5 PCL可视化

pcl_visualization包含27个类以及十多个函数，依赖库有pcl_common、pcl_range_image、pcl_kdtree、pcl_IO模块以及VTK外部开源可视化库。主要包含的内容有以下几个方面：

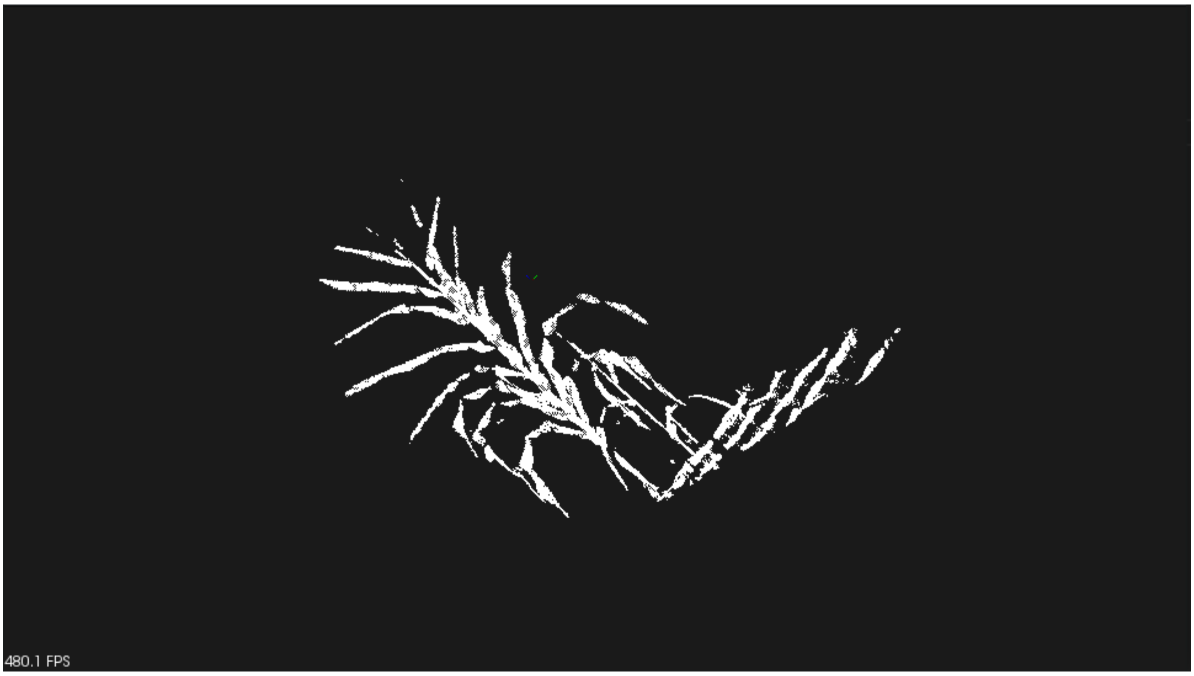
- 可视化`pcl::PointCloud<T>`格式的点云数据，设置 **颜色、尺寸、透明度**等；
- 利用点集合或者参数方程绘画基本的3D模型（圆柱、圆锥、线、多边形等）；
- 直方图可视化模块，画2D图(PCLHistogramVisualizer)；
- 对`pcl::PointCloud<T>`点云数据进行各种几何变换和颜色处理；
- `pcl::RangeImage`可视化模块

5.1 简单点云可视化

例 1 显示点云

- 代码显示

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

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

    if (pcl::io::loadPCDFile<pcl::PointXYZ>("../maize.pcd", *cloud) == -1) {
        PCL_ERROR("Couldn't read file maize.pcd\n");
        return(-1);
    }
    std::cout << cloud->points.size() << std::endl;
    pcl::visualization::CloudViewer viewer("cloud viewer");
    viewer.showCloud(cloud);
    while (!viewer.wasStopped()) {

    }
    return 0;
}
```

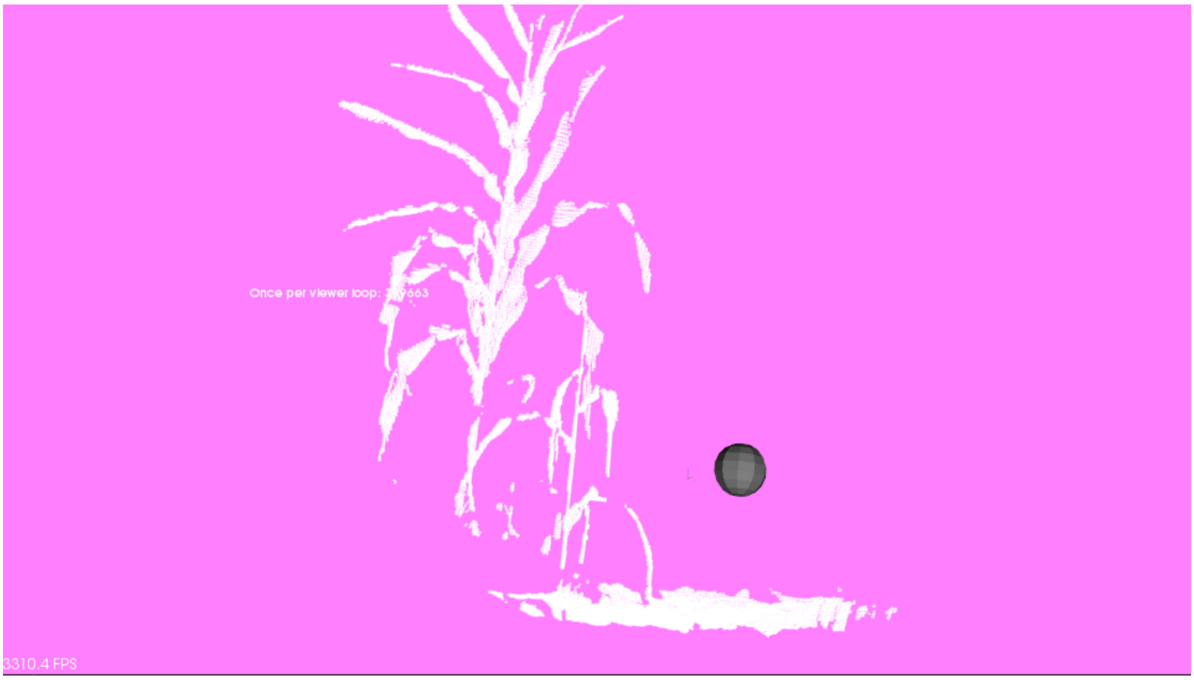


- 工具显示

```
sudo apt-get install pcl-tools  # 安装pcl_tools
pcl_viewer maize.pcd
```

例 2 viewer的简单用法

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

int user_data;
void
viewerOneOff (pcl::visualization::PCLVisualizer& viewer)
{
    viewer.setBackgroundColor (1.0, 0.5, 1.0); #设置背景颜色
    pcl::PointXYZ o; #确定圆心坐标
    o.x = 1.0;
    o.y = 0;
    o.z = 0;
    viewer.addSphere (o, 0.25, "sphere", 0); #添加几何对象 半径0.25 窗口id = 0
    std::cout << "i only run once" << std::endl;

}

void
viewerPsycho (pcl::visualization::PCLVisualizer& viewer)
{
    static unsigned count = 0;
    std::stringstream ss;
    ss << "Once per viewer loop: " << count++;
    viewer.removeShape ("text", 0);
    viewer.addText (ss.str(), 200, 300, "text", 0);
    //FIXME: possible race condition here:
    user_data++;
}

int
main ()
{
    pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud (new
pcl::PointCloud<pcl::PointXYZRGBA>);
    pcl::io::loadPCDFile ("my_point_cloud.pcd", *cloud);
    pcl::visualization::CloudViewer viewer("Cloud Viewer");
    //showCloud函数是同步的，在此处等待直到渲染显示为止
    viewer.showCloud(cloud);
    //该注册函数在可视化时只调用一次 传入参数以一个函数引用boost::ref(x)
    viewer.runOnVisualizationThreadOnce (viewerOneOff);
    //该注册函数在渲染输出时每次都调用
    viewer.runOnVisualizationThread (viewerPsycho);
    while (!viewer.wasStopped ())
    {
    //在此处可以添加其他处理
    user_data++;
    }
    return 0;
}
```



5.2 视化深度图

两种可视化方法：

- 3D视窗中以点云形式可视化
- 将深度值映射为不同的颜色，以彩色图方式可视化深度图像

代码修改执行说明

- 代码修改说明：源代码中`pcl::visualization::PCLVisualizer`的成员变量`camera_`在pcl1.8版本被移除，更改为`pcl::visualization::Camera`对象，此处做了适当的修改代码见下方；
- 代码终端执行

```
Usage: ./range_image_visualization [options] <scene.pcd>

Options:
-------------------------------------------
-r <float>    angular resolution in degrees (default 0.00872665)
-c <int>      coordinate frame (default 0)
-l            live update - update the range image according to the selected
view in the 3D viewer.
-h            this help
```

```
./range_image_visualization -l ../room_scan1.pcd
```

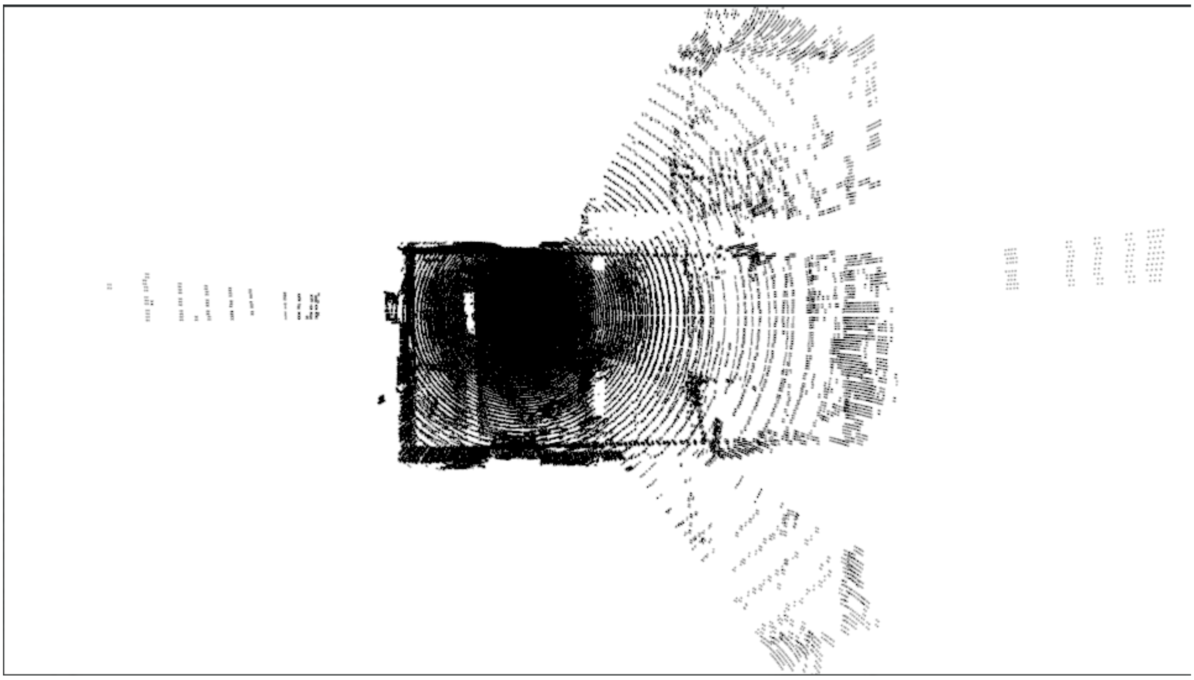
代码解读

```
#include <iostream>
#include <boost/thread/thread.hpp>
#include <pcl/common/common_headers.h>
#include <pcl/common/common_headers.h>
#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>
```

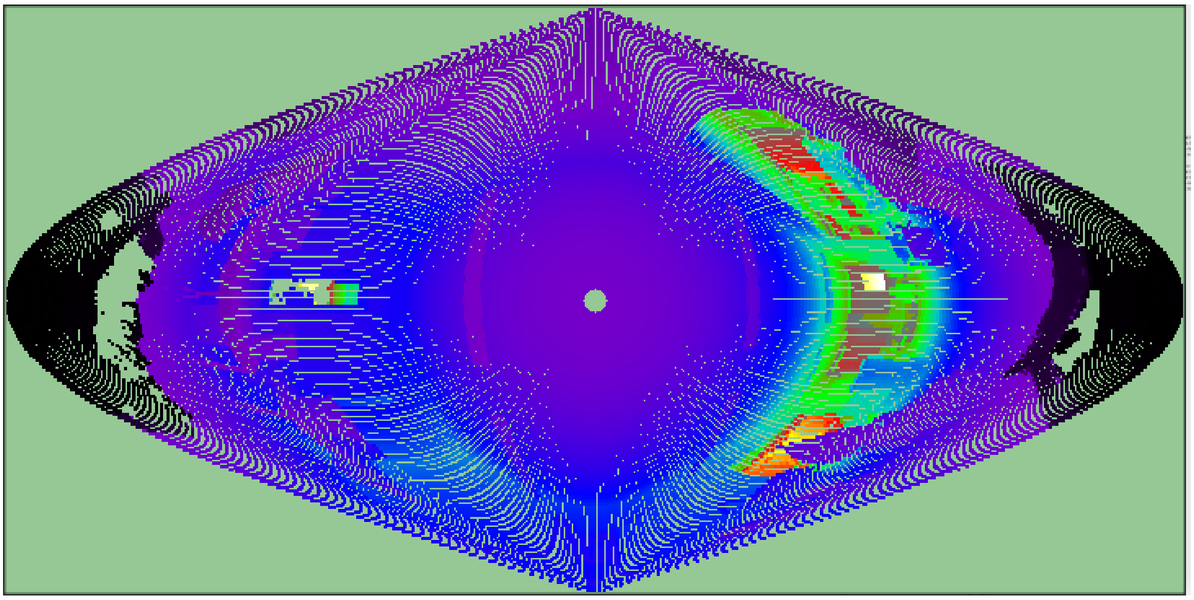
```
#include <pcl/console/parse.h>

typedef pcl::PointXYZ PointType;
// 全局参数
float angular_resolution = 0.5f;//angular_resolution为模拟的深度传感器的角度分辨率，即深度图像中一个像素对应的角度大小
pcl::RangeImage::CoordinateFrame coordinate_frame = pcl::RangeImage::CAMERA_FRAME;//深度图像遵循坐标系统
bool live_update = false;
// -----打印帮助-----
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"
            << "-l           live update - update the range image according to the selected view in the 3D viewer.\n"
            << "-h           this help\n"
            << "\n\n";
}

void
setViewerPose (pcl::visualization::PCLVisualizer viewer, pcl::visualization::Camera cam, const Eigen::Affine3f& viewer_pose)
{
  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);

  // https://www.cnblogs.com/v-weiwang/p/6072235.html
  cam.pos[0] = pos_vector[0]; //相机的位置
  cam.pos[1] = pos_vector[1];
  cam.pos[2] = pos_vector[2];
  cam.focal[0] = look_at_vector[0]; //看向那个坐标
  cam.focal[1] = look_at_vector[1];
  cam.focal[2] = look_at_vector[2];
  cam.view[0] = up_vector[0]; // 以哪个坐标轴为上方
  cam.view[1] = up_vector[1];
  cam.view[2] = up_vector[2];
  viewer.updateCamera();
}

// -----Main-----
int
main (int argc, char** argv)
{
  //解析命令行参数
  if (pcl::console::find_argument (argc, argv, "-h") >= 0)
  {
    printUsage (argv[0]);
    return 0;
  }
  if (pcl::console::find_argument (argc, argv, "-l") >= 0)
  {
    live_update = true;
    std::cout << "Live update is on.\n";
  }
  if (pcl::console::parse (argc, argv, "-r", angular_resolution) >= 0)
    std::cout << "Setting angular resolution to "<<angular_resolution<<"deg.\n";
  int tmp_coordinate_frame;
  if (pcl::console::parse (argc, argv, "-c", tmp_coordinate_frame) >= 0) //c = 0
点云呈方阵排列 CAMERA_FRAME， c = 1 点云圆环形状排列 LASER_FRAME
  {
    coordinate_frame = pcl::RangeImage::CoordinateFrame (tmp_coordinate_frame);
    std::cout << "Using coordinate frame "<< (int)coordinate_frame<<".\n";
  }
  angular_resolution = pcl::deg2rad (angular_resolution);

    // 读取给定的pcd点云文件或者自行创建随机点云
  pcl::PointCloud<PointType>::Ptr point_cloud_ptr (new
pcl::PointCloud<PointType>);
  pcl::PointCloud<PointType>& point_cloud = *point_cloud_ptr;
  Eigen::Affine3f scene_sensor_pose (Eigen::Affine3f::Identity ());
  std::vector<int> pcd_filename_indices =
pcl::console::parse_file_extension_argument (argc, argv, "pcd");//返回含有扩展名为
pcd的所有文件的索引
  if (!pcd_filename_indices.empty ())
  {
    std::string filename = argv[pcd_filename_indices[0]];
    if (pcl::io::loadPCDFile (filename, point_cloud) == -1)
    {
      std::cout << "Was not able to open file \""<<filename<<"\".\n";
      printUsage (argv[0]);
      return 0;
    }
    scene_sensor_pose = Eigen::Affine3f (Eigen::Translation3f
(point_cloud.sensor_origin_[0],

point_cloud.sensor_origin_[1],

point_cloud.sensor_origin_[2])) *
                        Eigen::Affine3f (point_cloud.sensor_orientation_); // 该
部分Affine3f 与 *都是重载 得到结果就是点云传感器的位姿
  }
  else //如果没有点云文件 那就自己创建一个
  {
    std::cout << "\nNo *.pcd file given => Genarating example point cloud.\n\n";
    for (float x=-0.5f; x<=0.5f; x+=0.01f)
    {
      for (float y=-0.5f; y<=0.5f; y+=0.01f)
      {
        PointType point;  point.x = x;  point.y = y;  point.z = 2.0f - y;
        point_cloud.points.push_back (point);
      }
    }
    point_cloud.width = (int) point_cloud.points.size ();  point_cloud.height =
1;
  }
  //从点云创建深度图像对象
  float noise_level = 0.0;
  float min_range = 0.0f;
  int border_size = 1;
  boost::shared_ptr<pcl::RangeImage> range_image_ptr(new pcl::RangeImage); //新建一个三维场景的深度图像指针对象
  pcl::RangeImage& range_image = *range_image_ptr;
   /*
   关于range_image.createFromPointCloud（）参数的解释 （涉及的角度都为弧度为单位） :
   point_cloud为创建深度图像所需要的点云
  angular_resolution_x深度传感器X方向的角度分辨率
  angular_resolution_y深度传感器Y方向的角度分辨率
   pcl::deg2rad (360.0f)深度传感器的水平最大采样角度
   pcl::deg2rad (180.0f)垂直最大采样角度
   scene_sensor_pose设置的模拟传感器的位姿是一个仿射变换矩阵，默认为4*4的单位矩阵变换
   coordinate_frame定义按照那种坐标系统的习惯  默认为CAMERA_FRAME
   noise_level  获取深度图像深度时，邻近点对查询点距离值的影响水平
   min_range 设置最小的获取距离，小于最小的获取距离的位置为传感器的盲区
   border_size  设置获取深度图像边缘的宽度 默认为0
   */
  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);
  //创建3D视图并且添加点云进行显示
  pcl::visualization::PCLVisualizer viewer ("3D Viewer");
  pcl::visualization::Camera cam;

  viewer.setBackgroundColor (1, 1, 1); //背景颜色白色
  pcl::visualization::PointCloudColorHandlerCustom<pcl::PointWithRange> range_image_color_handler (range_image_ptr, 0, 0, 0); //点云颜色 黑色
  viewer.addPointCloud (range_image_ptr, range_image_color_handler, "range image");
  viewer.setPointCloudRenderingProperties (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 1, "range image");//点云大小为1的深度图像
  //viewer.addCoordinateSystem (1.0f);
  //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 (); //初始化相机参数
  setViewerPose(viewer, cam, range_image.getTransformationToWorldSystem ());
  //显示深度图像 颜色取决于深度值
  pcl::visualization::RangeImageVisualizer range_image_widget ("Range image");
  range_image_widget.showRangeImage (range_image);
    //主循环
  while (!viewer.wasStopped ())
  {
      //两个可视化窗口
    range_image_widget.spinOnce (); //处理可视化深度图像的当前事件
    viewer.spinOnce (); //处理可视化3D窗口的当前事件
    pcl_sleep (0.01);
     //通过命令行 -l随时更新2D深度图
    if (live_update)
    {
      scene_sensor_pose = viewer.getViewerPose();
      range_image.createFromPointCloud (point_cloud, angular_resolution, pcl::deg2rad (360.0f), pcl::deg2rad (180.0f), scene_sensor_pose, pcl::RangeImage::LASER_FRAME, noise_level, min_range, border_size);
```

```
        range_image_widget.showRangeImage (range_image);
      }
    }
  }
```

无深度值颜色的3D点云视图：



根据深图值赋予3D点云不同的颜色视图：



5.3 PCLVisualizer 可视化类

简单介绍

对于3D视窗的操作：

- Q键退出视窗应用窗口
- R键居中并缩放整个点云
- 使用鼠标左键单击或拖动旋转窗口
- 鼠标滚轮 或 鼠标右键上下拖动，实现放大缩小

- 滚轮单击或拖动将会移动视窗

代码执行说明

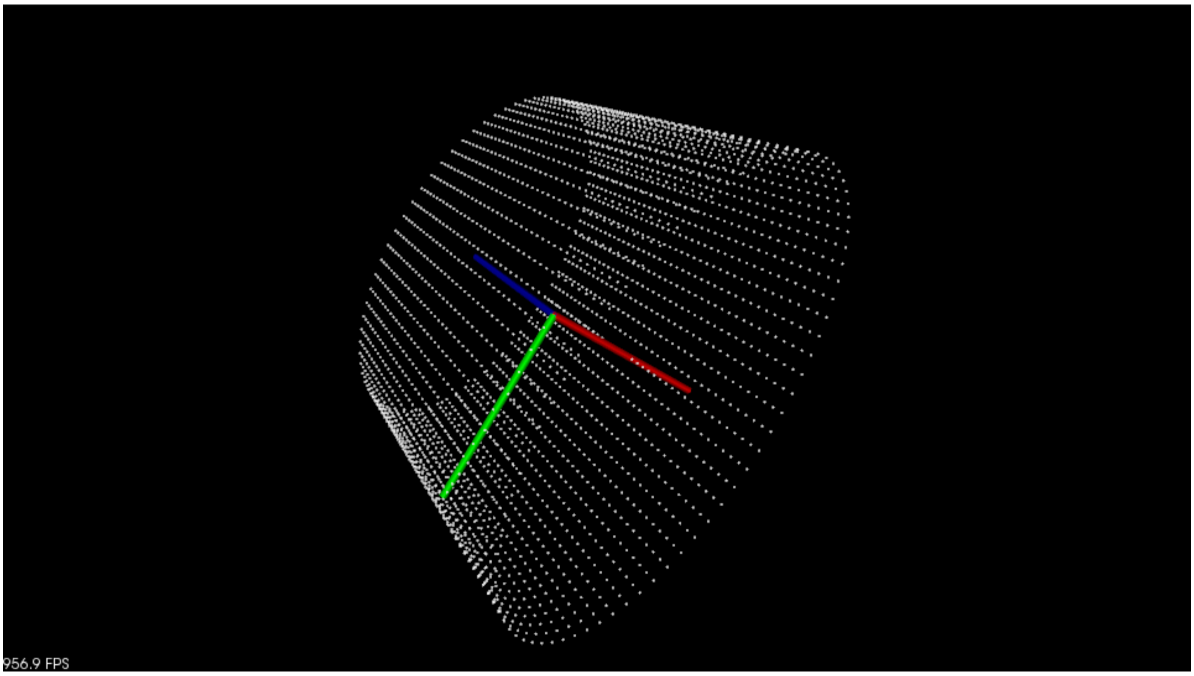
```
Usage: ./pcl_visualizer_demo [options]

Options:
-------------------------------------------
-h           this help
-s           Simple visualisation example
-r           RGB colour visualisation example
-c           Custom colour visualisation example
-n           Normals visualisation example
-a           Shapes visualisation example
-v           Viewports example
-i           Interaction Customization example
```

例如:

```
./pcl_visualizer_demo -s   //执行 Simple visualisation example
```

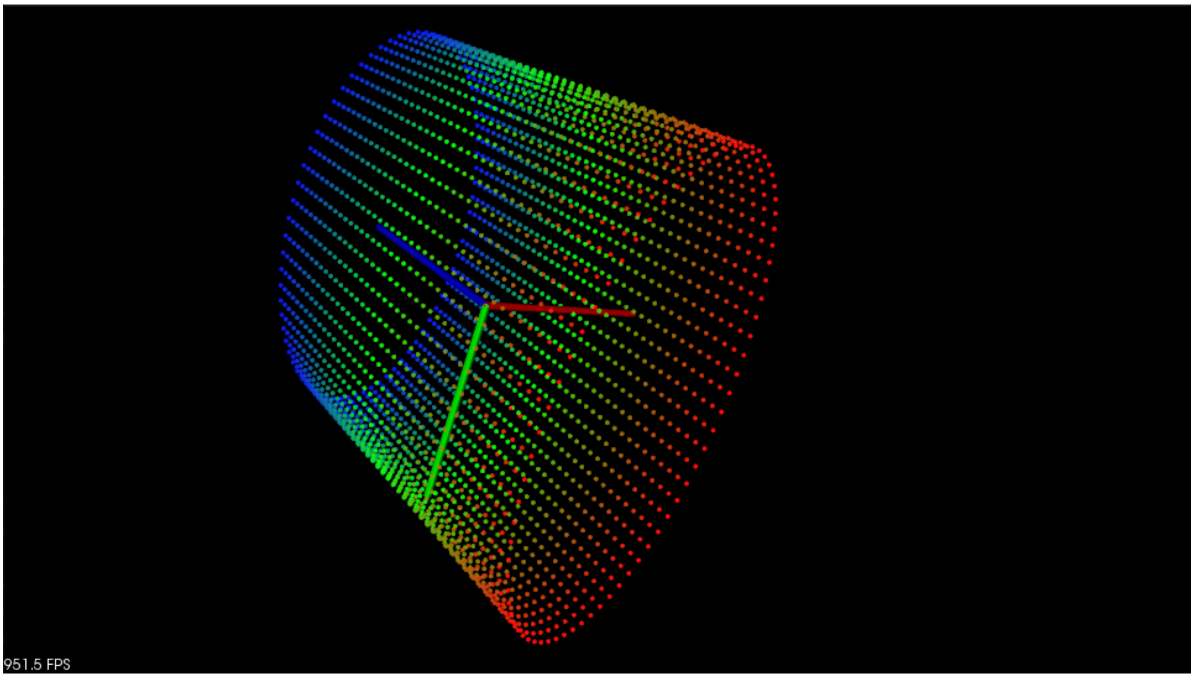
例 1 改变背景颜色改变坐标轴

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> simpleVis
(pcl::PointCloud<pcl::PointXYZ>::ConstPtr cloud)
{
  //创建3D窗口并添加点云
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  viewer->addPointCloud<pcl::PointXYZ> (cloud, "sample cloud");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 1, "sample cloud");
  viewer->addCoordinateSystem (1.0); //1.0是控制xyz轴圆柱体大小的尺度  默认值1.0
  viewer->initCameraParameters ();
  return (viewer);
}
```



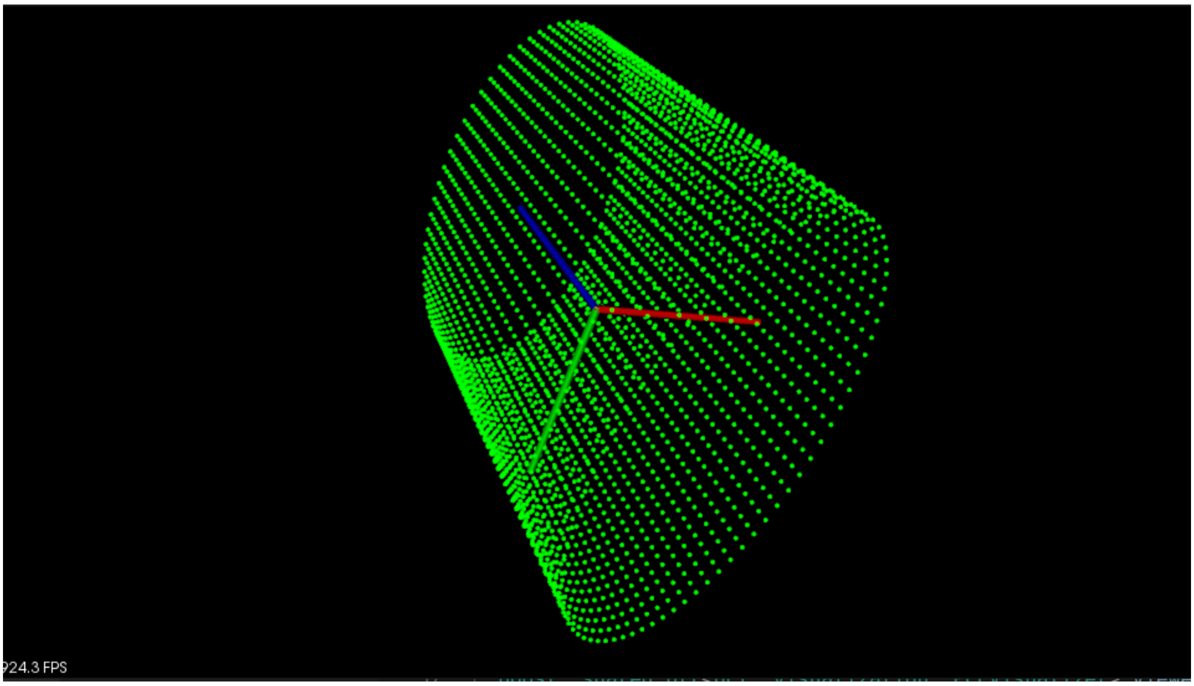
例 2 可视化点云颜色特征

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> rgbVis
(pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr cloud)
{
  //创建3D窗口并添加点云
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB>
rgb(cloud);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, rgb, "sample cloud");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud");
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  return (viewer);
}
```



例 3 点云着色

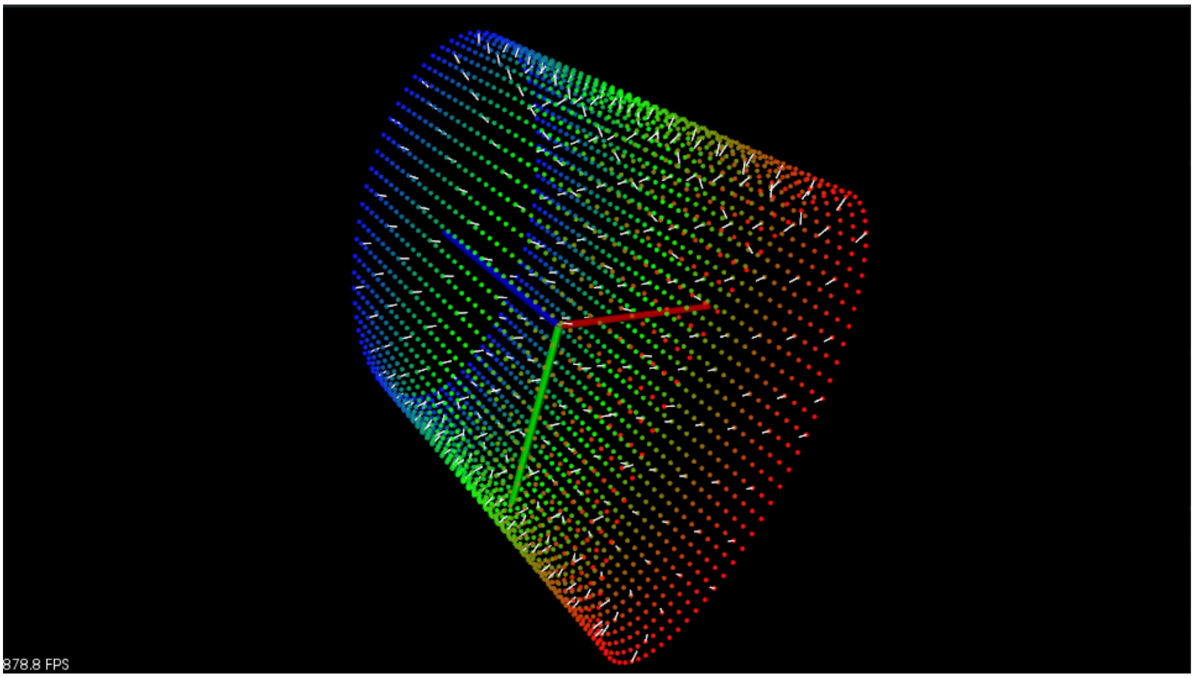
```
boost::shared_ptr<pcl::visualization::PCLVisualizer> customColourVis
(pcl::PointCloud<pcl::PointXYZ>::ConstPtr cloud)
{
//创建3D窗口并添加点云
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZ>
single_color(cloud, 0, 255, 0);
  viewer->addPointCloud<pcl::PointXYZ> (cloud, single_color, "sample cloud");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud");
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  return (viewer);
}
```



例 4 可视化点云法线和其他特征

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> normalsVis (
    pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr cloud,
pcl::PointCloud<pcl::Normal>::ConstPtr normals)
{
  //创建3D窗口并添加点云其包括法线
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB>
rgb(cloud);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, rgb, "sample cloud");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud");
  viewer->addPointCloudNormals<pcl::PointXYZRGB, pcl::Normal> (cloud, normals,
10, 0.05, "normals"); //表示每十个点显示一个 法线长度0.05
```

```
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  return (viewer);
}
```



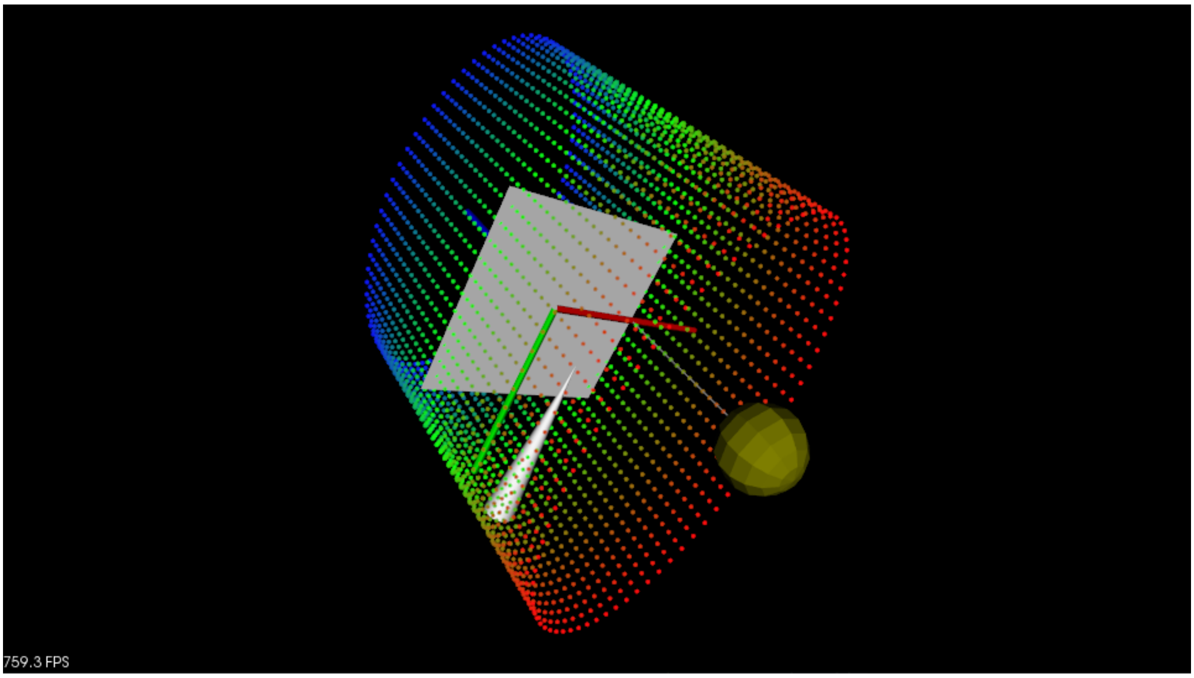
例 5 绘制普通形状

绘制一般图元的应用：可以通过绘制球体包围聚类得到的点云集以显示聚类结果。

四种形状可添加：

- 从点云第一个点到最后一个点之间的连线
- 原点所在平面
- 以点云中第一个点为中心的球体
- 沿着Y轴的椎体

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> shapesVis
(pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr cloud)
{
   //创建3D窗口并添加点云
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB>
rgb(cloud);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, rgb, "sample cloud");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud");
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  //在点云上添加直线和球体模型
  viewer->addLine<pcl::PointXYZRGB> (cloud->points[0],
                                     cloud->points[cloud->size() - 1], "line");
  viewer->addSphere (cloud->points[0], 0.2, 0.5, 0.5, 0.0, "sphere");
   //在其他位置添加基于模型参数的平面及圆锥体
   pcl::ModelCoefficients coeffs;
  coeffs.values.push_back (0.0);
  coeffs.values.push_back (0.0);
  coeffs.values.push_back (1.0);
  coeffs.values.push_back (0.0);
  viewer->addPlane (coeffs, "plane"); //ax + by +cz +d = 0 此代码表示 z = 0平面
  coeffs.values.clear ();
  /*参数说明
    pcl::ModelCoefficients cone_coeff;
    cone_coeff.values.resize (7);    // We need 7 values
    cone_coeff.values[0] = cone_apex.x (); //锥点坐标
    cone_coeff.values[1] = cone_apex.y ();
    cone_coeff.values[2] = cone_apex.z ();
    cone_coeff.values[3] = axis_direction.x (); //轴向量
    cone_coeff.values[4] = axis_direction.y ();
    cone_coeff.values[5] = axis_direction.z ();
    cone_coeff.values[6] = angle (); // degrees 角度
*/
  coeffs.values.push_back (0.3);
  coeffs.values.push_back (0.3);
  coeffs.values.push_back (0.0);
  coeffs.values.push_back (0.0);
  coeffs.values.push_back (1.0);
  coeffs.values.push_back (0.0);
  coeffs.values.push_back (5.0);
  viewer->addCone (coeffs, "cone");

  return (viewer);
}
```

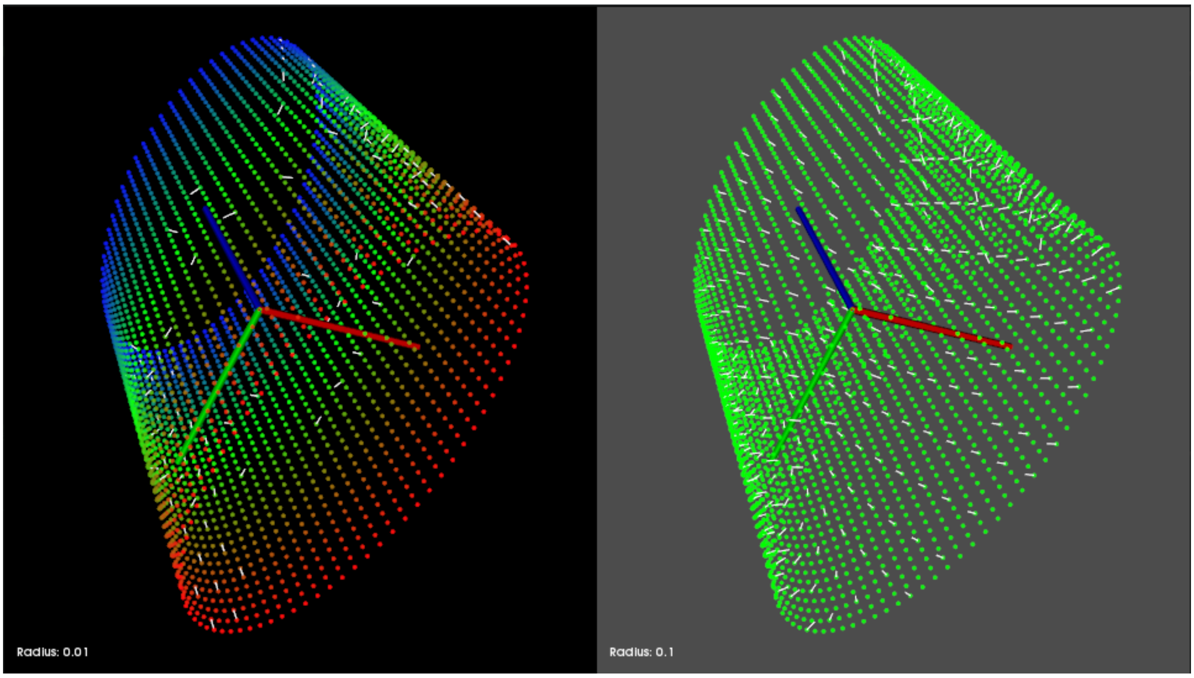


例 6 多视口显示

在绘制点云过程中，如果在一个窗口绘制多个点云，会导致信息混乱，所以PCLVisualizer允许通过不同窗口绘制(Viewport)多个点云，方便点云的对比分析。

如下代码，基于同一点云计算对应不同半径的两组法线：第一组搜索半径为0.01, 基于该半径计算的法线用黑色背景显示；第二组搜索半径为0.1，该半径计算法线用灰色背景显示。

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewportsVis (
    pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr cloud,
pcl::PointCloud<pcl::Normal>::ConstPtr normals1,
pcl::PointCloud<pcl::Normal>::ConstPtr normals2)
{
  // 创建3D窗口并添加显示点云其包括法线
   boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->initCameraParameters ();
  int v1(0);
  viewer->createViewPort(0.0, 0.0, 0.5, 1.0, v1);//该显示所占矩形的对角两点坐标（0，
0）（0.5， 1）v1表示视口id
  viewer->setBackgroundColor (0, 0, 0, v1);
  viewer->addText("Radius: 0.01", 10, 10, "v1 text", v1);
  pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB>
rgb(cloud);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, rgb, "sample cloud1", v1);
  int v2(0);
  viewer->createViewPort(0.5, 0.0, 1.0, 1.0, v2); //第二个 （0.5, 1,）(1, 1)
  viewer->setBackgroundColor (0.3, 0.3, 0.3, v2);
  viewer->addText("Radius: 0.1", 10, 10, "v2 text", v2);
  pcl::visualization::PointCloudColorHandlerCustom<pcl::PointXYZRGB>
single_color(cloud, 0, 255, 0);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, single_color, "sample cloud2",
v2);
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud1");
  viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud2");
  viewer->addCoordinateSystem (1.0);

  viewer->addPointCloudNormals<pcl::PointXYZRGB, pcl::Normal> (cloud, normals1,
10, 0.05, "normals1", v1);
  viewer->addPointCloudNormals<pcl::PointXYZRGB, pcl::Normal> (cloud, normals2,
10, 0.05, "normals2", v2);

  return (viewer);
}
```

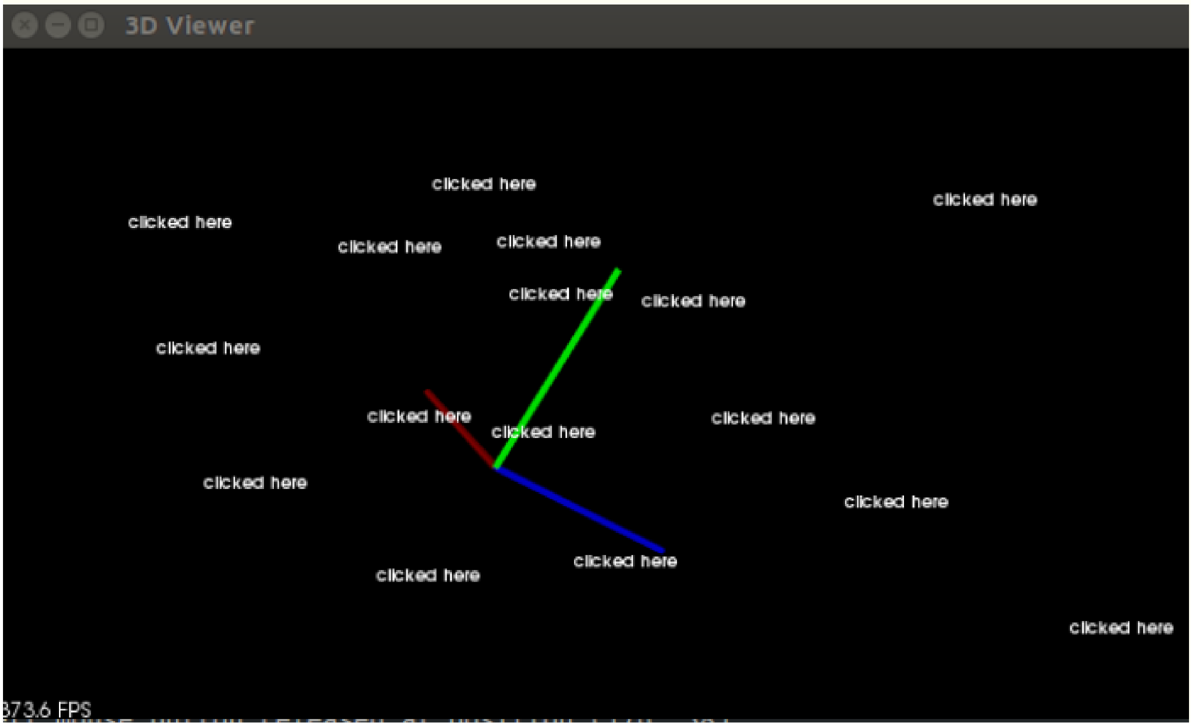


例 7 自定义交互

有时候鼠标和键盘交互设置不能满足用户的需求，需要扩展函数的某些功能，例如按下键盘保存点云信息、鼠标确定点云位置。在窗口右击会显示2D文本标签，按下R键擦除文本。

代码：

```
unsigned int text_id = 0;
void keyboardEventOccurred (const pcl::visualization::KeyboardEvent &event,
                            void* viewer_void)
{
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer =
*static_cast<boost::shared_ptr<pcl::visualization::PCLVisualizer> *>
(viewer_void);
  if (event.getKeySym () == "r" && event.keyDown ())
  {
    std::cout << "r was pressed => removing all text" << std::endl;

    char str[512];
    for (unsigned int i = 0; i < text_id; ++i)
    {
      sprintf (str, "text#%03d", i);
      viewer->removeShape (str);
    }
    text_id = 0;
  }
}

void mouseEventOccurred (const pcl::visualization::MouseEvent &event,
                         void* viewer_void)
{
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer =
*static_cast<boost::shared_ptr<pcl::visualization::PCLVisualizer> *>
(viewer_void);
  if (event.getButton () == pcl::visualization::MouseEvent::LeftButton &&
      event.getType () == pcl::visualization::MouseEvent::MouseButtonRelease)
  {
    std::cout << "Left mouse button released at position (" << event.getX () <<
", " << event.getY () << ")" << std::endl;

    char str[512];
    sprintf (str, "text#%03d", text_id ++);
    viewer->addText ("clicked here", event.getX (), event.getY (), str);
  }
}

boost::shared_ptr<pcl::visualization::PCLVisualizer> interactionCustomizationVis
()
{
  boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  viewer->addCoordinateSystem (1.0);
  // 注册响应键盘鼠标事件，调用回调函数
  viewer->registerKeyboardCallback (keyboardEventOccurred, (void*)&viewer);
  viewer->registerMouseCallback (mouseEventOccurred, (void*)&viewer);
  return (viewer);
}
```



可以输出相关信息，但是鼠标点击的时候会宕掉。

5.4 PCLPlotter 可视化直方图

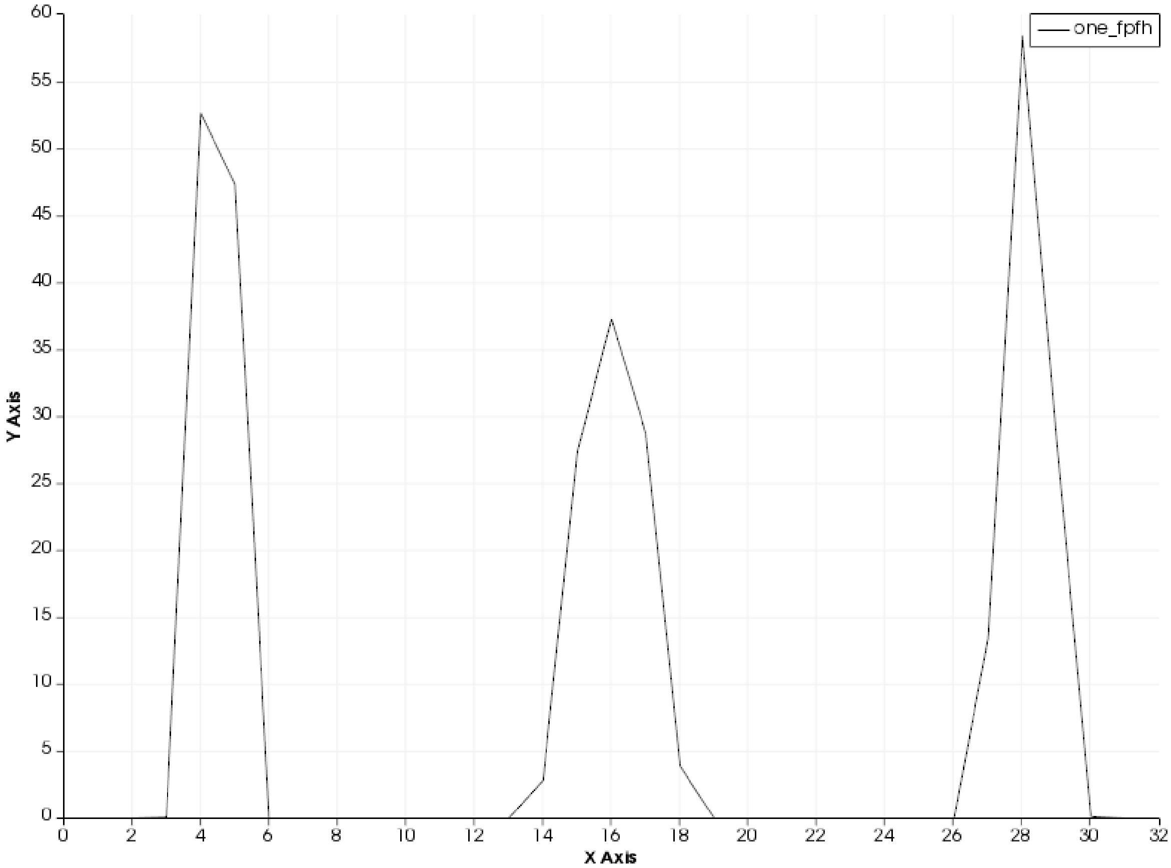
PCLPlotter提供一个直接简单的绘图接口，可绘制**二维图形**、**多项式函数**、**特征直方图**（FPFH）等。利用PCLPlotter绘图步骤：

- 一、声明PLCPlotter对象；
- 二、利用addPlotData() 函数添加绘图所需的函数或数据；
- 三、添加窗口特性，可以调整窗口大小和标题设置；
- 四、显示绘图

例 1 绘制点云FPFH特征点直方图

代码：

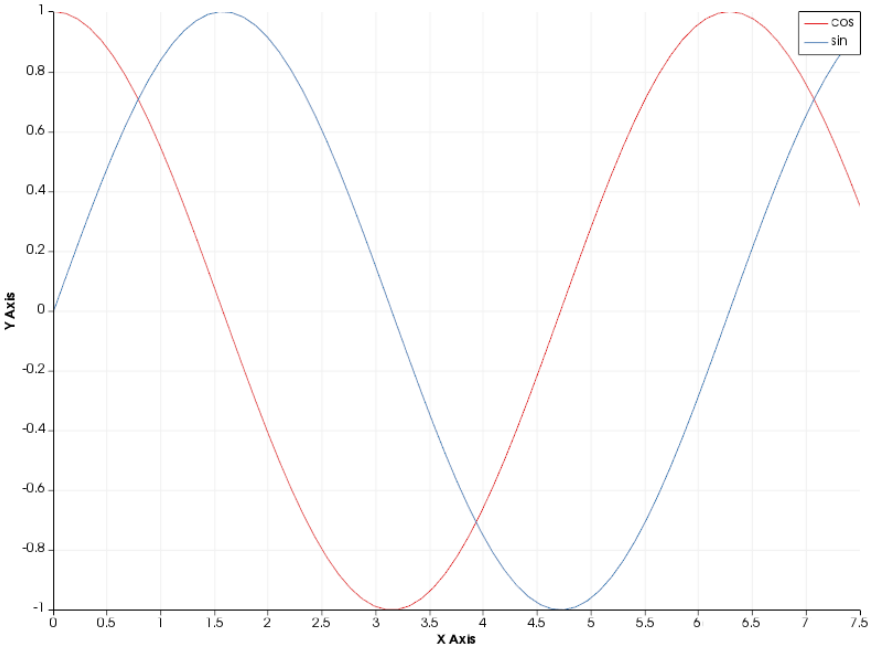
```
//定义绘图器
  PCLPlotter *plotter = new PCLPlotter ("My Plotter");
  //设置特性
  plotter->setShowLegend (true);//显示图例  如果是fasle右上角的one_fpfh图例就消失了
  std::cout<<pcl::getFieldsList<pcl::FPFHSignature33>(*fpfh_src);

  // 1.点云数据 2.点云类型的field name 3.特征点的索引 4.id 在图的右上角的图例，  绘制点云第5个点的点快速特征直方图
  plotter->addFeatureHistogram<pcl::FPFHSignature33>
(*fpfh_src,"fpfh",5,"one_fpfh");
  //显示两秒
  plotter->setWindowSize (800, 600);
  plotter->spinOnce (2000);//显示窗口的时间
  plotter->clearPlots ();//清除窗口曲线
```

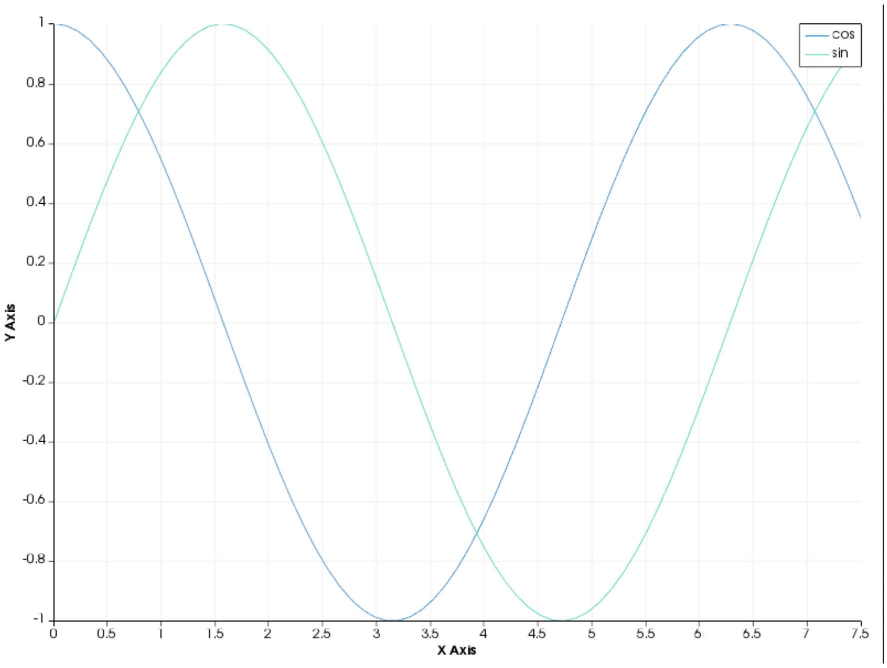


例 2 绘制正弦余弦图

代码：

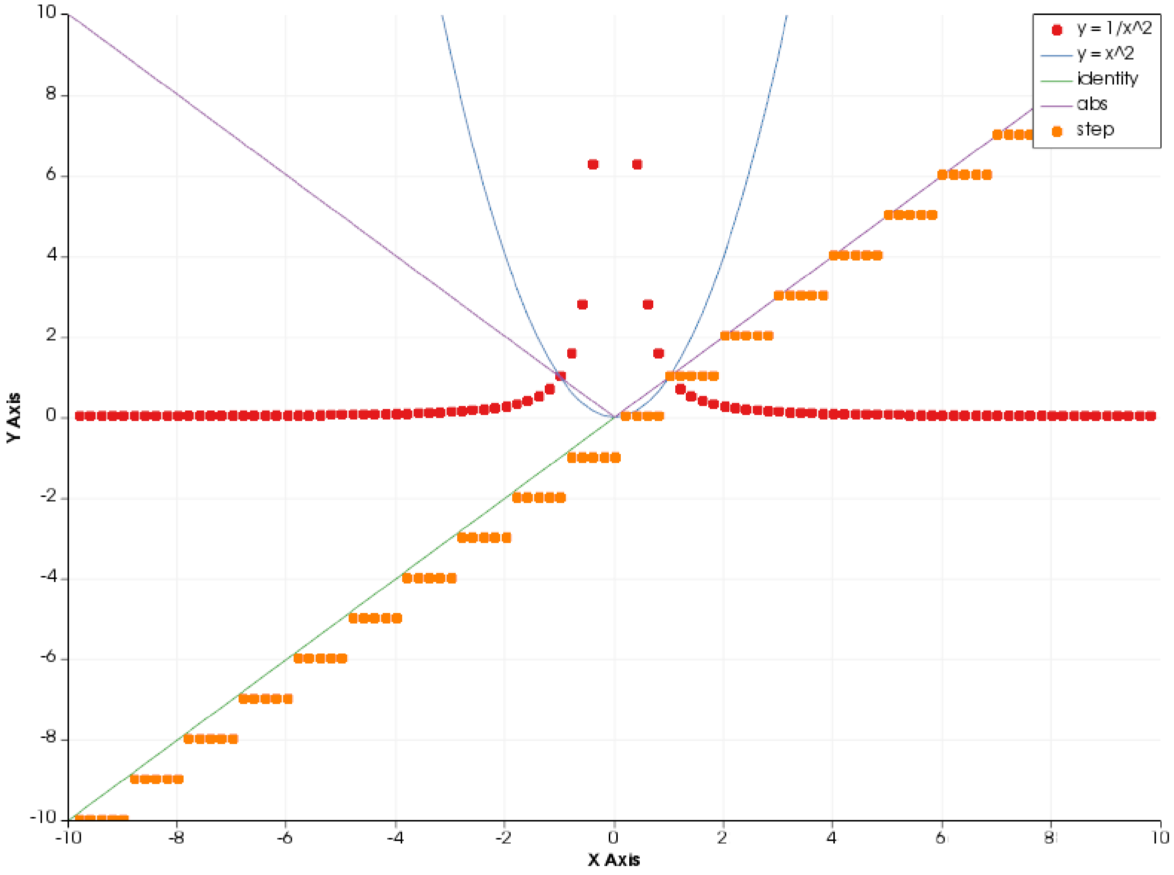
```
// 产生对应点对
  int numPoints = 69;
  double ax[100], acos[100], asin[100];
  generateData (ax, acos, asin, numPoints);

  /*添加绘图数据
  void pcl::visualization::PCLPlotter::addPlotData(const double *array_X, const
double *array_Y, unsigned long size, const char *name = "Y Axis", int type = 0,
const char  *color = (const char *)__null)
还有 6 个重载

Adds a plot with correspondences in the arrays arrayX and arrayY

参数:
array_X - X coordinates of point correspondence array
array_Y - Y coordinates of point correspondence array
size - length of the array arrayX and arrayY
name - name of the plot which appears in the legend when toggled on
type - type of the graph plotted. vtkChart::LINE for line plot, vtkChart::BAR for
bar plot, and vtkChart::POINTS for a scattered point plot

默认: vtkColorSeries::SPECTRUM 需要#include <vtkColorSeries.h>头文件
  vtkColorSeries::WARM
  vtkColorSeries::COOL
  vtkColorSeries::BLUES
  vtkColorSeries::WILD_FLOWER
  vtkColorSeries::CITRUS
  */
  //plotter->setColorScheme(vtkColorSeries::BLUES);
  plotter->addPlotData (ax, acos, numPoints, "cos");
  //plotter->setColorScheme(vtkColorSeries::COOL);
  plotter->addPlotData (ax, asin, numPoints, "sin");

  //显示2s
  plotter->spinOnce (2000);
  plotter->clearPlots ();
```





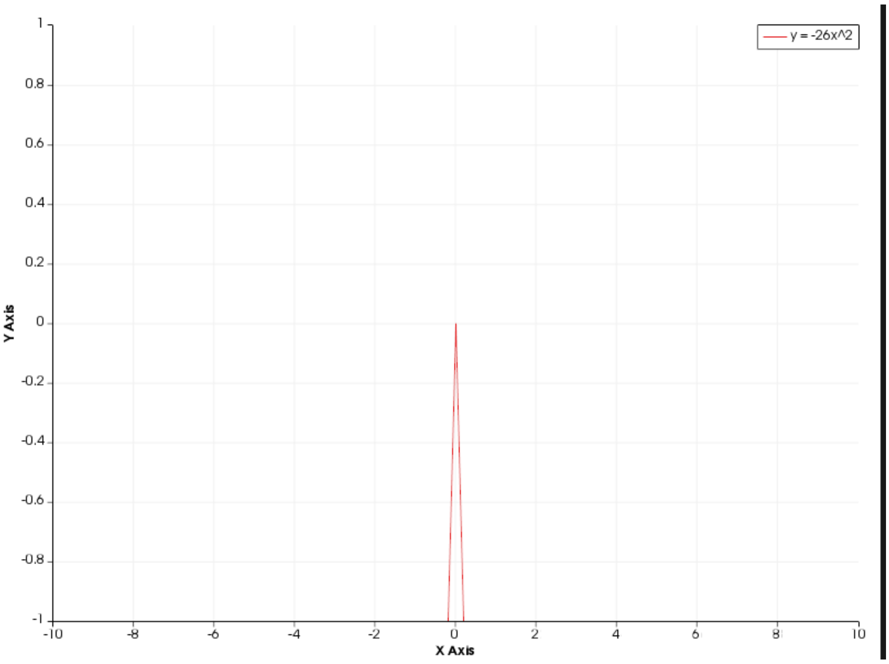
例 3 绘制隐式函数图组

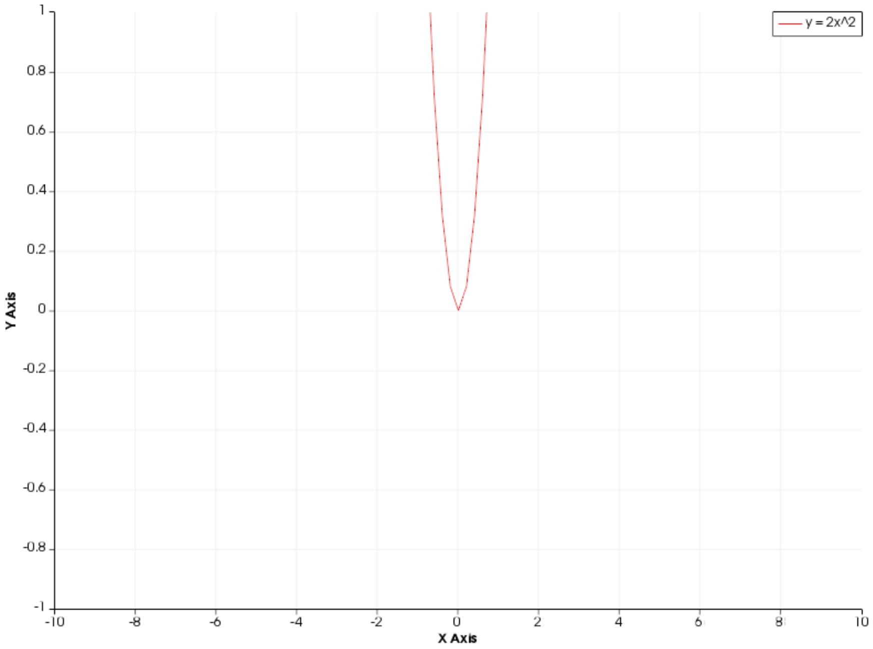
```
//...................绘制隐式函数...................

  //设置y轴范围
  plotter->setYRange (-10, 10);
  //定义多项式
  vector<double> func1 (1, 0);
  func1[0] = 1; //y = 1
  vector<double> func2 (3, 0);
  func2[2] = 1; //y = x^2

  plotter->addPlotData (std::make_pair (func1, func2), -10, 10, "y = 1/x^2",
100, vtkChart::POINTS);
  plotter->spinOnce (2000);

  plotter->addPlotData (func2, -10, 10, "y = x^2");
  plotter->spinOnce (2000);

  //回调函数
  plotter->addPlotData ( identity_i, -10, 10, "identity");
  plotter->spinOnce (2000);

  plotter->addPlotData (abs, -10, 10, "abs");
  plotter->spinOnce (2000);

  plotter->addPlotData (step, -10, 10, "step", 100, vtkChart::POINTS);
  plotter->spinOnce (200000);

  plotter->clearPlots ();
```



例 4 一个函数动画

```
//函数从 y = 100x^2 -> y = -100x^2 变化
vector<double> fsq (3, 0);
  fsq[2] = -100; //y = x^2
  while (plotter->wasStopped ())
  {
    if (fsq[2] == 100) fsq[2] = -100;
    fsq[2]++;
    char str[50];
    sprintf (str, "y = %dx^2", (int) fsq[2]);
    plotter->addPlotData (fsq, -10, 10, str);
      plotter->setYRange (-1, 1);
    plotter->spinOnce (100);
    plotter->clearPlots ();
  }

  return 1;
}
```


y = -26x^2
Y Axis
X Axis


y = 2x^2
Y Axis
X Axis