PCL小组分享_实例代码运行_点云特征描述与提取

https://www.yuque.com/huangzhongqing/pcl/kf9kmf
https://www.cnblogs.com/li-yao7758258/p/6481668.html
讲一讲代码
跑代码展示运行效果

例1 法向量估计

C++ 复制代码

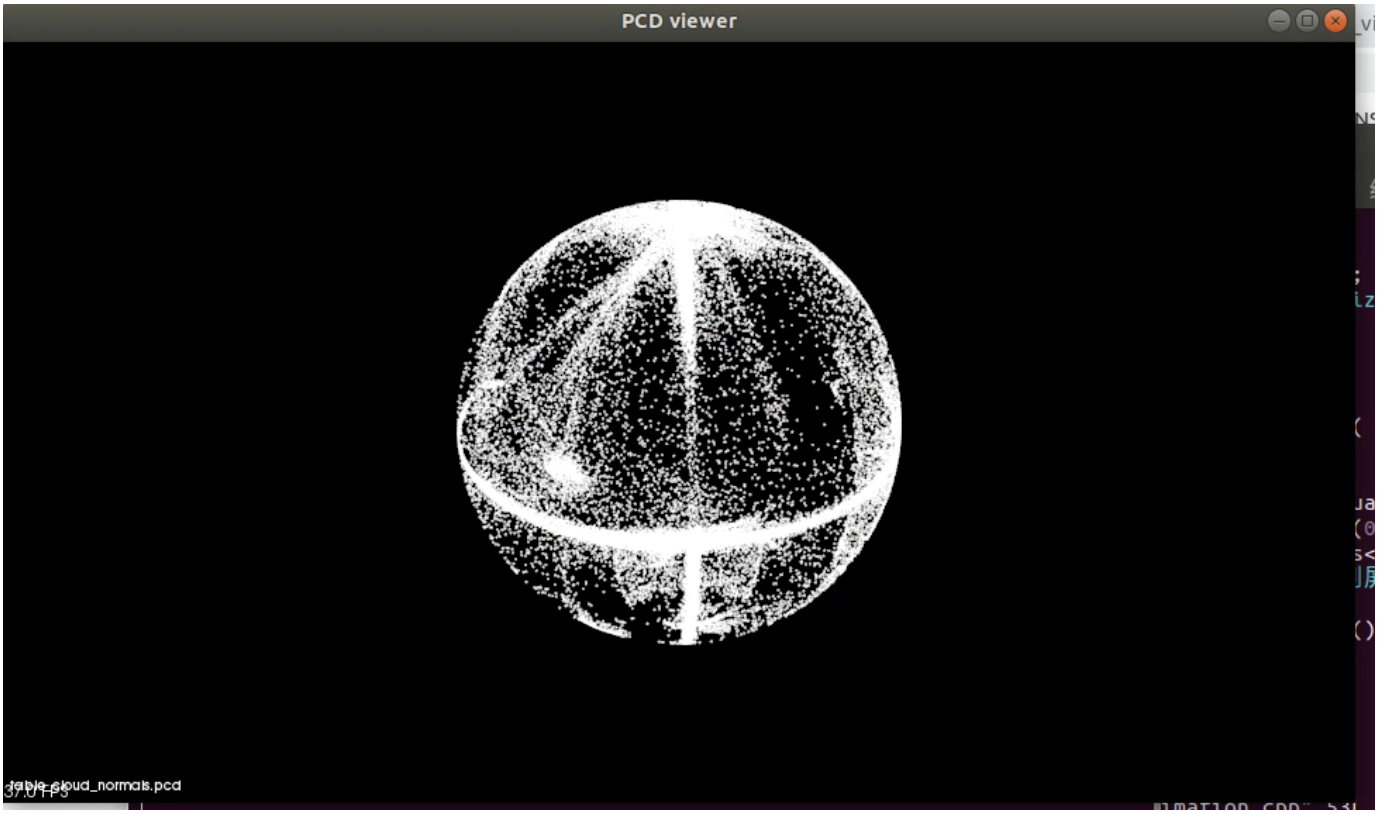
```cpp
/*
 * @Description: 法向量估计(运行耗时2min)。：https://www.cnblogs.com/li-yao7758258/p/6479255.html
 * @Author: HCQ
 * @Company(School): UCAS
 * @Date: 2020-10-19 16:33:43
 * @LastEditors: Please set LastEditors
 * @LastEditTime: 2020-10-19 14:34:00
 * @FilePath: /pcl-learning/10features特征/1法向量估计/normal_estimation.cpp
 */
#include <pcl/io/io.h>
#include <pcl/io/pcd_io.h>
#include <pcl/features/integral_image_normal.h>  //法线估计类头文件
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_types.h>
#include <pcl/features/normal_3d.h>

int
main ()
 {
//打开点云代码
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
pcl::io::loadPCDFile ("../table_scene_lms400.pcd", *cloud);

//创建法线估计估计向量
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
ne.setInputCloud(cloud);
//创建一个空的KdTree对象，并把它传递给法线估计向量
//基于给出的输入数据集，KdTree将被建立
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());
ne.setSearchMethod(tree);
//存储输出数据
pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>);
//使用半径在查询点周围3厘米范围内的所有临近元素
ne.setRadiusSearch(0.03);
//计算特征值
ne.compute(*cloud_normals);
// cloud_normals->points.size ()应该与input cloud_downsampled->points.size ()有相同的尺寸

// 存储特征值为点云
pcl::PCDWriter writer;
writer.write<pcl::Normal> ( "../table_cloud_normals.pcd" , *cloud_normals, false); // 保存文件
//可视化
```

```
43  pcl::visualization::PCLVisualizer viewer("PCL Viewer");
44  viewer.setBackgroundColor (0.0, 0.0, 0.0);
45  viewer.addPointCloudNormals<pcl::PointXYZ,pcl::Normal>(cloud, cloud_normals);
    // 将估计的点云表面法线添加到屏幕
46
47  while (!viewer.wasStopped ())
48  {
49       viewer.spinOnce ();
50  }
51
52  return 0;
53  }
```

可视化展示

运行可执行程序../normal_estimation，再打开原图二者对比

疑问1：直接pcl_viewer查看table_cloud_normals.pcd文件的时候，发现是法线球体；和代码里看到的不一样

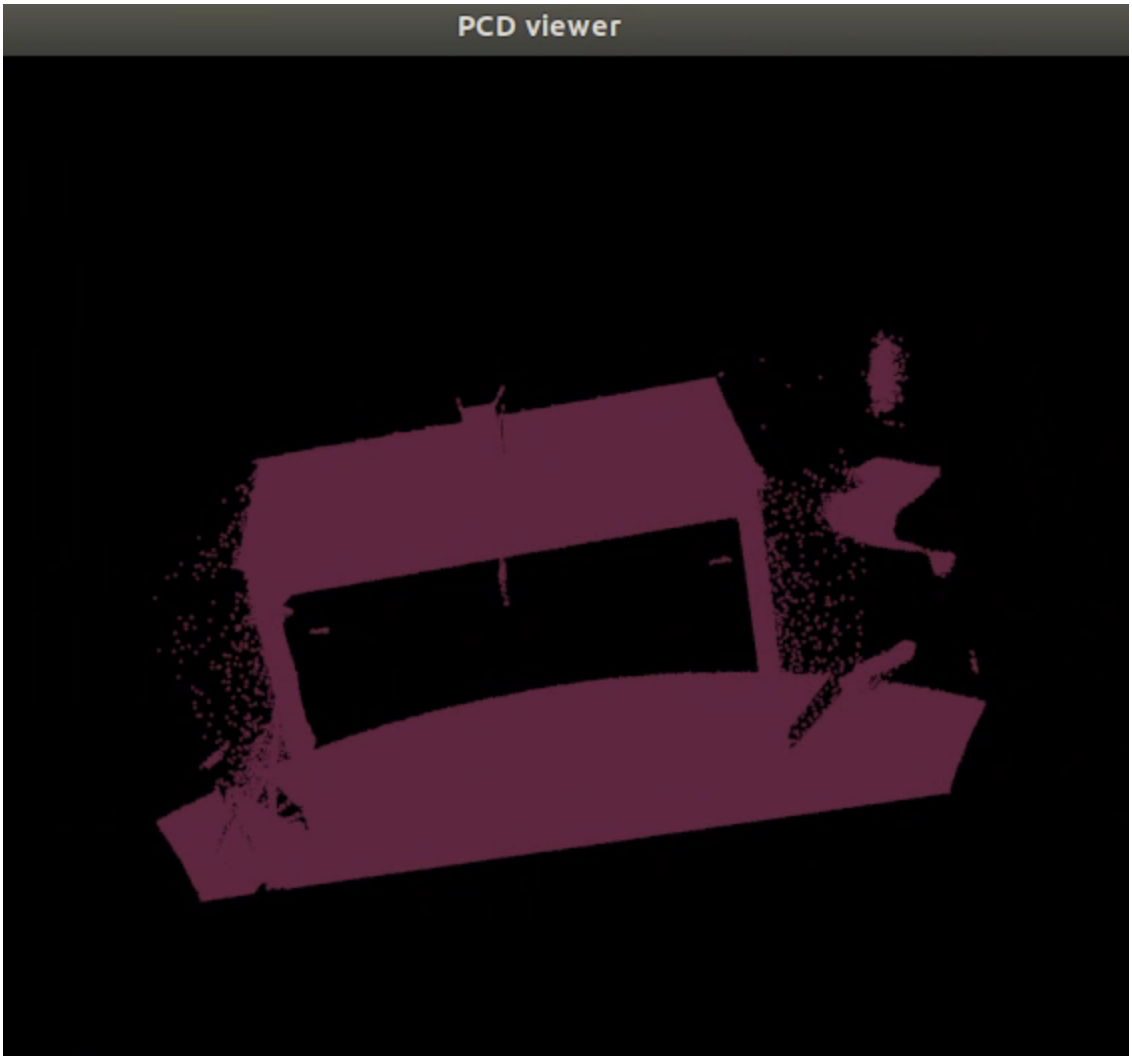


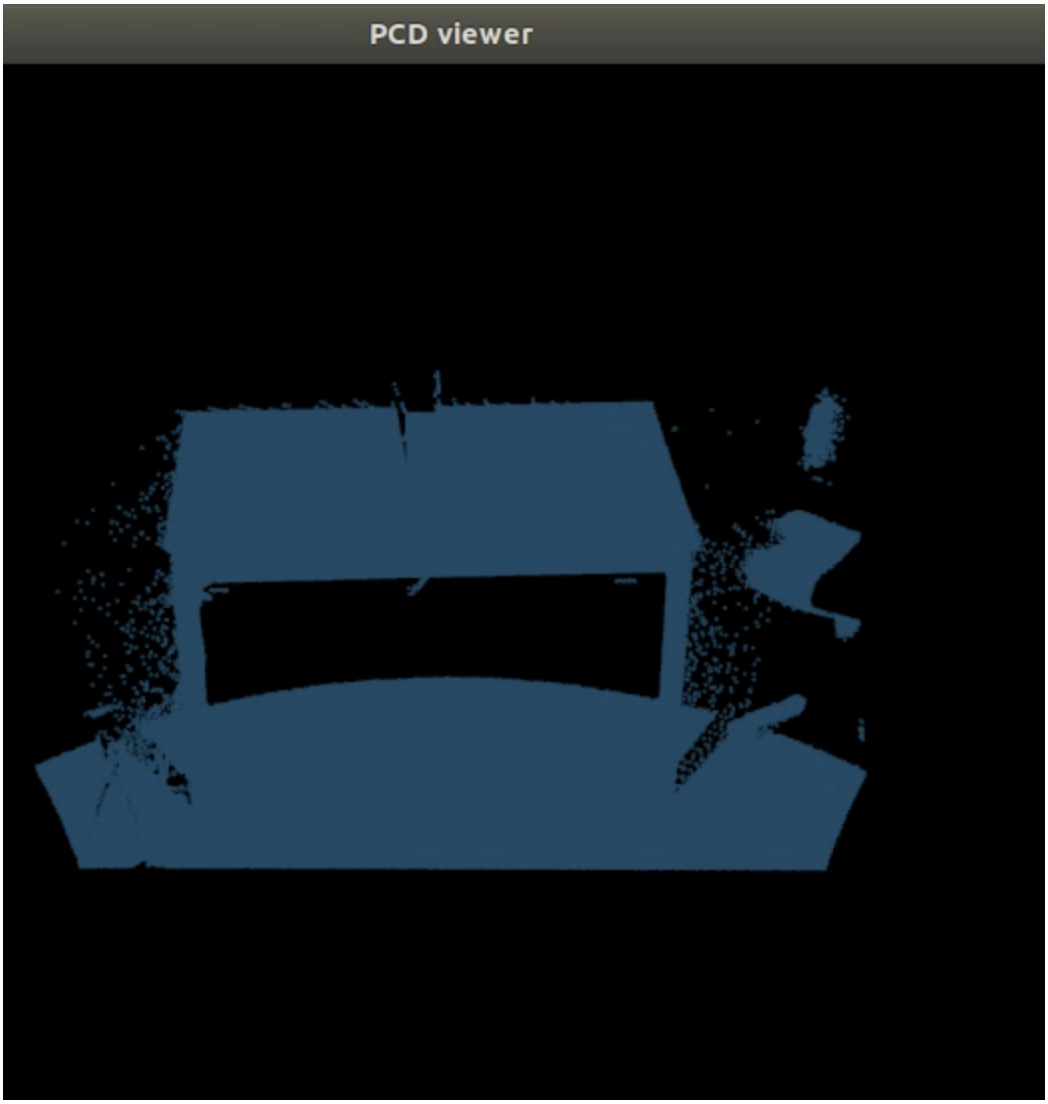

```cpp
viewer.addPointCloudNormals<pcl::PointXYZ,pcl::Normal>(cloud, cloud_normals);
// 将估计的点云表面法线添加到屏幕
```

另外发现如果注释掉这一句，结果是一片黑没有点云法线，即没添加到屏幕上；
推测是直接看是法线球体，投到屏幕上如上方的对比图

疑问2：为啥每次viewer打开颜色会随机变化？

```bash
pcl_viewer table_scene_lms400.pcd
```


PCD viewer


PCD viewer

例2 表面估计

表面法线是几何体表面一个十分重要的属性，例如：在进行光照渲染时产生符合可视习惯的效果时需要表面法线的信息才能正常进行，对于一个已经已经知道的几何体表面，根据垂直于点表面的的矢量，因此推推处表面某一点的法线方向比较容易，然而由于我们获取的点云的数据集在真实的物体的表面表现为一组定点的样本，这样就会有两种方法解决：

1. 使用曲面重建技术，从获取的点云数据中得到采样点对应的曲面，然后从曲面模型中计算出表面法线
2. 直接从点云数据中近似推断表面法线

在确定表面一点法线的问题近似于估计表面的一个相切面法线的问题，因此转换过来就是求一个最小二乘法平面拟合的问题

例3 使用积分图进行法线估计Normal Estimation Using Integral Images

三种法线估计方法

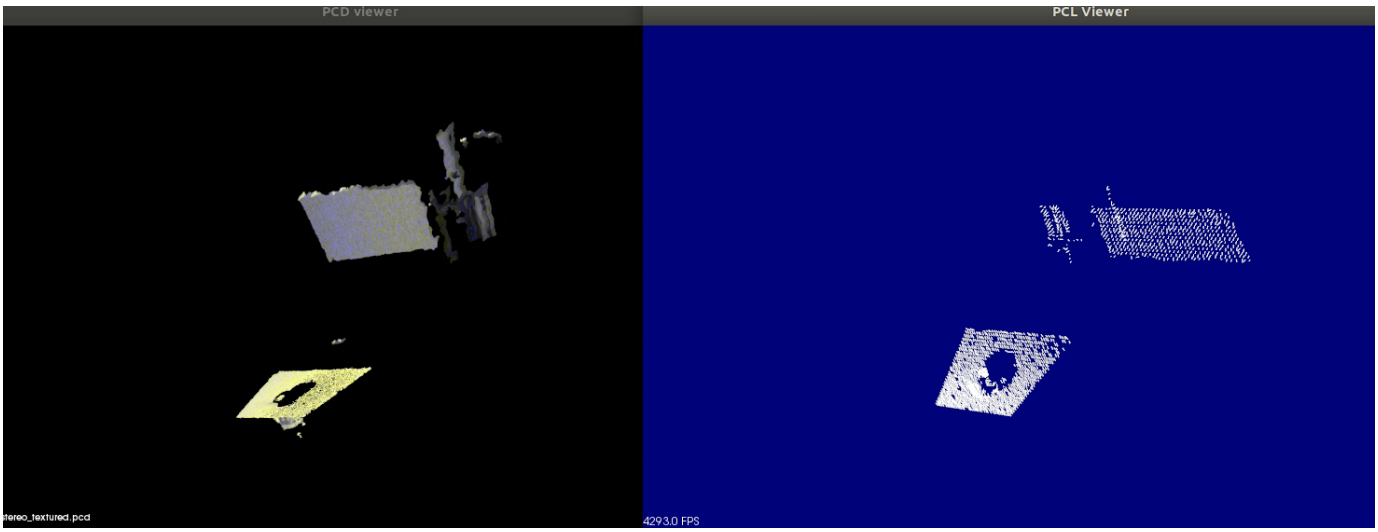
- COVARIANCE_MATRIX 模式从具体某个点的局部邻域的协方差矩阵创建9个积分，来计算这个点的法线
- AVERAGE_3D_GRADIENT　模式创建6个积分图来计算水平方向和垂直方向的平滑后的三维梯度并使用两个梯度间的向量积计算法线
- AVERAGE_DEPTH——CHANGE　模式只创建了一个单一的积分图，从而平局深度变化计算法线

C++ 复制代码

```cpp
/*
 * @Description: 为输入数据集中的点的子集估计一组表面法线。https://pcl.readthedocs.io/projects/tutorials/en/latest/how_features_work.html#how-3d-features-work
 * @Author: HCQ
 * @Company(School): UCAS
 * @Email: 1756260160@qq.com
 * @Date: 2020-10-19 15:04:50
 * @LastEditTime: 2020-10-19 18:34:17
 * @FilePath: /pcl-learning/10features特征/2估计点云子集的表面法线(error)/normal_estimation_subdset_points.cpp
 */
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/features/normal_3d.h>
// 报错已解决：下面报错 error: ‘shared_ptr’ is not a member of ‘pcl’

int main ()
 {
//打开点云代码
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
pcl::io::loadPCDFile ("../table_scene_lms400.pcd", *cloud);

  // //创建一组要使用的索引。为简单起见，我们将使用云中前10%的点
  std::vector<int> indices (std::floor (cloud->size () / 10));
  for (std::size_t i = 0; i < indices.size (); ++i) indices[i] = i;

  // //创建NormalEstimation类，并将输入数据集传递给它
  pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
  ne.setInputCloud (cloud);

  // 传递索引
  // std::shared_ptr<std::vector<int> > indicesptr (new std::vector<int> (indices)); // 报错
  pcl::shared_ptr<std::vector<int> > indicesptr (new std::vector<int> (indices)); // 解决报错
  ne.setIndices (indicesptr);

  // 创建一个空的kdtree表示形式，并将其传递给normal estimation 对象
  // 它的内容将根据给定的输入数据集填充到对象内部（因为未提供其他搜索表面）。
  pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ> ());
  ne.setSearchMethod (tree);

  // Output datasets
  pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::Normal>);
```

```
41
42      // 在半径3cm的范围内近邻
43      ne.setRadiusSearch (0.03);
44
45      // 计算特征
46      ne.compute (*cloud_normals);
47
48      // cloud_normals->size () should have the same size as the input
    indicesptr->size ()
49    }
```

可视化展示



例4 估算PFH特征

点特征直方图（PFH）在PCL中的实现是pcl_features模块的一部分。默认PFH的实现使用5个区间分类（例如：四个特征值中的每个都使用5个区间来统计），
以下代码段将对输入数据集中的所有点估计其对应的PFH特征

C++ 复制代码

```cpp
#include <pcl/point_types.h>                    //点类型头文件

#include <pcl/features/pfh.h>                   //pfh特征估计类头文件

...//其他相关操作

pcl::PointCloud<pcl::PointXYZ>::Ptrcloud(new pcl::PointCloud<pcl::PointXYZ>);

pcl::PointCloud<pcl::Normal>::Ptrnormals(new pcl::PointCloud<pcl::Normal>());

...//打开点云文件估计法线等

//创建PFH估计对象pfh，并将输入点云数据集cloud和法线normals传递给它

pcl::PFHEstimation<pcl::PointXYZ,pcl::Normal,pcl::PFHSignature125> pfh;

pfh.setInputCloud(cloud);

pfh.setInputNormals(normals);

//如果点云是类型为PointNormal,则执行pfh.setInputNormals (cloud);

//创建一个空的kd树表示法，并把它传递给PFH估计对象。

//基于已给的输入数据集，建立kdtree

pcl::KdTreeFLANN<pcl::PointXYZ>::Ptrtree(new pcl::KdTreeFLANN<pcl::PointXYZ>
());

pfh.setSearchMethod(tree);

//输出数据集

pcl::PointCloud<pcl::PFHSignature125>::Ptrpfhs(new
pcl::PointCloud<pcl::PFHSignature125>());

//使用半径在5厘米范围内的所有邻元素。

//注意：此处使用的半径必须要大于估计表面法线时使用的半径!!!

pfh.setRadiusSearch(0.05);

//计算pfh特征值

pfh.compute(*pfhs);

// pfhs->points.size ()应该与input cloud->points.size ()有相同的大小，即每个点都有
一个pfh特征向量
```

例5 估算FPFH

PFH和FPFH计算方式之间的主要区别总结如下：

1.FPFH没有对全互连 点的所有邻近点的计算参数进行统计，从图中可以看到，因此可能漏掉了一些重要的点对，而这些漏掉的对点可能对捕获查询点周围的几何特征有贡献。

2.PFH特征模型是对查询点周围的一个精确的邻域半径内，而FPFH还包括半径r范围以外的额外点对（不过在2r内）；

3.因为重新权重计算的方式，所以FPFH结合SPFH值，重新捕获邻近重要点对的几何信息；

4.由于大大地降低了FPFH的整体复杂性，因此FPFH有可能使用在实时应用中；

5.通过分解三元组，简化了合成的直方图。也就是简单生成d分离特征直方图，对每个特征维度来单独绘制，并把它们连接在一起

C++ 复制代码

```cpp
#include

#include            //fpfh特征估计类头文件声明

...//其他相关操作

pcl::PointCloud<pcl::PointXYZ>::Ptrcloud(new pcl::PointCloud<pcl::PointXYZ>);

pcl::PointCloud<pcl::Normal>::Ptrnormals(new pcl::PointCloud<pcl::Normal>());

...//打开点云文件估计法线等

//创建FPFH估计对象fpfh，并把输入数据集cloud和法线normals传递给它。

pcl::FPFHEstimation<pcl::PointXYZ,pcl::Normal,pcl::FPFHSignature33> fpfh;

fpfh.setInputCloud(cloud);

fpfh.setInputNormals(normals);

//如果点云是类型为PointNormal，则执行fpfh.setInputNormals (cloud);

//创建一个空的kd树对象tree，并把它传递给FPFH估计对象。

//基于已知的输入数据集，建立kdtree

pcl::search::KdTree<PointXYZ>::Ptrtree(new pcl::search::KdTree<PointXYZ>);

fpfh.setSearchMethod(tree);

//输出数据集

pcl::PointCloud<pcl::FPFHSignature33>::Ptrfpfhs(new
pcl::PointCloud<pcl::FPFHSignature33>());

//使用所有半径在5厘米范围内的邻元素

//注意：此处使用的半径必须要大于估计表面法线时使用的半径!!!

fpfh.setRadiusSearch(0.05);

//计算获取特征向量

fpfh.compute(*fpfhs);

// fpfhs->points.size ()应该和input cloud->points.size ()有相同的大小，即每个点有一
个特征向量
```

例6 估算VFH

视点特征直方图（或VFH）是源于FPFH描述子。由于它的获取速度和识别力，我们决定利用FPFH强大的识别力，但是为了使构造的特征保持缩放不变性的性质同时，还要区分不同的位姿，计算时需要考虑加入视点变量。我们做了以下两种计算来构造特征，以应用于目标识别问题和位姿估计：1.扩展FPFH，使其利用整个点云对象来进行计算估计（如2图所示），在计算FPFH时以物体中心点与物体表面其他所有点之间的点对作为计算单元。2.添加视点方向与每个点估计法线之间额外的统计信息，为了达到这个目的，我们的关键想法是在FPFH计算中将视点方向变量直接融入到相对法线角计算当中。

通过统计视点方向与每个法线之间角度的直方图来计算视点相关的特征分量。注意：并不是每条法线的视角，因为法线的视角在尺度变换下具有可变性，我们指的是平移视点到查询点后的视点方向和每条法线间的角度。第二组特征分量就是前面PFH中讲述的三个角度，如PFH小节所述，只是现在测量的是在中心点的视点方向和每条表面法线之间的角度

因此新组合的特征被称为视点特征直方图（VFH）。下图表体现的就是新特征的想法，包含了以下两部分：

1.一个视点方向相关的分量

2.一个包含扩展FPFH的描述表面形状的分量

C++ 复制代码

```cpp
#include <pcl/point_types.h>

#include <pcl/features/vfh.h>                    //VFH特征估计类头文件

...//其他相关操作

  pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);

  pcl::PointCloud<pcl::Normal>::Ptr normals (new pcl::PointCloud<pcl::Normal>());

...//打开点云文件估计法线等

//创建VFH估计对象vfh，并把输入数据集cloud和法线normal传递给它

  pcl::VFHEstimation<pcl::PointXYZ, pcl::Normal, pcl::VFHSignature308> vfh;

  vfh.setInputCloud (cloud);

  vfh.setInputNormals (normals);

//如果点云是PointNormal类型，则执行vfh.setInputNormals (cloud);

//创建一个空的kd树对象，并把它传递给FPFH估计对象。

//基于已知的输入数据集，建立kdtree

pcl::KdTreeFLANN<pcl::PointXYZ>::Ptr tree (new pcl::KdTreeFLANN<pcl::PointXYZ> ());

  vfh.setSearchMethod (tree);

//输出数据集

  pcl::PointCloud<pcl::VFHSignature308>::Ptr vfhs (new pcl::PointCloud<pcl::VFHSignature308> ());

//计算特征值

  vfh.compute (*vfhs);

// vfhs->points.size ()的大小应该是1，即vfh描述子是针对全局的特征描述
```

例7 完整的算法代码例子

如何从一个深度图像（range image）中提取NARF特征

代码解析narf_feature_extraction.cpp

C++ 复制代码

```cpp
#include <iostream>

#include <boost/thread/thread.hpp>
#include <pcl/range_image/range_image.h>
#include <pcl/io/pcd_io.h>
#include <pcl/visualization/range_image_visualizer.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/features/range_image_border_extractor.h>
#include <pcl/keypoints/narf_keypoint.h>
#include <pcl/features/narf_descriptor.h>
#include <pcl/console/parse.h>

typedef pcl::PointXYZ PointType;

//参数的设置
float angular_resolution = 0.5f;
float support_size = 0.2f;
pcl::RangeImage::CoordinateFrame coordinate_frame =
pcl::RangeImage::CAMERA_FRAME;
bool setUnseenToMaxRange = false;
bool rotation_invariant = true;

//命令帮助
void
printUsage (const char* progName)
{
  std::cout << "\n\nUsage: "<<progName<<" [options] <scene.pcd>\n\n"
            << "Options:\n"
            << "-------------------------------------------\n"
            << "-r <float>   angular resolution in degrees (default "
<<angular_resolution<<")\n"
            << "-c <int>     coordinate frame (default "<<
(int)coordinate_frame<<")\n"
            << "-m           Treat all unseen points to max range\n"
            << "-s <float>   support size for the interest points (diameter
of the used sphere - "
                                                          "default "
<<support_size<<")\n"
            << "-o <0/1>     switch rotational invariant version of the
feature on/off"
            <<               " (default "<< (int)rotation_invariant<<")\n"
            << "-h           this help\n"
            << "\n\n";
}

void
setViewerPose (pcl::visualization::PCLVisualizer& viewer, const
Eigen::Affine3f& viewer_pose)//setViewerPose
```

```
{
  Eigen::Vector3f pos_vector = viewer_pose * Eigen::Vector3f (0, 0, 0);
  Eigen::Vector3f look_at_vector = viewer_pose.rotation () * Eigen::Vector3f
(0, 0, 1) + pos_vector;
  Eigen::Vector3f up_vector = viewer_pose.rotation () * Eigen::Vector3f (0,
-1, 0);
  viewer.setCameraPosition (pos_vector[0], pos_vector[1], pos_vector[2],
                            look_at_vector[0], look_at_vector[1],
look_at_vector[2],
                            up_vector[0], up_vector[1], up_vector[2]);
}

int
main (int argc, char** argv)
{
 // 设置参数检测
  if (pcl::console::find_argument (argc, argv, "-h") >= 0)
  {
    printUsage (argv[0]);
    return 0;
  }
  if (pcl::console::find_argument (argc, argv, "-m") >= 0)
  {
    setUnseenToMaxRange = true;
    cout << "Setting unseen values in range image to maximum range
readings.\n";
  }
  if (pcl::console::parse (argc, argv, "-o", rotation_invariant) >= 0)
    cout << "Switching rotation invariant feature version "<<
(rotation_invariant ? "on" : "off")<<".\n";
  int tmp_coordinate_frame;
  if (pcl::console::parse (argc, argv, "-c", tmp_coordinate_frame) >= 0)
  {
    coordinate_frame = pcl::RangeImage::CoordinateFrame
(tmp_coordinate_frame);
    cout << "Using coordinate frame "<< (int)coordinate_frame<<".\n";
  }
  if (pcl::console::parse (argc, argv, "-s", support_size) >= 0)
    cout << "Setting support size to "<<support_size<<".\n";
  if (pcl::console::parse (argc, argv, "-r", angular_resolution) >= 0)
    cout << "Setting angular resolution to "<<angular_resolution<<"deg.\n";
  angular_resolution = pcl::deg2rad (angular_resolution);

//打开一个磁盘中的.pcd文件  但是如果没有指定就会自动生成
  pcl::PointCloud<PointType>::Ptr    point_cloud_ptr (new
pcl::PointCloud<PointType>);
  pcl::PointCloud<PointType>& point_cloud = *point_cloud_ptr;

  pcl::PointCloud<pcl::PointWithViewpoint> far_ranges;
  Eigen::Affine3f scene_sensor_pose (Eigen::Affine3f::Identity ());
```

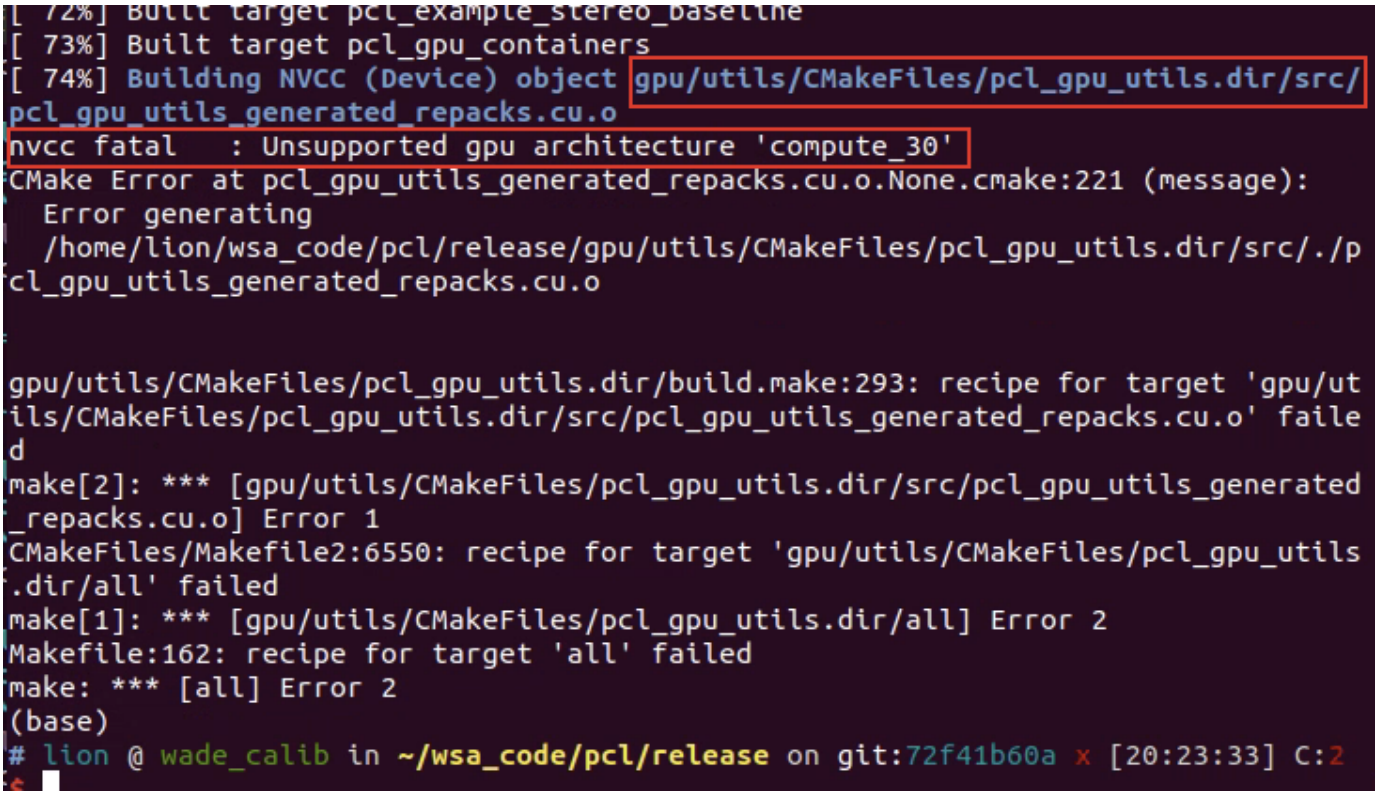
```
85    std::vector<int> pcd_filename_indices =
      pcl::console::parse_file_extension_argument (argc, argv, "pcd");
86    if (!pcd_filename_indices.empty ())   //检测是否有far_ranges.pcd
87    {
88      std::string filename = argv[pcd_filename_indices[0]];
89      if (pcl::io::loadPCDFile (filename, point_cloud) == -1)
90      {
91        cerr << "Was not able to open file \""<<filename<<"\".\n";
92        printUsage (argv[0]);
93        return 0;
94      }
95      scene_sensor_pose = Eigen::Affine3f (Eigen::Translation3f
      (point_cloud.sensor_origin_[0],
96
      point_cloud.sensor_origin_[1],
97
      point_cloud.sensor_origin_[2])) *
98                            Eigen::Affine3f (point_cloud.sensor_orientation_);
99      std::string far_ranges_filename = pcl::getFilenameWithoutExtension
      (filename)+"_far_ranges.pcd";
100     if (pcl::io::loadPCDFile (far_ranges_filename.c_str (), far_ranges) ==
      -1)
101       std::cout << "Far ranges file \""<<far_ranges_filename<<"\" does not
      exists.\n";
102   }
103   else
104   {
105     setUnseenToMaxRange = true;
106     cout << "\nNo *.pcd file given => Genarating example point cloud.\n\n";
107     for (float x=-0.5f; x<=0.5f; x+=0.01f)   //如果没有打开的文件就生成一个矩形的点
      云
108     {
109       for (float y=-0.5f; y<=0.5f; y+=0.01f)
110       {
111         PointType point;  point.x = x;  point.y = y;  point.z = 2.0f - y;
112         point_cloud.points.push_back (point);
113       }
114     }
115     point_cloud.width = (int) point_cloud.points.size ();  point_cloud.height
      = 1;
116   }
117
118   //从点云中建立生成深度图
119   float noise_level = 0.0;
120   float min_range = 0.0f;
121   int border_size = 1;
122   boost::shared_ptr<pcl::RangeImage> range_image_ptr (new pcl::RangeImage);
123   pcl::RangeImage& range_image = *range_image_ptr;
124   range_image.createFromPointCloud (point_cloud, angular_resolution,
      pcl::deg2rad (360.0f), pcl::deg2rad (180.0f),
```

```
                                    scene_sensor_pose, coordinate_frame,
noise_level, min_range, border_size);
  range_image.integrateFarRanges (far_ranges);
  if (setUnseenToMaxRange)
    range_image.setUnseenToMaxRange ();

  //打开3D viewer并加入点云
  pcl::visualization::PCLVisualizer viewer ("3D Viewer");
  viewer.setBackgroundColor (1, 1, 1);
  pcl::visualization::PointCloudColorHandlerCustom<pcl::PointWithRange>
range_image_color_handler (range_image_ptr, 0, 0, 0);
  viewer.addPointCloud (range_image_ptr, range_image_color_handler, "range
image");
  viewer.setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 1, "range image");
  //viewer.addCoordinateSystem (1.0f, "global");
  //PointCloudColorHandlerCustom<PointType> point_cloud_color_handler
(point_cloud_ptr, 150, 150, 150);
  //viewer.addPointCloud (point_cloud_ptr, point_cloud_color_handler,
"original point cloud");
  viewer.initCameraParameters ();
  setViewerPose (viewer, range_image.getTransformationToWorldSystem ());
  //显示
  pcl::visualization::RangeImageVisualizer range_image_widget ("Range
image");
  range_image_widget.showRangeImage (range_image);

  //提取NARF特征
  pcl::RangeImageBorderExtractor range_image_border_extractor;    //申明深度图
边缘提取器
  pcl::NarfKeypoint narf_keypoint_detector;
//narf_keypoint_detector为点云对象

  narf_keypoint_detector.setRangeImageBorderExtractor
(&range_image_border_extractor);
  narf_keypoint_detector.setRangeImage (&range_image);
  narf_keypoint_detector.getParameters ().support_size = support_size;    //获
得特征提取的大小

  pcl::PointCloud<int> keypoint_indices;
  narf_keypoint_detector.compute (keypoint_indices);
  std::cout << "Found "<<keypoint_indices.points.size ()<<" key points.\n";

  // ----------------------------------------------
  // -----Show keypoints in range image widget-----
  // ----------------------------------------------
  //for (size_t i=0; i<keypoint_indices.points.size (); ++i)
    //range_image_widget.markPoint
(keypoint_indices.points[i]%range_image.width,

```

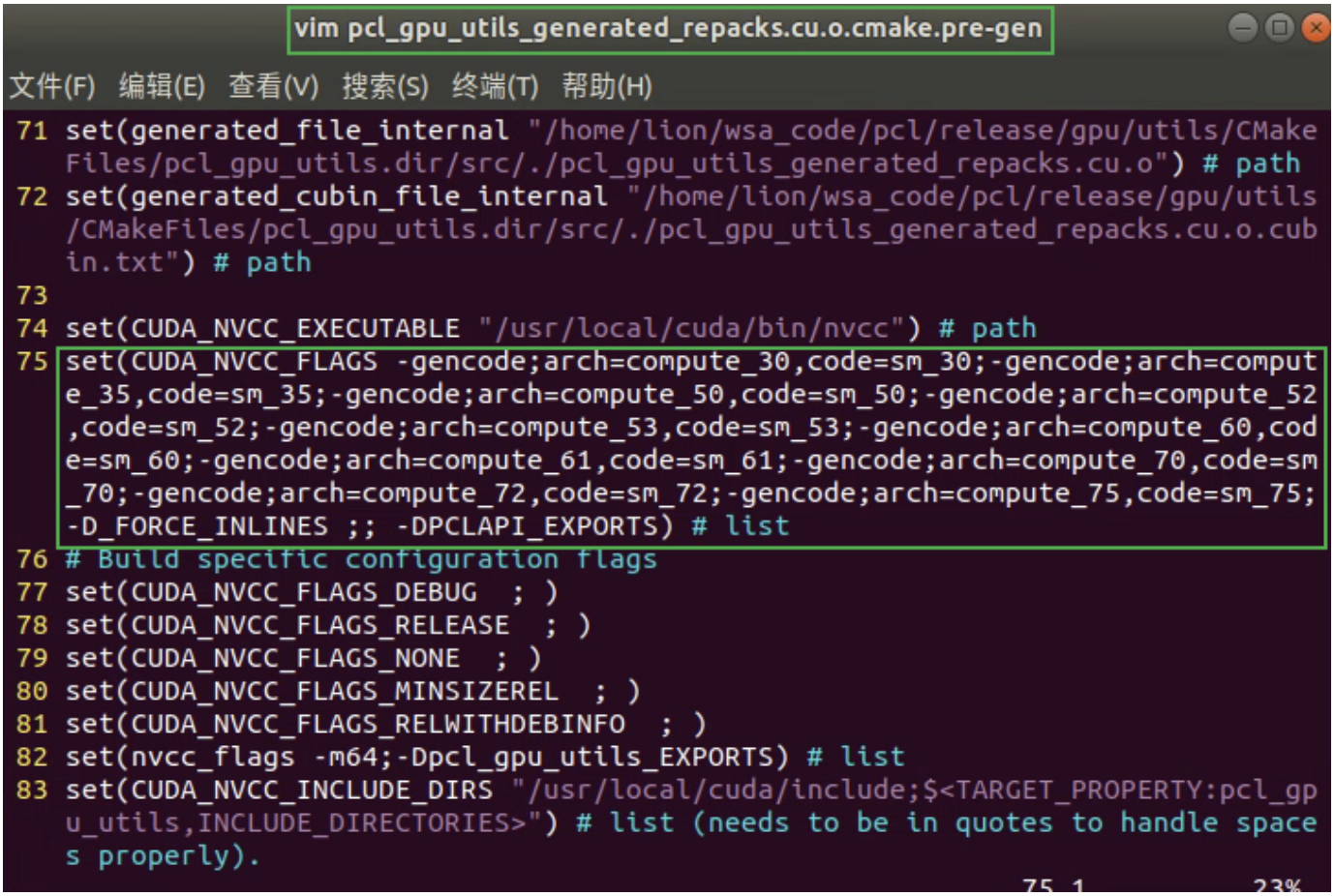
```
    //keypoint_indices.points[i]/range_image.width);
163
164   //在3Dviewer显示提取的特征信息
165   pcl::PointCloud<pcl::PointXYZ>::Ptr keypoints_ptr (new
    pcl::PointCloud<pcl::PointXYZ>);
166   pcl::PointCloud<pcl::PointXYZ>& keypoints = *keypoints_ptr;
167   keypoints.points.resize (keypoint_indices.points.size ());
168   for (size_t i=0; i<keypoint_indices.points.size (); ++i)
169     keypoints.points[i].getVector3fMap () =
    range_image.points[keypoint_indices.points[i]].getVector3fMap ();
170   pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZ>
    keypoints_color_handler (keypoints_ptr, 0, 255, 0);
171   viewer.addPointCloud<pcl::PointXYZ> (keypoints_ptr,
    keypoints_color_handler, "keypoints");
172   viewer.setPointCloudRenderingProperties
    (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 7, "keypoints");
173
174   //在关键点提取NARF描述子
175   std::vector<int> keypoint_indices2;
176   keypoint_indices2.resize (keypoint_indices.points.size ());
177   for (unsigned int i=0; i<keypoint_indices.size (); ++i) // This step is
    necessary to get the right vector type
178     keypoint_indices2[i]=keypoint_indices.points[i];      ///建立NARF关键点的索
    引向量，此矢量作为NARF特征计算的输入来使用
179
180   pcl::NarfDescriptor narf_descriptor (&range_image, &keypoint_indices2);//创
    建narf_descriptor对象。并给了此对象输入数据（特征点索引和深度像）
181   narf_descriptor.getParameters ().support_size = support_size;//support_size
    确定计算描述子时考虑的区域大小
182   narf_descriptor.getParameters ().rotation_invariant = rotation_invariant;
    //设置旋转不变的NARF描述子
183   pcl::PointCloud<pcl::Narf36> narf_descriptors;              //创建Narf36的点
    类型输入点云对象并进行实际计算
184   narf_descriptor.compute (narf_descriptors);                 //计算描述子
185   cout << "Extracted "<<narf_descriptors.size ()<<" descriptors for "   //打印
    输出特征点的数目和提取描述子的数目
186                        <<keypoint_indices.points.size ()<< " keypoints.\n";
187
188 //主循环函数
189   while (!viewer.wasStopped ())
190   {
191     range_image_widget.spinOnce ();  // process GUI events
192     viewer.spinOnce ();
193     pcl_sleep(0.01);
194   }
195 }
```

缺点函数库，分享者没跑起来

补充：CUDA11.0下编译PCL报错'compute_30'



主要报错是红框里的这个，网上查了一下是cuda11.0开始淘汰了'compute_30'这种写法



想的方式是，在这个路径下，vim查找'compute_30'，注销掉带来问题的的代码，然后运行
尝试了几次发现，明明注释了，但是make编译后还是报错，这里又会回来

所以想解决需要在makefile里找到问题，然后修改
暂时还没找到，准备进一步尝试
如有想法可以讨论

目前用的ros里带的pcl