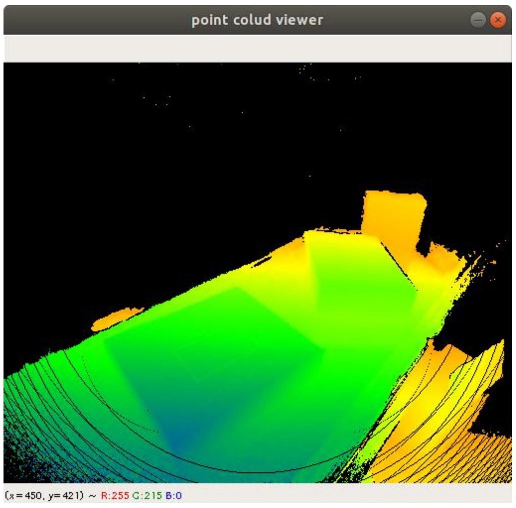
期末考试实验报告

算法的基本思想：

观察深度图得到照相机在图像的右后方，当箱子出现在图像靠左时，更靠近照相机，便于计算。故取 pc0.csv 为背景，分别用 pc85.csv 、pc135.csv、pc180.csv、pc260.csv、pc330.csv 来提取第 1-5 个箱子，把点云分割出来之后，在转化为 pcd 格式，用 pcl 中的最小包围矩形算法来计算最小包围的矩形，即为相应箱子的尺寸大小。

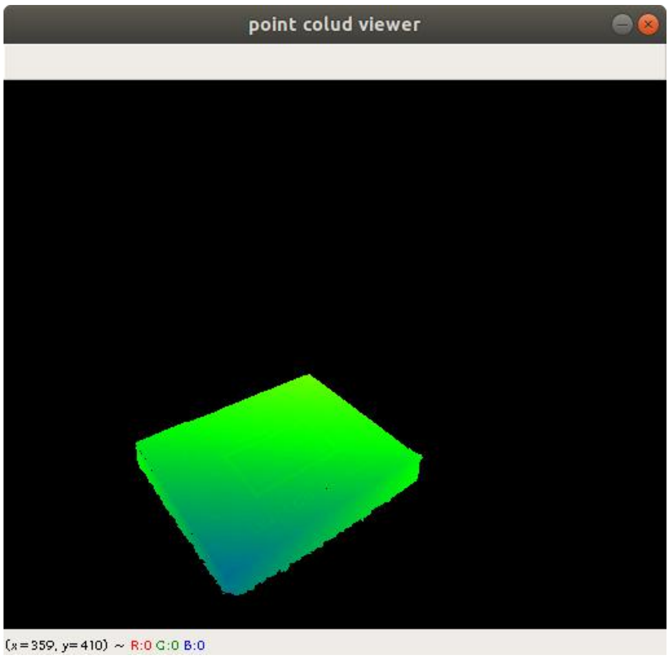
算法流程(以最后一个箱子的计算为例)

1.保留距离相机 1.5 米内的物体。

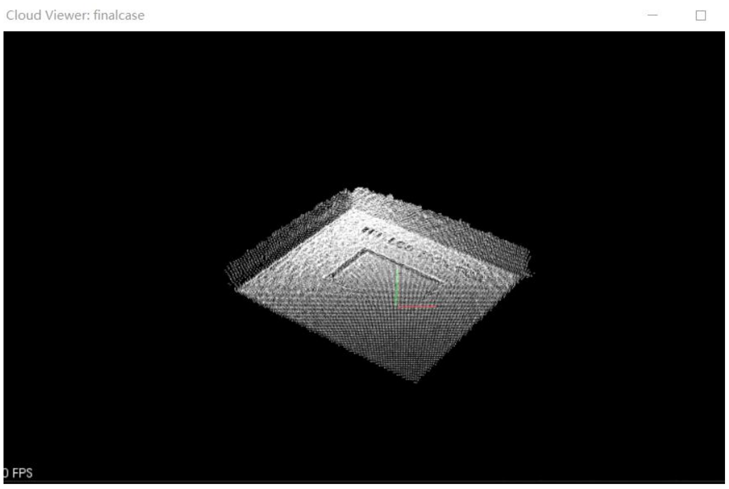


2. 过滤掉距离背景近的点、离群点，使用区域生长法进行分割再去掉小的物体。将最后一个箱子的点云取出来，存到文件 **pc_ojb2.csv** 中。深度图如下：

误差缘于噪音。pcl 取得的是最小包围矩形，所以比实际的大小要稍大。个人认为可以先用 ransac 算法取得一个面上的点，然后对这个面上的点进行处理，以得到更好效果。



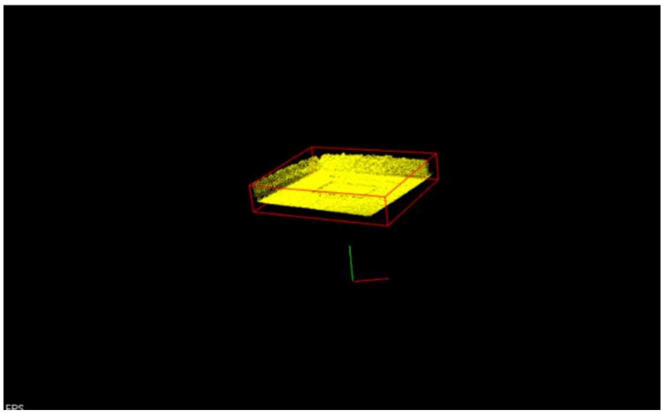
3. 在 windows 下使用程序将 pc_obj2 转为 pcd 格式，可视化结果如下：



4. 使用 pcl 中求最小包围矩形的算法，计算得

0.40 0.33 0.076

误差缘于噪音。pcl 取得的是最小包围矩形，所以比实际的大小要稍大。个人认为可以先用 ransac 算法取得一个面上的点，然后对这个面上的点进行处理，以得到更好效果。


类似计算得到其他箱子的参数。

Python 代码：

```
#!/usr/bin/python3
# coding=utf-8

import numpy as np
from pc_view import pc_view
import os



os.chdir(r'/home/che/final_test/data_csv2')
CAM_HGT, CAM_WID = 424, 512
CAM_CX = 2.543334e+02
CAM_CY = 1.967047e+02
CAM_FX = 3.664861e+02
CAM_FY = 3.664861e+02
CAM_F = (CAM_FX + CAM_FY) * 0.5

IMG_HGT, IMG_WID = CAM_HGT, CAM_WID
IMG_SZ = IMG_WID * IMG_HGT
IMG_SHAPE = (IMG_HGT, IMG_WID)
#pc = np.genfromtxt('pc0.csv', delimiter=',').astype(np.float32)
#pc_view(pc, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT)

#程序中用到的参数
TH_DIST = 1.5    #  距离门限，截取这个距离内的点云
TH1 = 0.012    #  背景分离门限
TH2 = 0.008    #  离群点滤除参数和物体点云聚合的距离门限
```

```
print('loading CSV...')
pc_bg = np.genfromtxt('pc0.csv', delimiter=',').astype(np.float32)
pc = np.genfromtxt('pc330.csv', delimiter=',').astype(np.float32)

pc_view(pc, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0, dmax=2)
pc_view(pc_bg, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)



#print(pc.shape)

print('clipping by distance...')

pc_sel = np.array([p for p in pc if p[2] < TH_DIST])
print(pc_sel.shape)
pc_view(pc_sel, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)


import cv2
## 背景提取
print('nn searching...')
flann = cv2.FlannBasedMatcher(dict(algorithm=1, trees=5), dict(checks=50))
matches = flann.knnMatch(pc_sel, pc_bg, k=1)

## 保留非背景点
print('filtering...')
pc_sel = np.array([p for p, m in zip(pc_sel, matches) if m[0].distance > TH1])
print(pc_sel.shape)
pc_view(pc_sel, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)


print('noise filter by distance...')
matches = flann.knnMatch(pc_sel, pc_sel, k=5)
pc_sel = np.array([p for p, m in zip(pc_sel, matches) if max([i.distance for i in m]) < TH2])
pc_view(pc_sel, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)




## 基于邻域拓展的物体分割
```

```
def pc_obj_merge(pc, p0, k, r, mask=None, ret_idx=False):
    flann = cv2.FlannBasedMatcher(dict(algorithm=1, trees=5), dict(checks=50))

    # 指示尚未分割的点云点(这里用 copy()是为了防止修改输入的 mask 对象)
    mask = np.ones(len(pc), dtype=bool) if mask is None else mask.copy()

    pc_chk = [p0]
    idx_out = []
    while len(pc_chk) > 0:
        p = pc_chk.pop()
        # 找到半径 r 领域内的未检查过的点的序号
        idx_nn = [m.trainIdx for m in flann.knnMatch(p, pc, k=k)[0] \
                      if m.distance < r and mask[m.trainIdx]]
        idx_out += idx_nn    # 新增的点云序号加入输出集合
        pc_chk += [pc[i] for i in idx_nn]    # 新增点云加入待检测集合
        mask[idx_nn] = False    # 标注已被处理的点
    return (pc[idx_out], idx_out) if ret_idx else pc[idx_out]

#print(pc_sel.shape)
## 逐个提取里面的物体
pc_obj_list = []
mask = np.ones(len(pc_sel), dtype=bool)
print(np.sum(mask))
while np.sum(mask) > 0:
    # 找到种子点对应的物体
    p0 = pc_sel[np.flatnonzero(mask)[0]]    # 选取种子点

    pc_obj, idx_obj = pc_obj_merge(pc_sel, p0, k=10, r=TH1, mask=mask, ret_idx=True)
    pc_obj_list += [pc_obj]
    mask[idx_obj] = False
    print('p0:', p0, ', size:', len(idx_obj))




# 过滤太小的物体
print('removing small objects...')
print(len(pc_obj_list))
pc_obj1 = pc_obj
pc_obj_list_sel = [obj for obj in pc_obj_list if len(obj) > 1000]
print(len(pc_obj_list_sel))

for n, pc_obj in enumerate(pc_obj_list_sel):
    np.savetxt('pc_obj%d.csv' % n, pc_obj, fmt='%.12f', delimiter=',', newline='\n')
    pc_view(pc_obj, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)


pcwhole = np.genfromtxt('pc330.csv', delimiter=',').astype(np.float32)
pc_view(pcwhole, CAM_FX, CAM_FY, CAM_CX, CAM_CY, CAM_WID, CAM_HGT, cz=1, dmin=0,
dmax=2)



import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

ax = plt.figure(1).gca(projection='3d')
ax.plot(pc2[:, 0], pc2[:, 1], pc2[:, 2], 'b.', markersize=0.5)
plt.title('point cloud')
plt.show()

import matplotlib.pyplot as plt

ax = plt.figure(1).gca(projection='3d')
ax.plot(pc_remain[:, 0], pc_remain[:, 1], pc_remain[:, 2], 'b.', markersize=0.5)
plt.title('point cloud')
plt.show()
```

PCL 代码：

Pcl 安装

https://blog.csdn.net/lhm_19960601/article/details/81196640
https://www.bilibili.com/video/av56040593?from=search&seid=6111246468897334557

测试 pcl 是否安装成功

```
#include <iostream>
```

```
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>

int
main(int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZ> cloud;

    // Fill in the cloud data
    cloud.width = 5;
    cloud.height = 1;
    cloud.is_dense = false;
    cloud.points.resize(cloud.width * cloud.height);

    for (size_t i = 0; i < cloud.points.size(); ++i)
    {
        cloud.points[i].x = 1024 * rand() / (RAND_MAX + 1.0f);
        cloud.points[i].y = 1024 * rand() / (RAND_MAX + 1.0f);
        cloud.points[i].z = 1024 * rand() / (RAND_MAX + 1.0f);
    }

    pcl::io::savePCDFileASCII("test_pcd.pcd", cloud);
    std::cerr << "Saved " << cloud.points.size() << " data points to test_pcd.pcd." <<
std::endl;

    for (size_t i = 0; i < cloud.points.size(); ++i)
        std::cerr << "    " << cloud.points[i].x << " " << cloud.points[i].y << " " <<
cloud.points[i].z << std::endl;

    return (0);
}
```

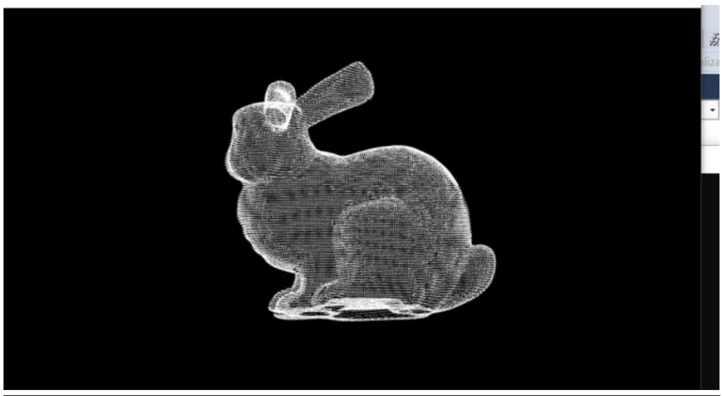
Pcl 可视化

显示斯坦福兔子

```
#include<pcl/visualization/cloud_viewer.h>
#include<iostream>//标准 C++库中的输入输出类相关头文件。

#include<pcl/io/io.h>
```

```
#include<pcl/io/pcd_io.h>//pcd 读写类相关的头文件。
#include<pcl/io/ply_io.h>
#include<pcl/point_types.h> //PCL 中支持的点类型头文件。
using namespace std;
using namespace pcl;
void viewerOneOff(visualization::PCLVisualizer& viewer) {
    viewer.setBackgroundColor(0, 0, 0);
};
int main() {
    PointCloud<PointXYZ>::Ptr cloud(new PointCloud<PointXYZ>);
    char strfilepath[256] = "E:\\rabbit.pcd";
    if (-1 == io::loadPCDFile(strfilepath, *cloud)) {
        cout << "error input!" << endl;
        return -1;
    }
    cout << cloud->points.size() << endl;
    visualization::CloudViewer viewer("Cloud Viewer: Rabbit");
    viewer.showCloud(cloud);
    viewer.runOnVisualizationThreadOnce(viewerOneOff);
    system("pause");
    return 0;
}
```



把点云数组转化为 pcd 格式

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
```

```
using namespace std;

void main()
{
    pcl::PointCloud<pcl::PointXYZ> cloud;
    //ofstream outFile("Data.csv", ios::out);
    ////ios::out：如果没有文件，那么生成空文件；如果有文件，清空文件
    //if (!outFile)
    //{
    //  cout << "打开文件失败！" << endl;
    //  exit(1);
    //}
    ////写入 3 行数据
    //for (int i = 0; i < 3; i++)
    //{
    //  outFile << 12 << ",";
    //  outFile << 13 << ",";
    //  outFile << 14 << ",";
    //  outFile << "NaN" << endl;
    //}
    //outFile.close();
    //cout << "写入数据完成" << endl;



    ifstream inFile("pc_obj2.csv", ios::in);
    if (!inFile)
    {
        cout << "打开文件失败！" << endl;
        exit(1);
    }
    int i = 0;
    string line;
    string field;
//根据点云的点数调整 cloud.width 的值
    cloud.width = 19188;
    cloud.height = 1;
    cloud.is_dense = false;
    cloud.points.resize(cloud.width * cloud.height);
    while (getline(inFile, line))//getline(inFile, line)表示按行读取 CSV 文件中的数据
    {
        string field;
        istringstream sin(line); //将整行字符串 line 读入到字符串流 sin 中
        // Fill in the cloud data


        getline(sin, field, ','); //将字符串流 sin 中的字符读入到 field 字符串中，以逗号为分隔符
        cout << atof(field.c_str()) << " ";//将刚刚读取的字符串转换成 int
        cloud.points[i].x = atof(field.c_str());

        getline(sin, field, ','); //将字符串流 sin 中的字符读入到 field 字符串中，以逗号为分隔符
        cout << atof(field.c_str()) << " ";//将刚刚读取的字符串转换成 int
        cloud.points[i].y = atof(field.c_str());

        getline(sin, field); //将字符串流 sin 中的字符读入到 field 字符串中，以逗号为分隔符
        cout << atof(field.c_str()) << endl;//将刚刚读取的字符串转换成 int
        cloud.points[i].z = atof(field.c_str());
        i++;
    }


    inFile.close();
    cout << "共读取了：" << i << "行" << endl;
    cout << "读取数据完成" << endl;

    pcl::io::savePCDFileASCII("finalcase.pcd", cloud);
    std::cerr << "Saved " << cloud.points.size() << " data points to test_pcd.pcd." << std::endl;

    for (size_t i = 0; i < cloud.points.size(); ++i)
        std::cerr << "    " << cloud.points[i].x << " " << cloud.points[i].y << " " << cloud.points[i].z << std::endl;

    getchar();

}
```

取最小包围矩形的程序

```
#include <iostream>
#include <pcl/ModelCoefficients.h>
#include <pcl/io/pcd_io.h>
```

```
#include <pcl/filters/project_inliers.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/passthrough.h>
#include <pcl/features/normal_3d.h>
#include <pcl/filters/radius_outlier_removal.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/segmentation/extract_clusters.h>
#include <Eigen/Core>
#include <pcl/common/transforms.h>
#include <pcl/common/common.h>


using namespace std;
typedef pcl::PointXYZ PointType;

int main(int argc, char **argv)
{
    pcl::PointCloud<PointType>::Ptr cloud(new pcl::PointCloud<PointType>());
    pcl::io::loadPCDFile("E:\\finalcase.pcd", *cloud);

    Eigen::Vector4f pcaCentroid;
    pcl::compute3DCentroid(*cloud, pcaCentroid);
    Eigen::Matrix3f covariance;
    pcl::computeCovarianceMatrixNormalized(*cloud, pcaCentroid, covariance);
    Eigen::SelfAdjointEigenSolver<Eigen::Matrix3f> eigen_solver(covariance,
Eigen::ComputeEigenvectors);
    Eigen::Matrix3f eigenVectorsPCA = eigen_solver.eigenvectors();
    Eigen::Vector3f eigenValuesPCA = eigen_solver.eigenvalues();
    eigenVectorsPCA.col(2) = eigenVectorsPCA.col(0).cross(eigenVectorsPCA.col(1)); //校
正主方向间垂直
    eigenVectorsPCA.col(0) = eigenVectorsPCA.col(1).cross(eigenVectorsPCA.col(2));
    eigenVectorsPCA.col(1) = eigenVectorsPCA.col(2).cross(eigenVectorsPCA.col(0));

    std::cout << "特征值 va(3x1):\n" << eigenValuesPCA << std::endl;
    std::cout << "特征向量 ve(3x3):\n" << eigenVectorsPCA << std::endl;
    std::cout << "质心点(4x1):\n" << pcaCentroid << std::endl;
    /*
    // 另一种计算点云协方差矩阵特征值和特征向量的方式:通过 pcl 中的 pca 接口，如下，这种情
```

```
况得到的特征向量相似特征向量
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloudPCAprojection (new
pcl::PointCloud<pcl::PointXYZ>);
    pcl::PCA<pcl::PointXYZ> pca;
    pca.setInputCloud(cloudSegmented);
    pca.project(*cloudSegmented, *cloudPCAprojection);
    std::cerr << std::endl << "EigenVectors: " << pca.getEigenVectors() << std::endl;//
计算特征向量
    std::cerr << std::endl << "EigenValues: " << pca.getEigenValues() << std::endl;//计
算特征值
    */
    Eigen::Matrix4f tm = Eigen::Matrix4f::Identity();
    Eigen::Matrix4f tm_inv = Eigen::Matrix4f::Identity();
    tm.block<3, 3>(0, 0) = eigenVectorsPCA.transpose();   //R.
    tm.block<3, 1>(0, 3) = -1.0f * (eigenVectorsPCA.transpose())
*(pcaCentroid.head<3>());//  -R*t
    tm_inv = tm.inverse();

    std::cout << "变换矩阵 tm(4x4):\n" << tm << std::endl;
    std::cout << "逆变矩阵 tm'(4x4):\n" << tm_inv << std::endl;

    pcl::PointCloud<PointType>::Ptr transformedCloud(new pcl::PointCloud<PointType>);
    pcl::transformPointCloud(*cloud, *transformedCloud, tm);

    PointType min_p1, max_p1;   //点云的最大值与最小值点
    Eigen::Vector3f c1, c;
    pcl::getMinMax3D(*transformedCloud, min_p1, max_p1);
    c1 = 0.5f*(min_p1.getVector3fMap() + max_p1.getVector3fMap());

    std::cout << "型心 c1(3x1):\n" << c1 << std::endl;

    Eigen::Affine3f tm_inv_aff(tm_inv);
    pcl::transformPoint(c1, c, tm_inv_aff);

    Eigen::Vector3f whd, whd1;
    whd1 = max_p1.getVector3fMap() - min_p1.getVector3fMap();
    whd = whd1;
    float sc1 = (whd1(0) + whd1(1) + whd1(2)) / 3;  //点云平均尺度，用于设置主方向箭头大
小

    std::cout << "width1=" << whd1(0) << endl;
    std::cout << "heght1=" << whd1(1) << endl;
    std::cout << "depth1=" << whd1(2) << endl;
    std::cout << "scale1=" << sc1 << endl;
```

```
	const Eigen::Quaternionf bboxQ1(Eigen::Quaternionf::Identity());
	const Eigen::Vector3f    bboxT1(c1);

	const Eigen::Quaternionf bboxQ(tm_inv.block<3, 3>(0, 0));
	const Eigen::Vector3f    bboxT(c);


	//变换到原点的点云主方向
	PointType op;
	op.x = 0.0;
	op.y = 0.0;
	op.z = 0.0;
	Eigen::Vector3f px, py, pz;
	Eigen::Affine3f tm_aff(tm);
	pcl::transformVector(eigenVectorsPCA.col(0), px, tm_aff);
	pcl::transformVector(eigenVectorsPCA.col(1), py, tm_aff);
	pcl::transformVector(eigenVectorsPCA.col(2), pz, tm_aff);
	PointType pcaX;
	pcaX.x = sc1 * px(0);
	pcaX.y = sc1 * px(1);
	pcaX.z = sc1 * px(2);
	PointType pcaY;
	pcaY.x = sc1 * py(0);
	pcaY.y = sc1 * py(1);
	pcaY.z = sc1 * py(2);
	PointType pcaZ;
	pcaZ.x = sc1 * pz(0);
	pcaZ.y = sc1 * pz(1);
	pcaZ.z = sc1 * pz(2);

	//visualization
	pcl::visualization::PCLVisualizer viewer;


	pcl::visualization::PointCloudColorHandlerCustom<PointType> color_handler(cloud, 255,
255, 0); //输入的初始点云相关
	viewer.addPointCloud(cloud, color_handler, "cloud");
	viewer.addCube(bboxT, bboxQ, whd(0), whd(1), whd(2), "bbox");
	viewer.setShapeRenderingProperties(pcl::visualization::PCL_VISUALIZER_REPRESENTATIO
N, pcl::visualization::PCL_VISUALIZER_REPRESENTATION_WIREFRAME, "bbox");
	viewer.setShapeRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR, 1.0,
0.0, 0.0, "bbox");
```

```
    viewer.addCoordinateSystem(0.5f*sc1);
    viewer.setBackgroundColor(0.0, 0.0, 0.0);
    while (!viewer.wasStopped())
    {
        viewer.spinOnce();
    }

    return 0;
}
```