图像拼接

一． 环境&工具

操作系统：ubuntu14.04

模拟工具：octave4.0

相关库：OpenCV3.1

编译器：g++ 4.8.4

运行说明：

1．OpenCV3 开始，features2d 库已经被移除，相关性质被放到 xfeatures2d 库中，例如 SIFT、SURF 特征检测。然而一般的 OpenCV3 安装，不包括 OpenCV 的附件库，opencv_contrition 中的模块

附加库配置指南：

https://nishantnath.com/2015/10/19/open-cv-3-x-adding-contrib-support-to-default-installation/

http://www.cnblogs.com/asmer-stone/p/5089764.html

Github 上的附加库源码：

https://github.com/Itseez/opencv_contrib/tree/master/modules/xfeatures2d

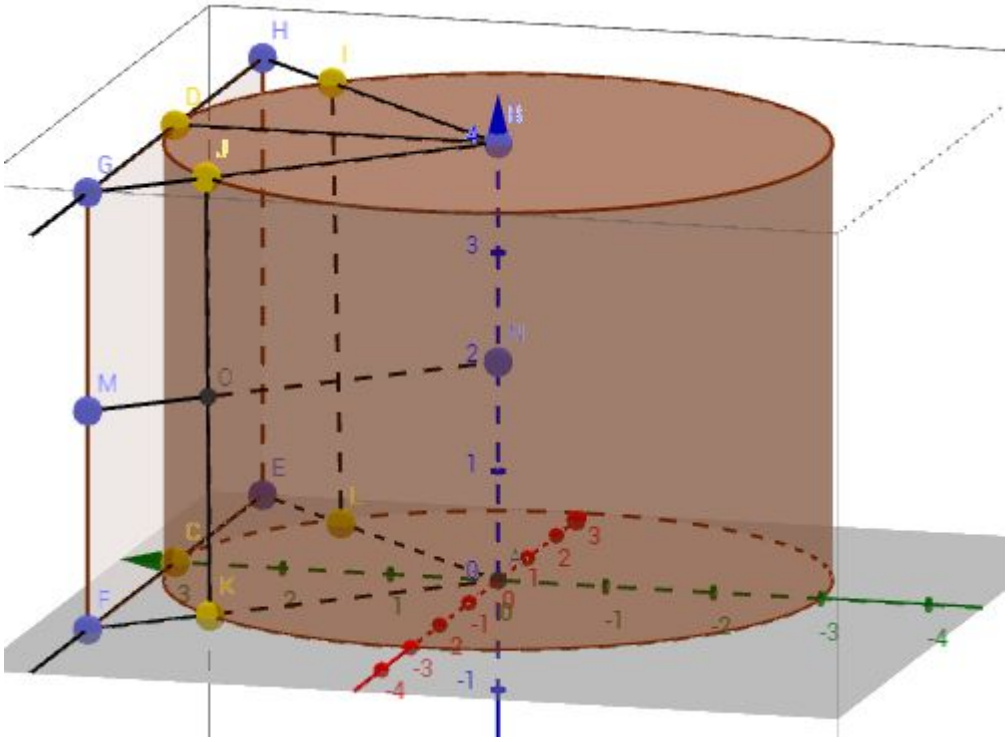
最后，opencv_contrition 中的模块并不完善，也不太稳定，可能运行的时候会有一些 bug，比如我的实验过程中就出现了同一个编译结果，同一输入，会发生运行错误以及正常运行两种情况。

二． 原理

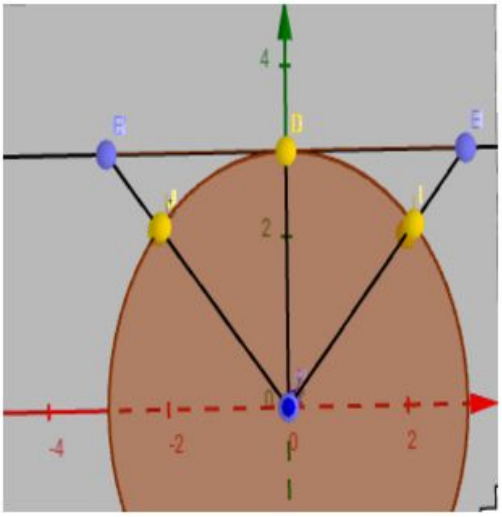
1．柱面投影

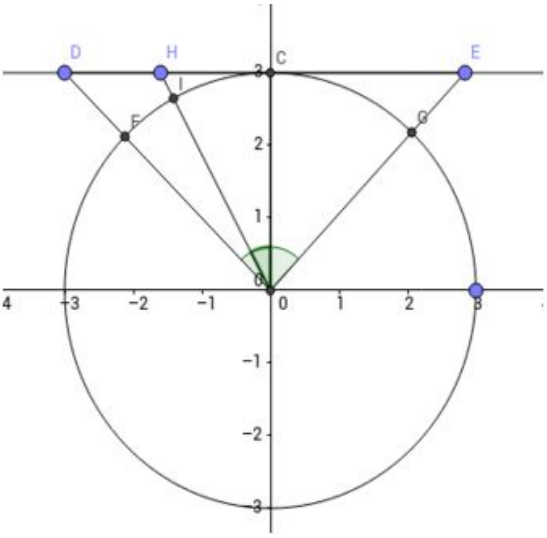
把平面图像投影到圆柱的曲面上。

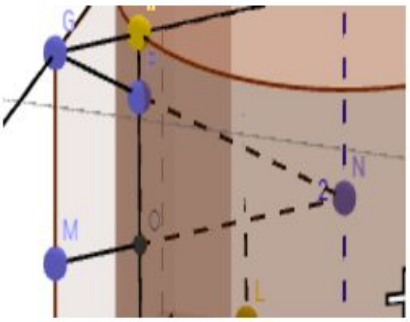
如下图，四边形 GHEF 表示待处理原图，投影之后，变成曲面 JDILCK（黄色点标注）

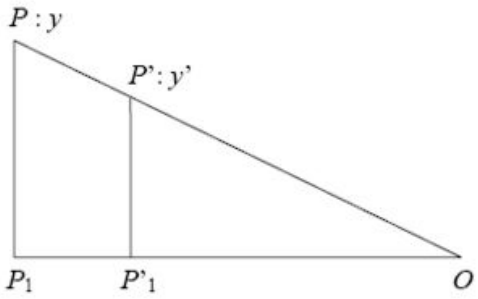


俯视图如下，DCE 为待处理图像平面，FCG 为投影所得曲面。









设，原图像宽 W，高 H，角度 FOG 为相机视场角度α（一般为 45°，即 PI/4），圆形半径（焦距）f 有 tan 1/2α = W / (2 * f), 则有 f = W / (2 * tan(α/2))

依次推算出，目标图像的宽（曲线 FCG 长）W‘ = f * α， 目标图像高 H’不变，H‘ = H

第一种推算：以图像像素原始坐标计算（即，左上角为原点）

$$\begin{cases} x' = f \cdot \operatorname{arctg}\left(\dfrac{x-\dfrac{W}{2}}{f}\right) + f \cdot \operatorname{arctg}\left(\dfrac{W}{2f}\right) \\ y' = \dfrac{f \cdot \left(y-\dfrac{H}{2}\right)}{\sqrt{\left(x-\dfrac{W}{2}\right)^2 + f^2}} + \dfrac{H}{2} \end{cases}$$

第二种推算：设置图像原点为（W/2， H/2），用以简化计算步骤

结果如下图：

2．特征点检测、提取、匹配

可以用 OpenCV 自带的 API，检测角点：

1）SIFT 检测 + BFMatcher

2）SURF 检测 + FlannBasedMatcher

3．计算透视矩阵

OpenCV 中自带有：分别求透视矩阵和该矩阵下的顶点变化

findHomography
perspectiveTransform

4．图像拼接融合

利用透视矩阵，进行拼接

三． 代码迁移到 C++

a） 预处理（柱面投影、旋转缩放

柱面投影（此处估计相机的视角为 30 度

```
//柱面投影
Mat Cylinder_projection(Mat& input) {
        int width = input.cols;
        int height = input.rows;
        int centerX = width / 2;
        int centerY = height / 2;

        Mat output = input.clone();
        double f = width / (2 * tan(PI / 6 / 2)); //视角设为30度

        double theta, pointX, pointY;
        for (int i = 0; i < height; i ++) {
                for (int j = 0; j < width; j ++) {
                        theta = asin((j - centerX) / f);
                        pointY = f * tan((j - centerX) / f) + centerX;
                        pointX = (i - centerY) / cos(theta) + centerY;
                        for (int k = 0; k < input.channels(); k ++) {
                                if (pointX >= 0 && pointX <= height && pointY >= 0 && pointY <= width)
                                        output.at<Vec3b>(i, j)[k] = input.at<Vec3b>(pointX, pointY)[k];
                                else
                                        output.at<Vec3b>(i, j)[k] = 0;
                        }
                }
        }
        return output;
```

针对 dataset2 的特殊处理（旋转缩放

```
//dataset 2 预处理
Mat pre_processing(Mat input) {
        //缩小为1/4
        //逆时针90度
        int row = input.rows / 4;
        int col = input.cols / 4;
        Mat output = Mat::zeros(col, row, CV_8UC3);
        for (int i = 0; i < col; i ++) {
                for (int j = 0; j < row; j ++) {
                        if(input.rows - 4 * j >= 0 && input.rows - 4 * j < input.rows && input.cols - 4 * i >=
                                output.at<Vec3b>(i, j) = input.at<Vec3b>(input.rows - 4 * j, input.cols - 4 *
                }
        }
        return output;
}
```

b） 角点检测（此处采用 surf 算法的特征检测

```
Ptr<SURF> detector =  SURF::create();
std::vector<KeyPoint> keypoints_1, keypoints_2, kp1, kp2;

detector->detect( img_1, kp1);
detector->detect( img_2, kp2);
```

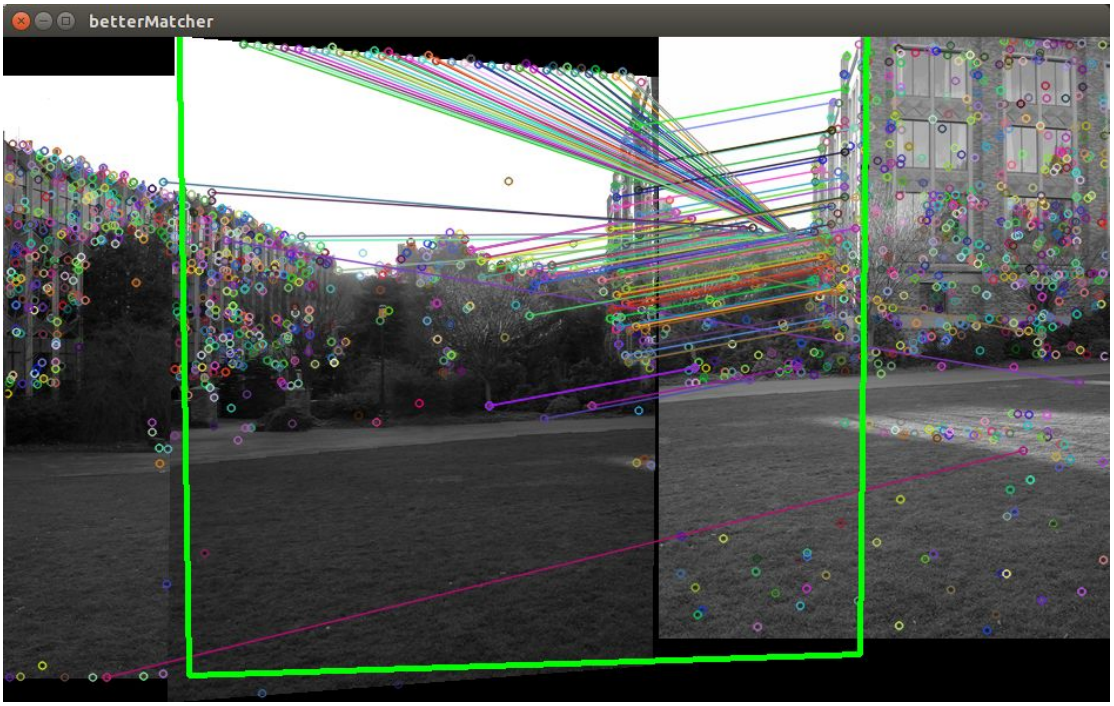
<KeyPoint>的结构如下：（可以知道，记录了特征点在原图的坐标、角度等信息

特征点类：

```
class KeyPoint
{           Point2f  pt;  //坐标
        float  size; //特征点邻域直径
        float  angle; //特征点的方向，值为[零,三百六十)，负值表示不使用
        float  response;
        int  octave; //特征点所在的图像金字塔的组
        int  class_id; //用于聚类的id
}
```

此处还应该对特征点进行提纯，左右待拼接图的重叠部分只有最小图的 1/2 宽不到，同时也应该防止多张图片拼接之后边界角点出现干扰正常角点的情况

如图：

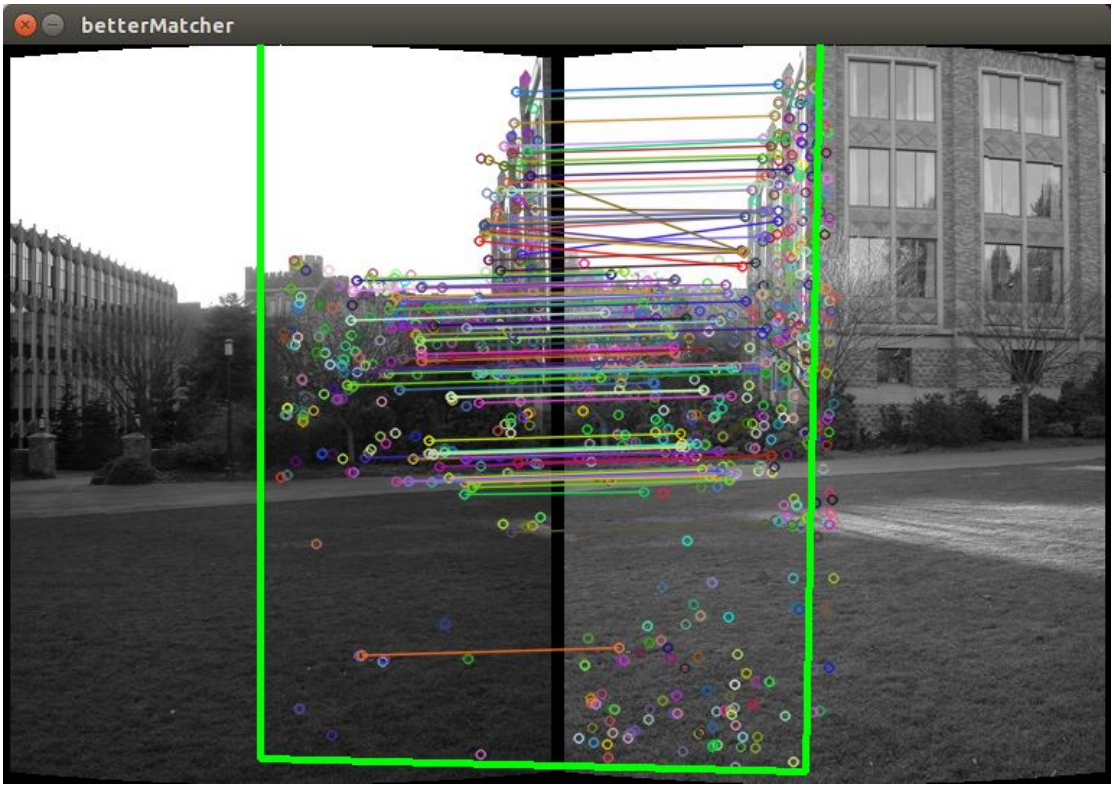


所以此处也是一个可以优化的步骤

```
int thv = input2.cols > input1.cols ? input1.cols : input2.cols;
thv /= 2;
int range = 3;
for (int i = 0; i < kp1.size(); i ++) {
        if(kp1[i].pt.x > input1.cols - thv) {
                int count = 0;
                for(int p = - range; p <= range; p ++)
                        for (int q = - range; q <= range; q ++) {
                                if(kp1[i].pt.y + p >= 0 && kp1[i].pt.y + p < input1
                                        count++;
                        }

                if (count <= 6) keypoints_1.push_back(kp1[i]);
        }
}
```

优化之后：



c） 特征描述子

由于 SIFT 算法中构造的高斯差分金字塔中搜索得到的特征点需要一个结构体来记录描述（位置、方向等

```
//描述子
Mat descriptors_1, descriptors_2;
detector->compute(img_1, keypoints_1, descriptors_1);
detector->compute(img_2, keypoints_2, descriptors_2);
```

d） 匹配

OpenCV 中给出的 API，FlannBasedMatcher 可以做最佳匹配的运算

```
FlannBasedMatcher matcher;
vector< DMatch > matches;
matcher.match(descriptors_1, descriptors_2, matches);
Mat matche_result;
drawMatches(img_1, keypoints_1,  img_2, keypoints_2, matches, matche_result);
//imshow("##", matche_result);
```

其中，DMatch 的结构如下：

存放匹配结果的结构：

```
struct DMatch
{           //三个构造函数
        DMatch():
queryIdx(-1),trainIdx(-1),imgIdx(-1),distance(std::numeric_limits<float>::max()) {}
        DMatch(int  _queryIdx, int  _trainIdx, float  _distance ) :
queryIdx( _queryIdx),trainIdx( _trainIdx), imgIdx(-1),distance( _distance) {}
        DMatch(int  _queryIdx, int  _trainIdx, int  _imgIdx, float  _distance ) :                qu
eryIdx(_queryIdx), trainIdx( _trainIdx), imgIdx( _imgIdx),distance( _distance) {}
        intqueryIdx;  //此匹配对应的查询图像的特征描述子索引
        inttrainIdx;   //此匹配对应的训练(模板)图像的特征描述子索引
        intimgIdx;    //训练图像的索引(若有多个)
        float distance;  //两个特征向量之间的欧氏距离，越小表明匹配度越高。
        booloperator < (const DMatch &m) const;
};
```

这样我们就可以利用 DMatch 的 distance 信息再做一次过滤，distance 越小，匹配度越高

e) 提取好的匹配点

```
float threshold =  3 * min_dis;
//float threshold =  max_dis;
cout << "min_dis = " << min_dis << "   max_dis = " << max_dis << endl;
std::vector<DMatch> betterMatcher;
for (int i = 0; i < matches.size(); i ++) {
        if (matches[i].distance < threshold) {
                betterMatcher.push_back(matches[i]);
        }
}
```

f) 透视矩阵 H

```
vector<Point2f> KP1, KP2;
for (int i = 0; i < betterMatcher.size(); i ++) {
        KP1.push_back(keypoints_1[betterMatcher[i].queryIdx].pt);
        KP2.push_back(keypoints_2[betterMatcher[i].trainIdx].pt);
}
Mat H = findHomography( KP1, KP2, CV_RANSAC);
```

g） 拼接

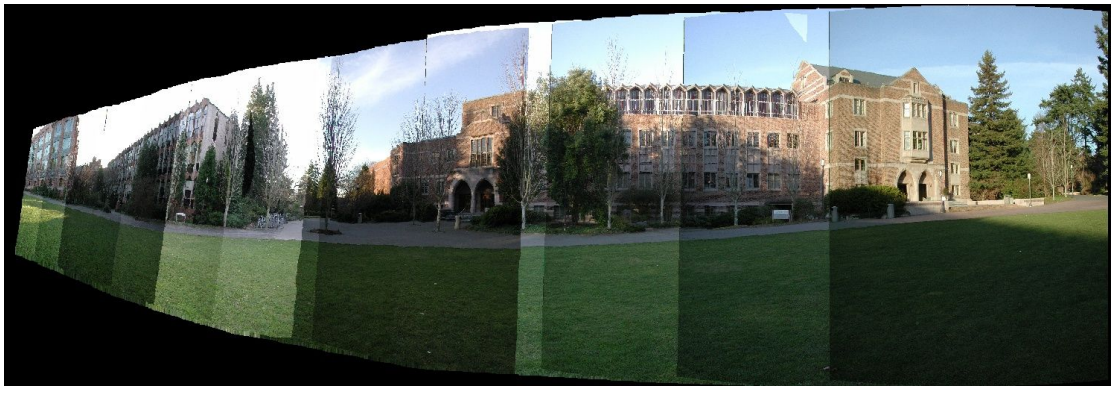
四． 效果比较

1．Dataset1

2．Dataset2

i. 从左，一张一张拼到右

（由于图太多，拼到十张左右，最左边的图像发生很大的形变，很影响后面几张的透视矩阵计算

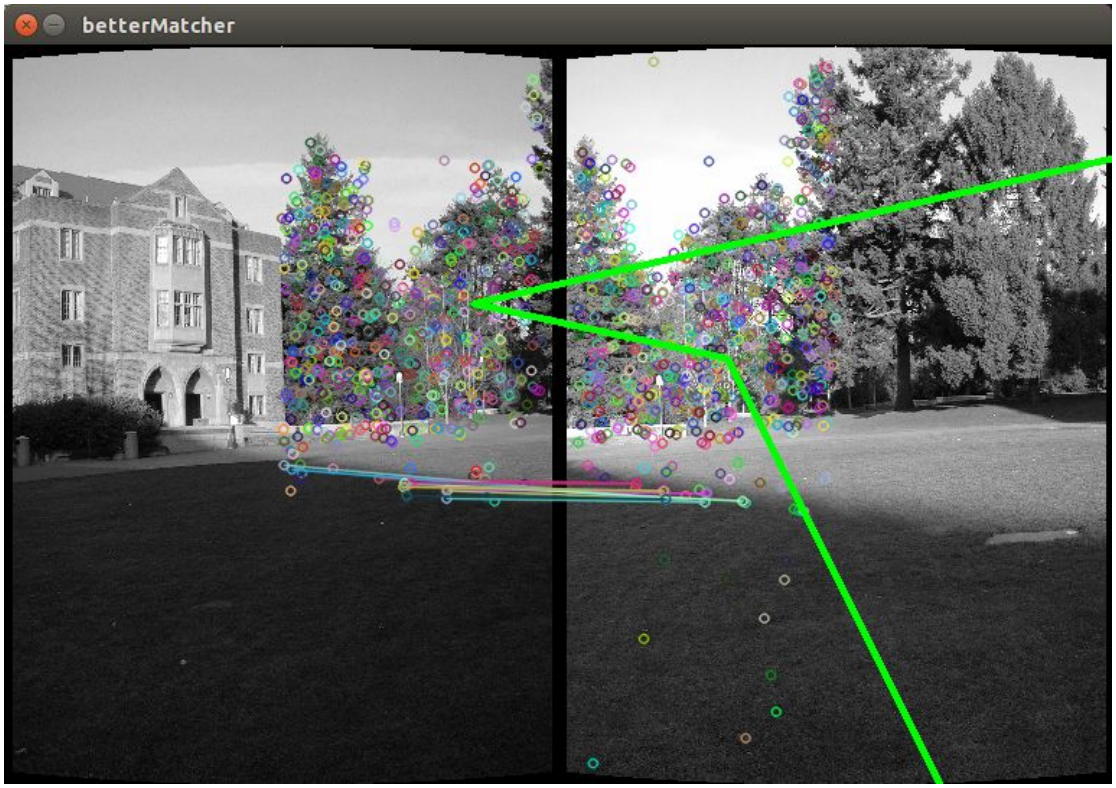


ii. 最左，最右分别开始拼，最后得到一张完整图

（效果不好，因为形变和源素材的关系，匹配不好，导致最终结果没有很理想

4 张小图的效果

但是大图，由于第 19 和 20 张的匹配不好，所以无法拼接



以下为 1-19,20-25 的拼接