导致图片显示出来非常奇怪。

得到原图与提供转换颜色的图的 lab 后，就要进入 Color constancy correction：对于原图处理得到的 lab 空间中的每个值，做以下操作，其中< l >代表原图 lab 空间中 l 的平均值，$< l_t >$代表转换图 lab 空间中 l 的平均值，以此类推。

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

$$l^* = l - \langle l \rangle \qquad \alpha' = \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha^* \qquad l' = \frac{\sigma_t^l}{\sigma_s^l} l^* + \langle l_t \rangle$$

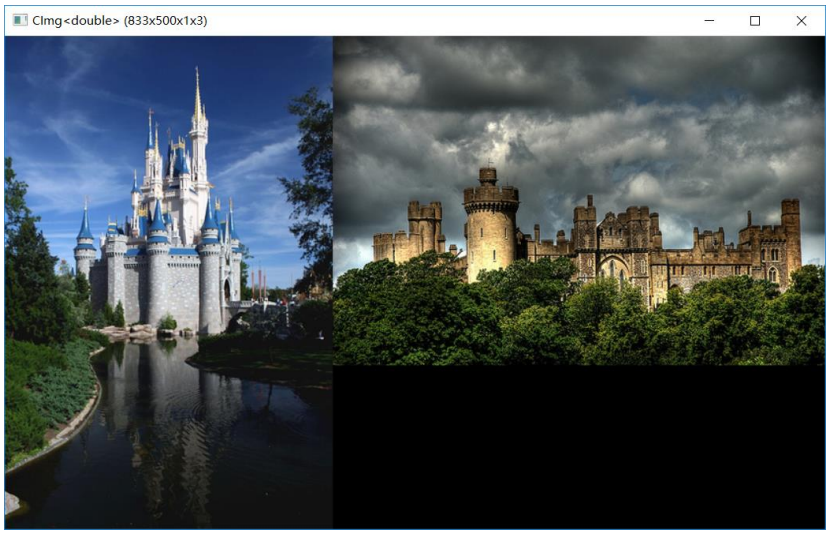
$$\alpha^* = \alpha - \langle \alpha \rangle \qquad \beta' = \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta^* \qquad \alpha' = \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha^* + \langle a_t \rangle$$

$$\beta^* = \beta - \langle \beta \rangle \qquad \beta' = \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta^* + \langle b_t \rangle$$

最后再将得到的 lab 空间转换回 RGB 空间。

以下是图像测试：

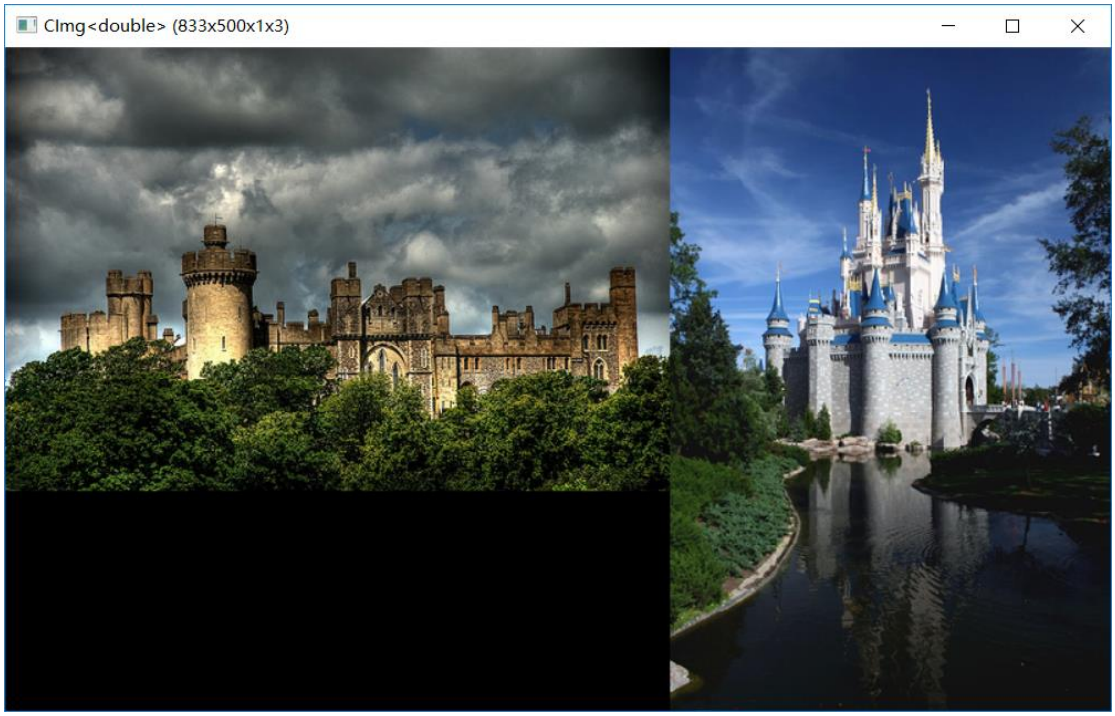
1. 原图与转换图：



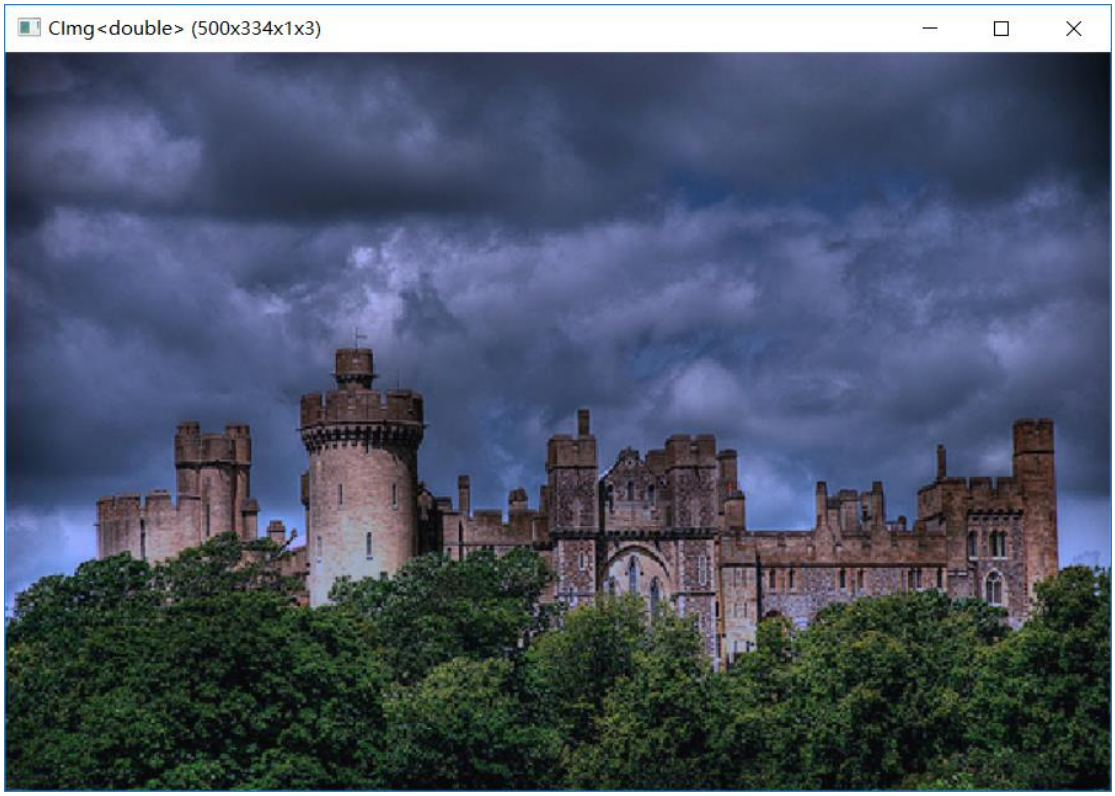
转换后：

很好的将右图的比较昏黄的色调转换到了左图上，不过也有一些问题，注意到左图的云彩（天空）这部分，也转为了昏黄的色调，而非对应右图的天空。

2. 原图与转换图与一相反：

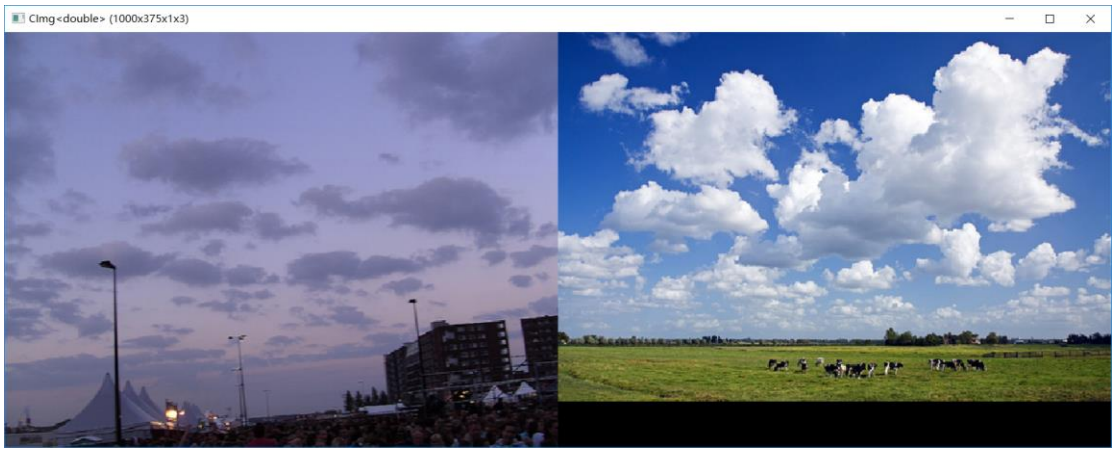


处理结果：

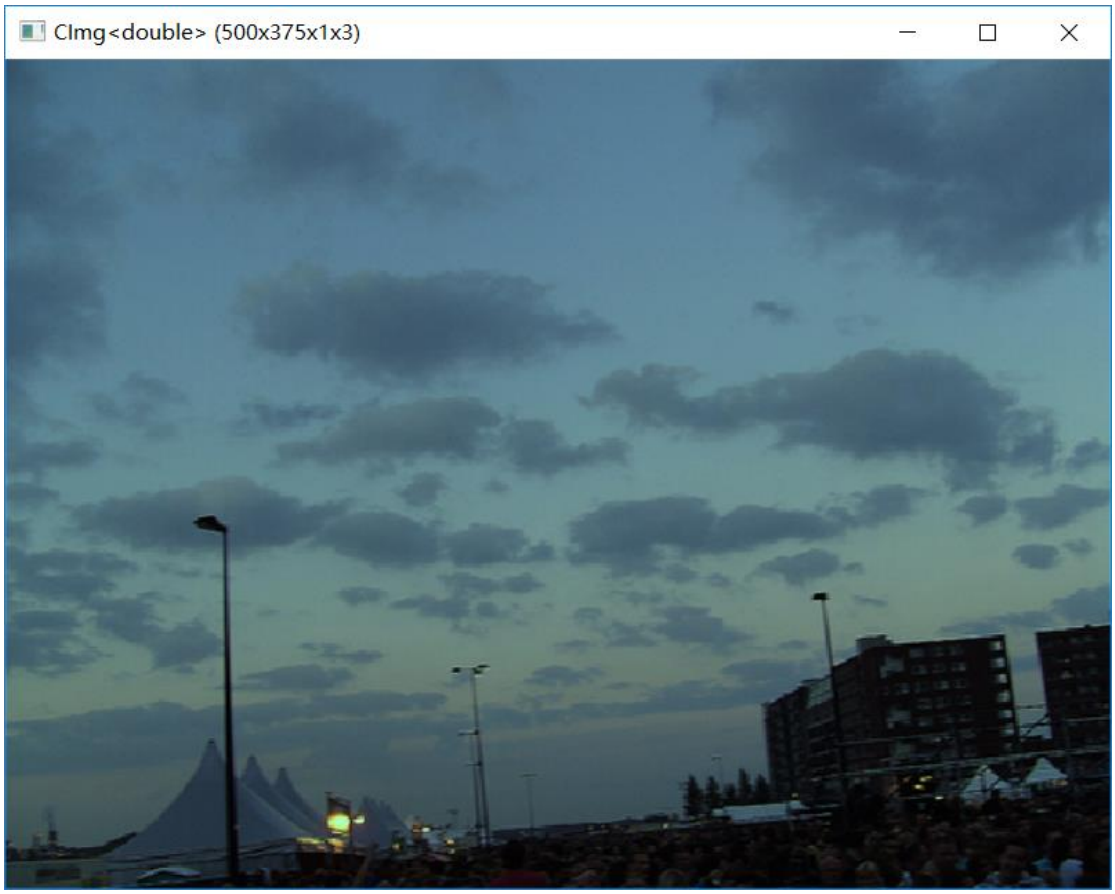


可以看到，整个图的颜色确实转换成了右图的配色，效果还是比较好的。

3. 原图与转换图：

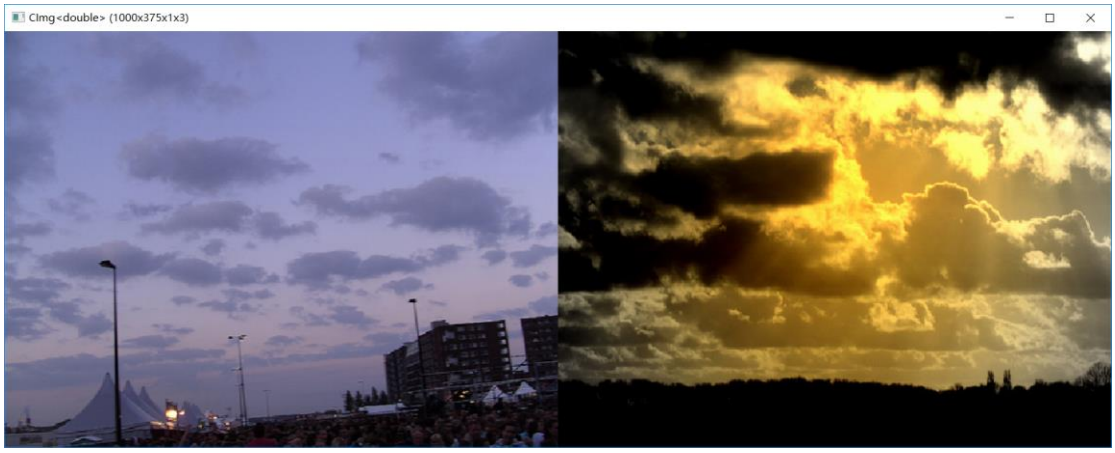


转换后：

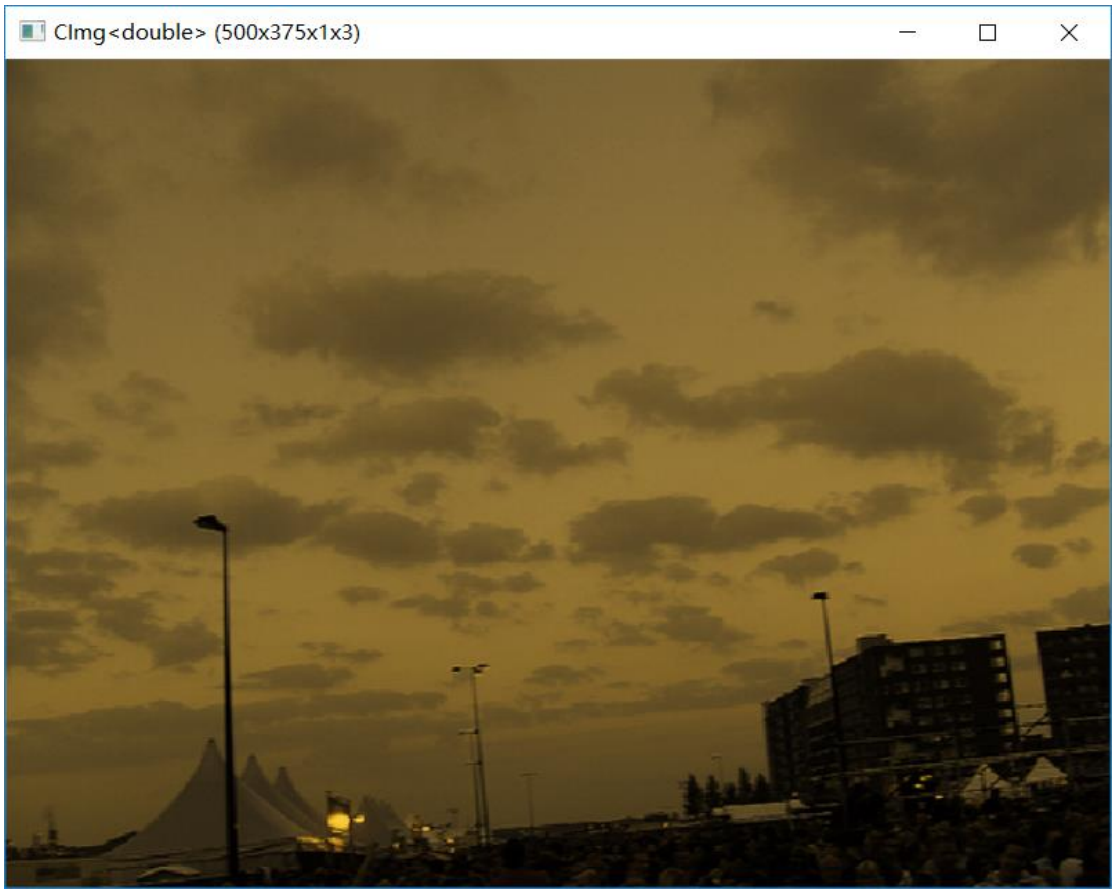


可以看到原图有大片的淡紫色，而转换图是蓝天白云的镜像，转换后的图由大片的淡紫色转为天蓝色，还是比较自然的。

4. 原图与转换图：

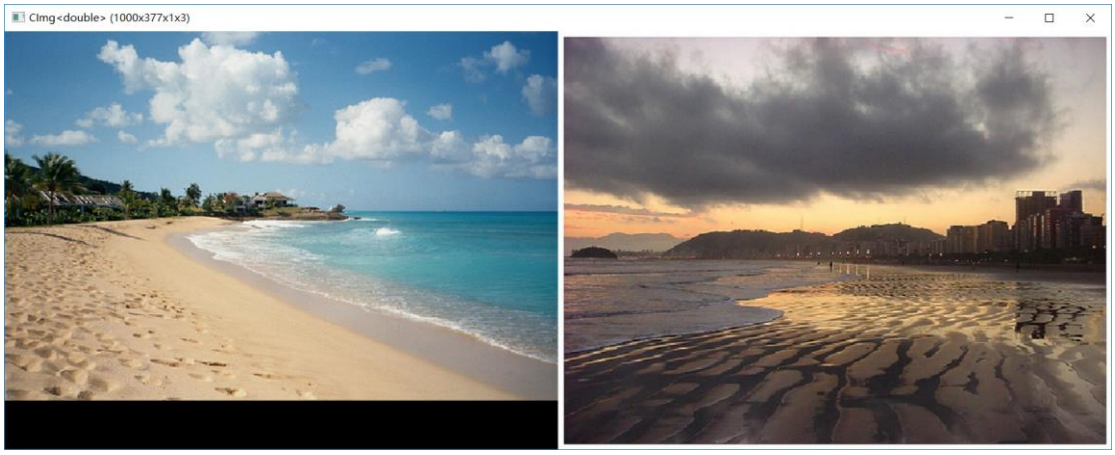


转换后:

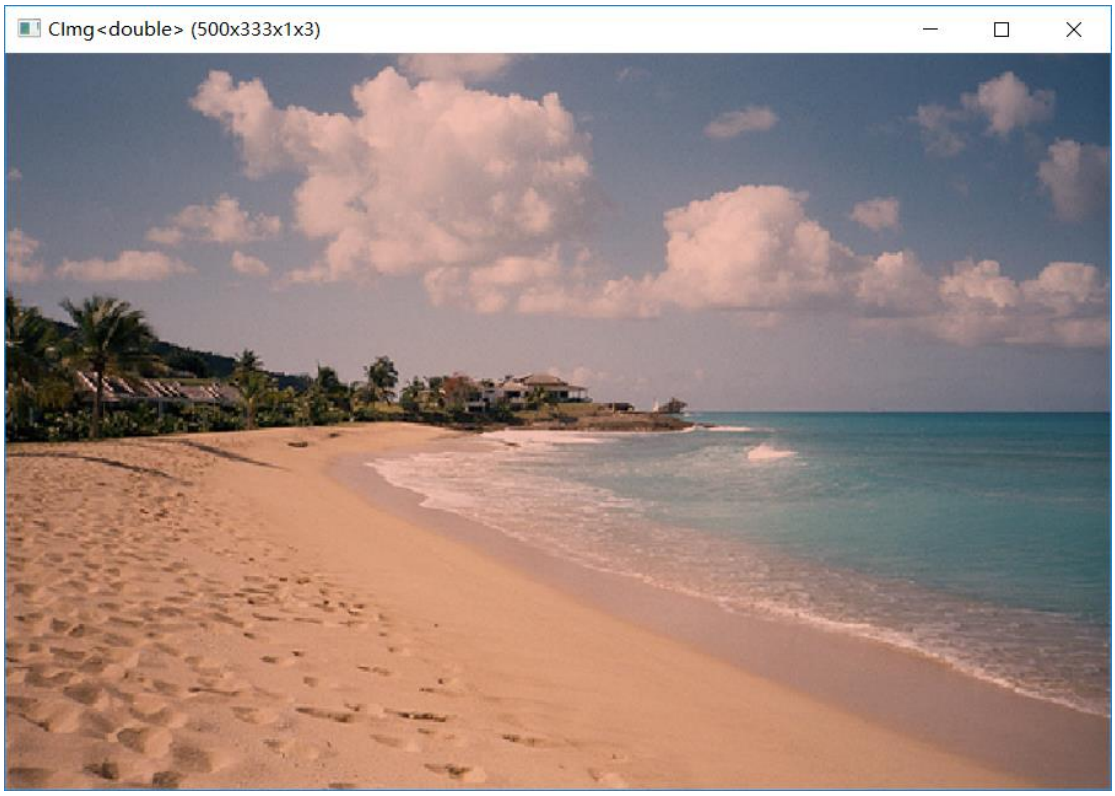


转换后这个差别还是非常大的，整体还是正确的转换到了右图的颜色，只是不像图三那么搭配，效果还是正常。

5. 原图与转换图:



转换后:



这张图处理的就有些不尽人意了，主要是没有很好的呈现出右图的感觉，只像是色调有些变化。

经过测试后可以看到，如果两图的构图都比较简单且结构相近，那么颜色转换可以很好的工作。但如果图片比较复杂，此时想要颜色转换自动对各个区域分别转换是不现实的，这时候就需要对特别区域进行指定，分别进行转换才能得到好的结果。比如说最后一张图，两个图中的元素基本一样：沙滩，天空、云彩、远方的山。此时就需要对这些区域分别进行转换。