实验六.图像拼接

实验要求：实现图像拼接算法，将给定的图像拼成一个。

实验步骤：

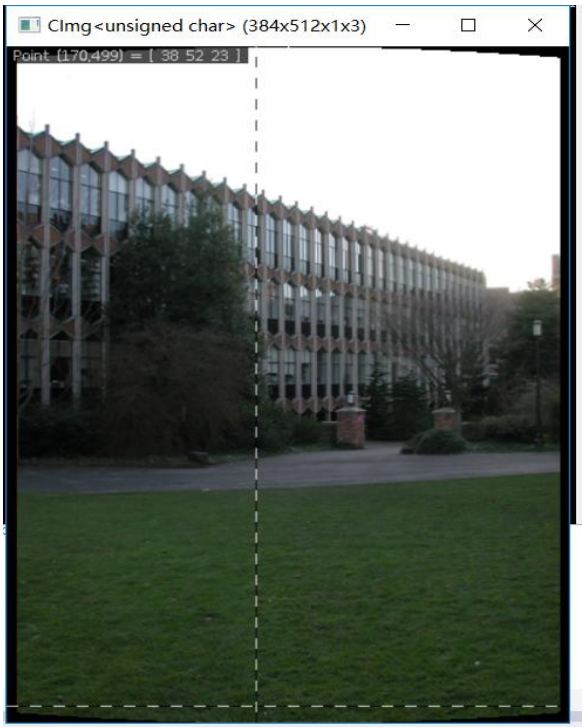
1.柱面投影：

通过三角形相似求取对应柱面坐标的原坐标，得到的原坐标可能不是整数，此时需要坐标附近的像素值进行双线性插值。

比如：求得的原坐标为 15.1,30.5

此时应该对原图像上（15,30）（16,30）（15,31）（16,31）的四个坐标进行双线性插值。

结果：



2. SIFT 特征提取

这里使用 vlfeat 已经实现了的 sift 来做特征点提取。

初始化过滤器。noctaves = 金字塔的组数，nlevels = 每组金字塔的层数， o_min = 每组金字塔从那一层开始，

```
SiftFilt = vl_sift_new(ciSrc.width(), ciSrc.height(), noctaves, nlevels, o_min);
```

处理第一组金字塔

```
if (vl_sift_process_first_octave(SiftFilt, ImageData) != VL_ERR_EOF) {
```

检测第一组金字塔的关键点

```
vl_sift_detect(SiftFilt);
```

生成关键点描述子

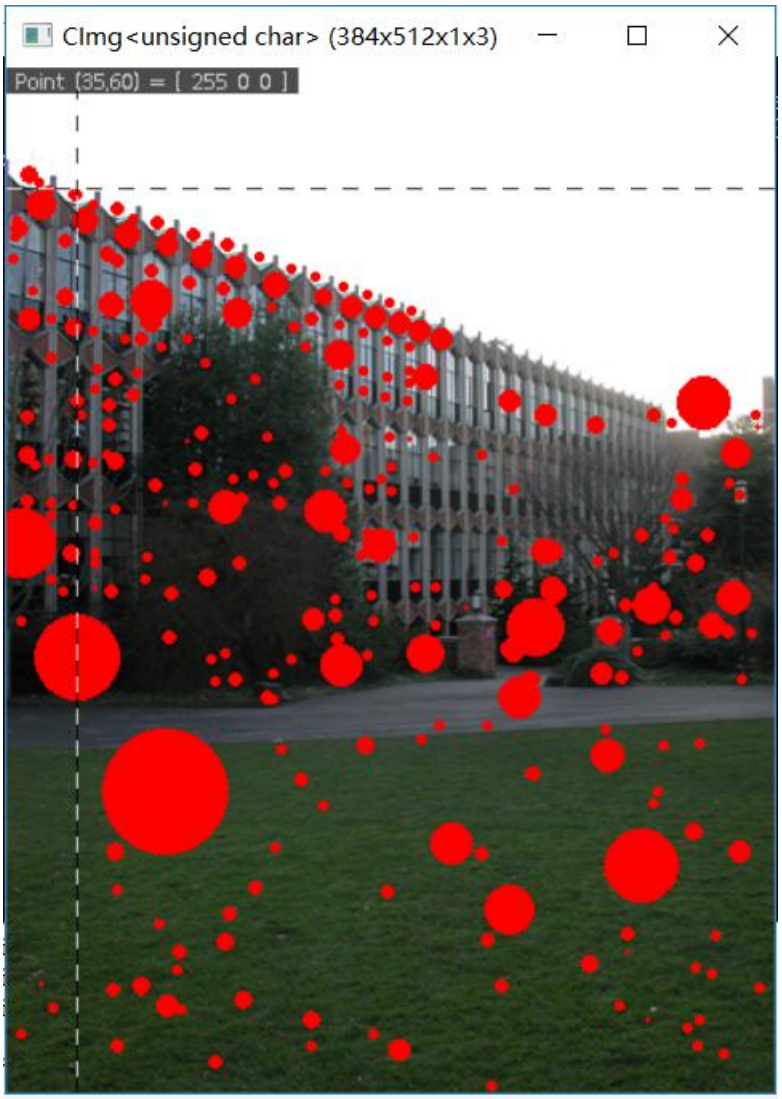
```
vl_sift_calc_keypoint_descriptor(SiftFilt, Descriptors, &TempKeyPoint, TemptAngle);
```

我的关键点描述子存储类型

```
struct KeyPointDescriptor {
    int angleOffset;
    vl_sift_pix orientation;
    VlSiftKeypoint keypoint;
    vl_sift_pix Descriptor[128];
};
```

将每一层探测到的关键点及生成的关键点描述子存储到一个以上述结构体为基础类型的 vector 里。

结果：

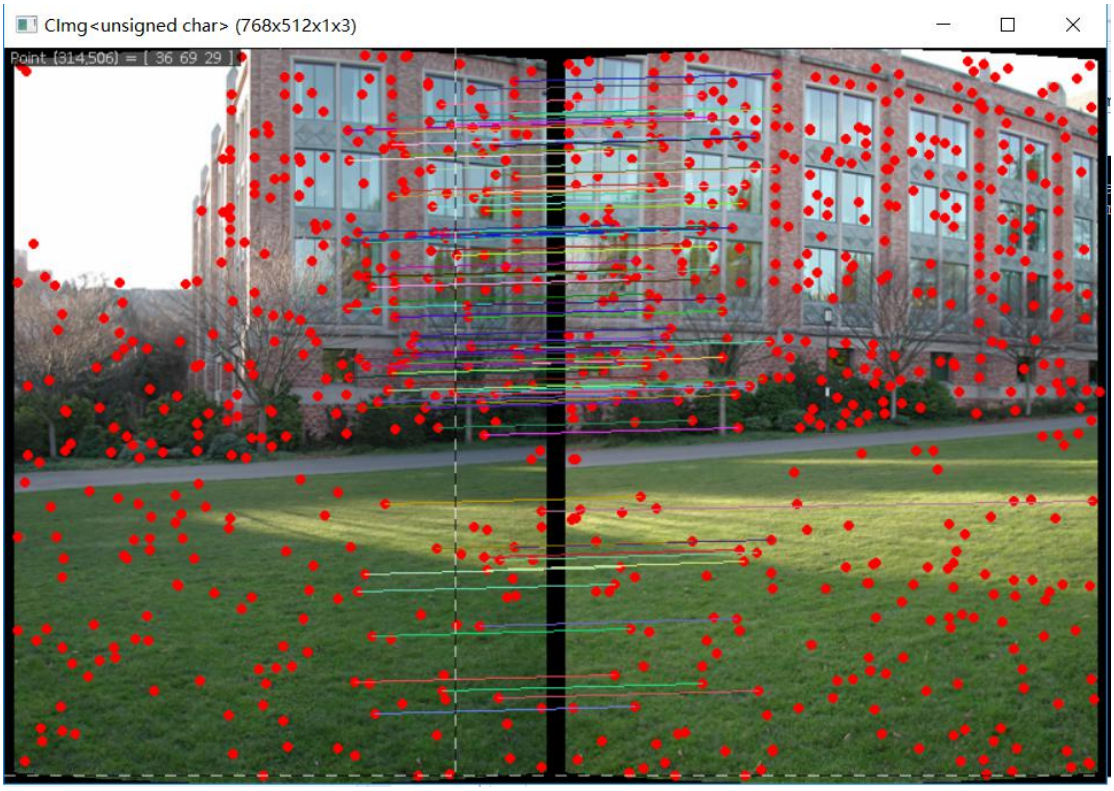


3. 特征点匹配

```
inline vector<MatchPair> FeatureMatching( vector<mySift::KeyPointDescriptor>&left,  vector<mySift::KeyPointDescriptor>&right) {
    vector<MatchPair> ret;
    for (int i = 0; i < left.size(); i++) {
        mySift::KeyPointDescriptor leftkp = left[i];
        mySift::KeyPointDescriptor first,second;

        FeatureMatching(leftkp, right,first,second);
        auto dis1 = CalculateDistance(leftkp, first);
        auto dis2 = CalculateDistance(leftkp, second);

        if (dis1/dis2 <= 0.4) {
            //match
            MatchPair mp;
            mp.distance = CalculateDistance(leftkp, first);
            mp.pointLeft = leftkp;
            mp.pointRight = first;

            //cout << "left:" << leftkp.keypoint.x << ", " << leftkp.keypoint.y << " right:" << first.keypoint.x << ", " << first.keypoint.y << "\n";
            //cout << "distance:" << mp.distance<<endl;
            ret.push_back(mp);
        }
    }
    return ret;
}
```

这里使用了比较笨的穷举法，将第一张图片的每个特征点和第二张图片的每个特征点匹配，筛选出离之最近的两个特征点，计算其离最近特征点的距离和次近的特征点的距离。求出二者的比值，如果比值小于 0.4，则认为该特征点和最近特征点视为一个好的匹配。

结果：



4. Ransac 筛选特征点对，并求取转换矩阵

使用的转化矩阵如下

X*p0+y*p1+x*y*p2+p3 = x’　　x*p4+y*p5+x*y*p6 + p7 = y’

为了求解 p0-7 八个参数需要四对匹配点八个方程。

x0,y0 ,x0*y0,1　　p0　　x0’

X1,y1,x1*y1,1　　p1　　x1’

X2,y2,x2*y2,1　　p2　　x2’

X3,y3,x3*y3,1　　p3　　x3’

x0,y0 ,x0*y0,1　　p4　　y0’

X1,y1,x1*y1,1	p5	y1’
X2,y2,x2*y2,1	p6	y2’
X3,y3,x3*y3,1	p7	y3’

使用 cimg 的 get_solve()方法解两个方程。

Ransac 算法：

首先算出需要几次循环，k = log(1-P)/log(1-pow(p,n));

每次循环随机选四组不同的匹配点对，用上述方法计算出转移参数，

用所有的匹配点对 测试该转移参数的 inliners，同时更新拥有最多 inliners 的一个 vector。

经过 k 次循环后对最大 inliners 按照上述方法计算出真实转移参数。

5. 图像拼接

拼接两张图片：

按照前几部的方法计算出两张图片之间的转移参数。

假设计算的是从第一张到第二张的转换参数。

为了防止第一张图片转换后超过边界，需要 求一下转换后图像的新边界。

Newwidth 和 newHeight 新建一个 CImg<unsigned char>对象，用来接收转换后的图像。将第二张图片填充到新的图像上面。

结果：

6. Blend

没做。