城市空间建模与仿真

第五讲　城市空间三维数据表达和重建-点云与面片数据表达与转换

任课教师：汤圣君
建筑与城市规划学院　城市空间信息工程系

其中部分图片来自互联网和同行专家

目录

CONTENTS

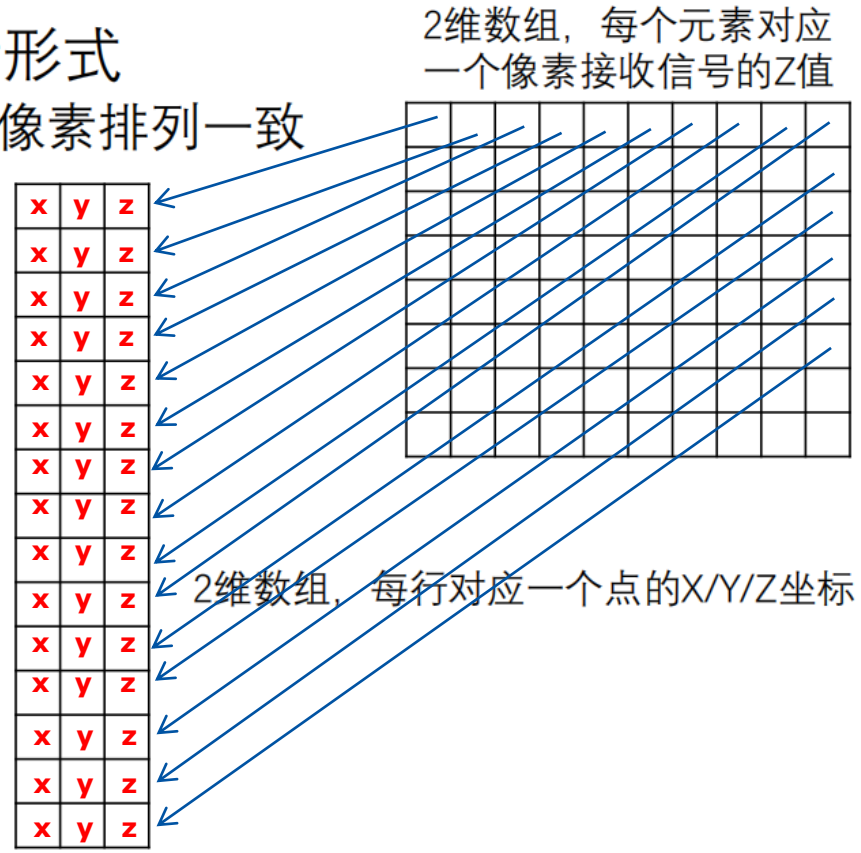
目录

CONTENTS

点云数据表达与格式转换

- 3D相机数据的数据存储形式
  存储次序往往和传感器的像素排列一致

- 点云的数据存储形式
  存储次序（行次序）任意



point cloud formats:*.pcd, *.ply, *.E57, *.FLS, *.RCP, *.PTX, *.ZFS,*.LAS, *.LAZ, *.PTS, *.DP, *.FPR, *.LSPROJ, *.FWS, *.CL3, *.CLR, *.RSP, ASCII / NEZ (X,Y,Z/i/RGB) and custom ASCII / TXT file format imports.

点云数据表达与格式转换

PCD全称Point Cloud Data，是一种存储点云数据的文件格式。发明PCD文件格式不是重造轮子，因为现有的“轮子”都无法满足PCL的数据处理需求。**PCD格式出现之前表示激光扫描仪获取的点云、任意多边形等的文件格式比如：**

1、PLY：表示多边形的文件格式。
2、STL：CAD文件格式，用3d max或CAD软件处理。
3、OBJ：一种几何学格式文件。
4、X3D：ISO标准的基于XML格式的计算机3D图形文件格式。

样例：

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

点云数据表达与格式转换

```
# .PCD v.7 - Point Cloud Data file
format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

·VERSION –指定PCD文件版本

·FIELDS –指定一个点可以有的每一个维度和字段的名字。例如：

```
FIELDS x y z                          # XYZ data
FIELDS x y z rgb                      # XYZ + colors
FIELDS x y z normal_xnormal_y normal_z      # XYZ + surface normals
FIELDS j1 j2 j3                       # moment invariants
```

PCD数据解析

点云数据表达与格式转换

```
# .PCD v.7 - Point Cloud Data file
format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

·SIZE –用字节数指定每一个维度的大小。例如：
unsigned char/char has 1 byte
unsigned short/short has 2 bytes
unsignedint/int/float has 4 bytes
double has 8 bytes
·TYPE –用一个字符指定每一个维度的类型。现在被接受的类型有：
I –表示有符号类型int8（char）、int16（short）和int32（int）；
U – 表示无符号类型uint8（unsigned char）、uint16（unsigned short）和uint32（unsigned int）；
F –表示浮点类型。

PCD数据解析

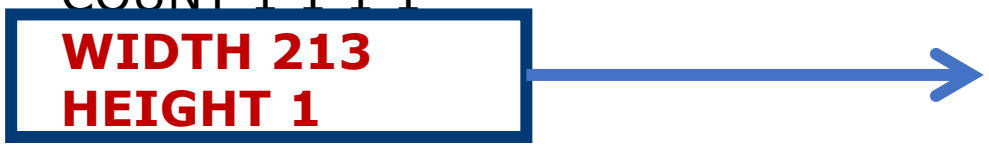
点云数据表达与格式转换

```
# .PCD v.7 - Point Cloud Data file
format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

·COUNT –指定每一个维度包含的元素数目。例如，x这个数据通常有一个元素，但是像VFH这样的特征描述子就有308个。实际上这是在给每一点引入n维直方图描述符的方法，把它们当做单个的连续存储块。默认情况下，如果没有COUNT，所有维度的数目被设置成1。

点云数据表达与格式转换

```
# .PCD v.7 - Point Cloud Data file
format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```



·*WIDTH –用点的数量表示点云数据集的宽度。根据是有序点云还是无序点云，WIDTH有两层解释：*

1)它能确定无序数据集的点云中点的个数（和下面的POINTS一样）；

2)它能确定有序点云数据集的宽度（一行中点的数目）。

例如：

WIDTH 640　　# 每行有640个点

·HEIGHT –用点的数目表示点云数据集的高度。类似于WIDTH，*HEIGHT也有两层解释：*

1)它表示有序点云数据集的高度（行的总数）；

2)对于无序数据集它被设置成1（被用来检查一个数据集是有序还是无序）。

有序点云例子：

WIDTH 640　　# 像图像一样的有序结构，有640行和480列，

HEIGHT 480　　# 这样该数据集中共有640*480=307200个点

无序点云例子：

WIDTH 307200

HEIGHT 1　　# 有307200个点的无序点云数据集

PCD数据解析

点云数据表达与格式转换

```
# .PCD v.7 - Point Cloud Data file
format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F FFF
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
```

·VIEWPOINT–指定数据集中点云的获取视点。VIEWPOINT有可能在不同坐标系之间转换的时候应用，在辅助获取其他特征时也比较有用，例如曲面法线，在判断方向一致性时，需要知道视点的方位.

视点信息被指定为平移（txtytz）+四元数（qwqxqyqz）。默认值是：

VIEWPOINT 0 0 0 1 0 0 0

·POINTS–指定点云中点的总数。从0.7版本开始，该字段就有点多余了，因此有可能在将来的版本中将它移除。

例子：

POINTS 307200 #点云中点的总数为307200

·DATA –指定存储点云数据的数据类型。从0.7版本开始，支持两种数据类型：ascii和二进制。

PCD数据解析

注意：PCD文件的文件头部分必须以上面的顺序精确指定，也就是如下顺序：VERSION、FIELDS、SIZE、TYPE、COUNT、WIDTH、HEIGHT、VIEWPOINT、POINTS、DATA

点云数据表达与格式转换

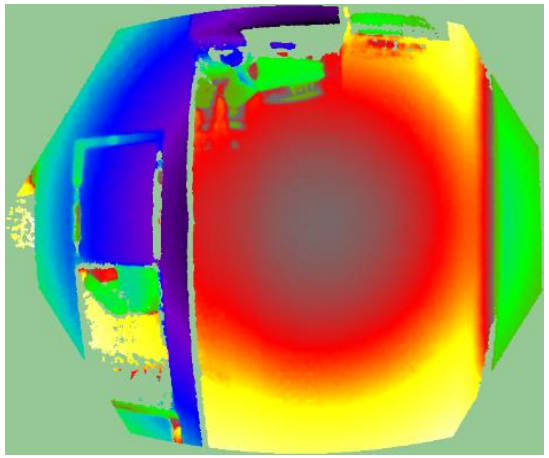
相对其他文件格式的优势

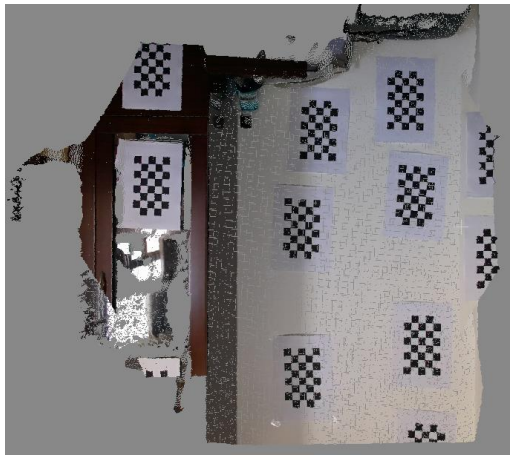
用PCD作为（另一种）文件格式可能被看成是没有必要的一项工作。但实际中，情况不是这样的，因为上面提到的文件格式无一能提高PCD文件的适用性和速度。PCD文件格式包括以下几个明显的优势：

- **存储和处理有序点云数据集的能力——这一点对于实时应用，例如增强现实、机器人学等领域十分重要；**
- **存储不同的数据类型（支持所有的基本类型：char，short，int，float，double）——使得点云数据在存储和处理过程中适应性强并且高效，其中无效的点的通常存储为NAN类型；**
- **特征描述子的n维直方图——对于3D识别和计算机视觉应用十分重要。**

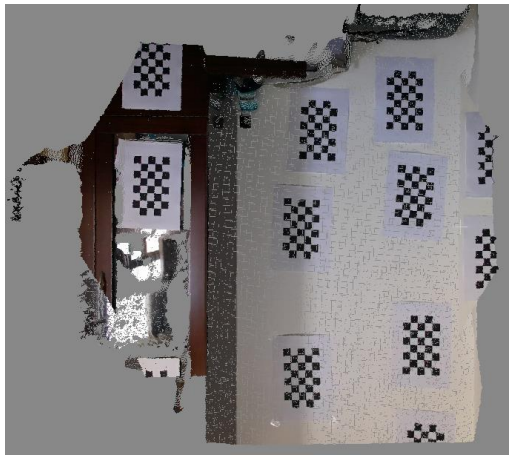
点云数据表达与格式转换

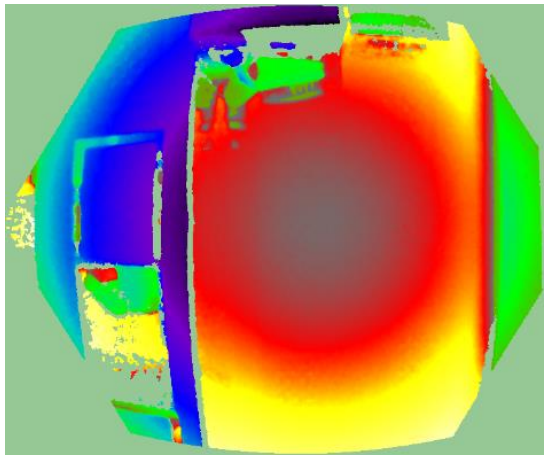
深度图到点云





点云到深度图





RangeImage生成

点云数据表达与格式转换

为从深度相机得到的每个点云“染色”

RGB相机中的像素位置
（和TOF相机坐标系
下的点云对应）

TOF相机坐标系
下的点云坐标

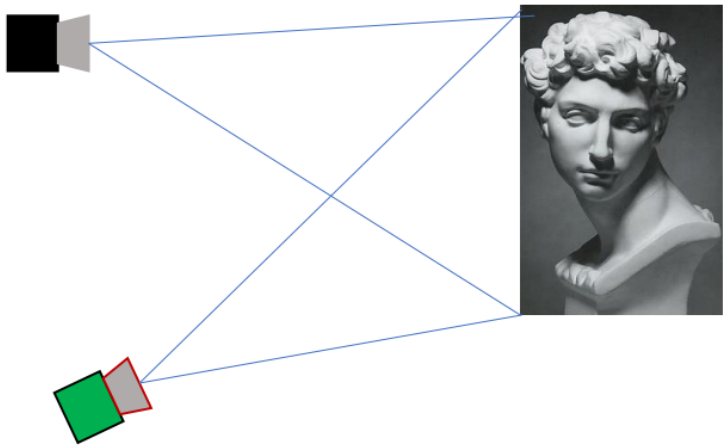
$$Z\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\r_{21} & r_{22} & r_{23} & t_y\\r_{31} & r_{32} & r_{33} & t_z\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

RGB相机
的内参

TOF相机坐标系转到
RGB相机坐标系

Rangeimage 的横纵坐标，

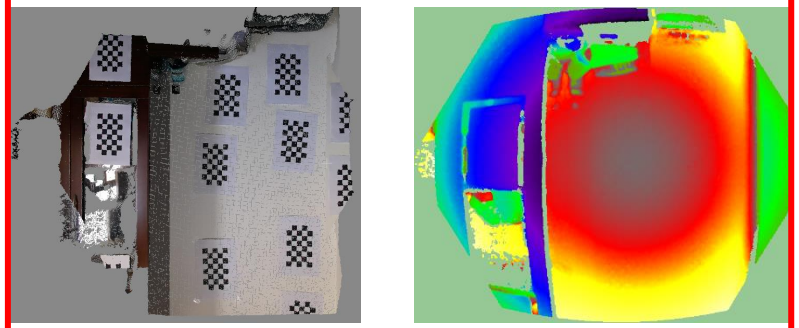
Rangeimage 的像素灰度



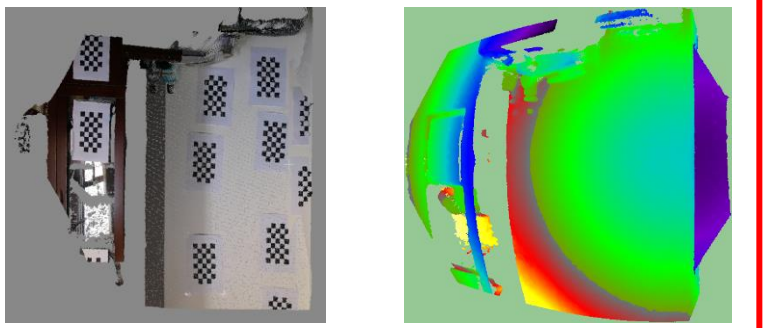
RangeImage生成

点云数据表达与格式转换

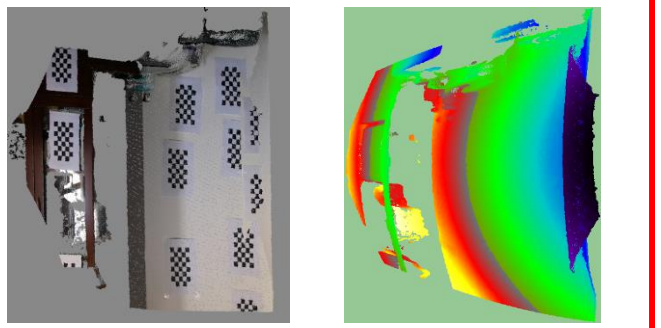
原始角度

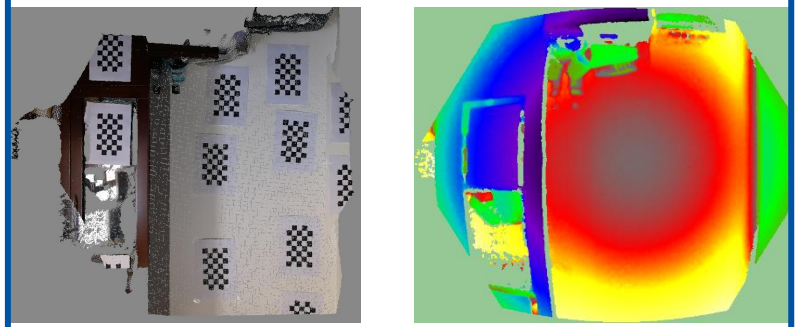


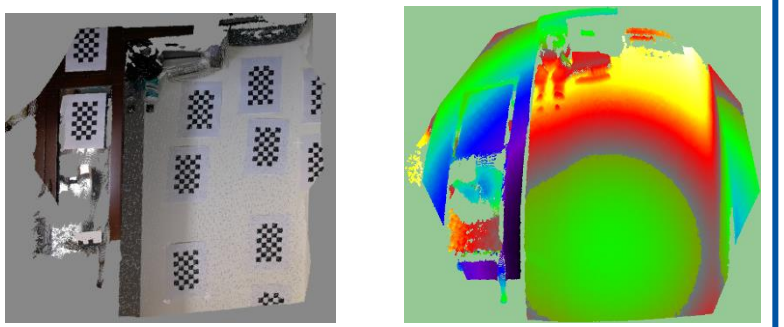
X轴旋转0.5弧度

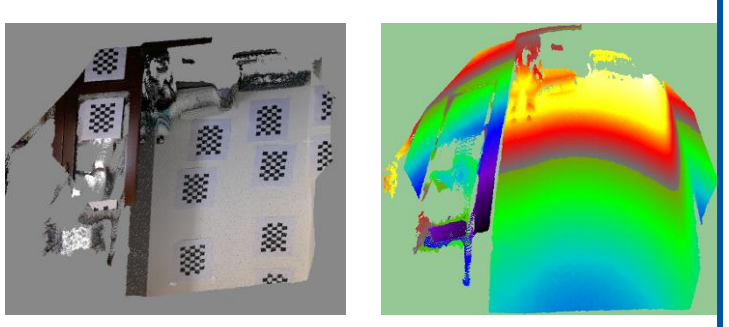


X轴旋转1弧度









原始角度

Y轴旋转0.5弧度

Y轴旋转1弧度

RangeImage生成

目录

CONTENTS

02 面片数据表达与转换

面片数据表达与格式转换

- 用多个平面片（三角形或者四边形）描述空间几何体表面
- 有ASCII和二进制2种格式（这里只介绍ASCII格式）
- 平面片描述拆分为：1）顶点描述；2）构成平面片的顶点序号

- 这里介绍基本的格式内容

面片数据表达与格式转换

```
ply
format ascii 1.0
comment test
element vertex 4
property float x
property float y
property float z
element face 4
property list uchar int vertex_index
end_header
0 3 0
2.449 -1.0 -1.414
0 -1 2.828
-2.449 -1.0 -1.414
3 0 1 3
3 0 2 1
3 0 3 2
3 1 2 3
```

头部分

数据部分

PLY数据格式—例子

面片数据表达与格式转换

格式描述

注释

顶点的数据格式描述

三角面的数据格式描述

```
ply
format ascii 1.0
comment test
element vertex 4
property float x
property float y
property float z
element face 4
property list uchar int vertex_index
end_header
0 3 0
2.449 -1.0 -1.414
0 -1 2.828
-2.449 -1.0 -1.414
3 0 1 3
3 0 2 1
3 0 3 2
3 1 2 3
```

- 存在**4个顶点**
- 他们的数据使用**x/y/z**三个**单精度浮点数**描述

- 存在**4个三角面**
- 他们的数据使用**列表**描述，
- 列表长度用**无符号8-bit整数**，对应**定点序号**用**整数表示**

注意：上面描述和左边的内容文字颜色对应

面片数据表达与格式转换

```
ply
format ascii 1.0
comment test
element vertex 4
property float x
property float y
property float z
element face 4
property list uchar int vertex_index
end_header
0 3 0          顶点0坐标
2.449 -1.0 -1.414   顶点1坐标
0 -1 2.828     顶点2坐标
-2.449 -1.0 -1.414  顶点3坐标
3 0 1 3
3 0 2 1
3 0 3 2
3 1 2 3
```

每行3个浮点数

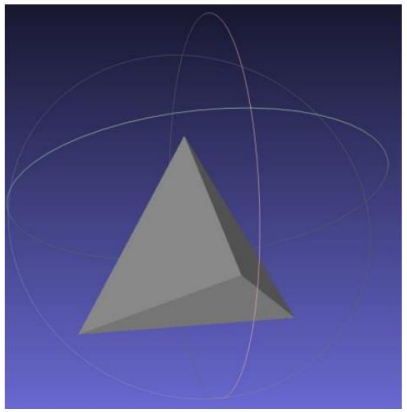
每行开头是列表长度，之后跟着3个整数对应的定点序号

对应4行数据

对应4行数据

- 存在4个顶点
- 他们的数据使用x/y/z三个单精度浮点数描述

- 存在4个面
- 他们的数据使用列表描述，
- 列表长度用无符号8-bit整数，对应定点序号用整数表示



面片数据表达与格式转换

头部分

```
ply
format ascii 1.0
comment author: ABC
comment object: cube
element vertex 8
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
element face 7
property list uchar int vertex_index
end_header
```

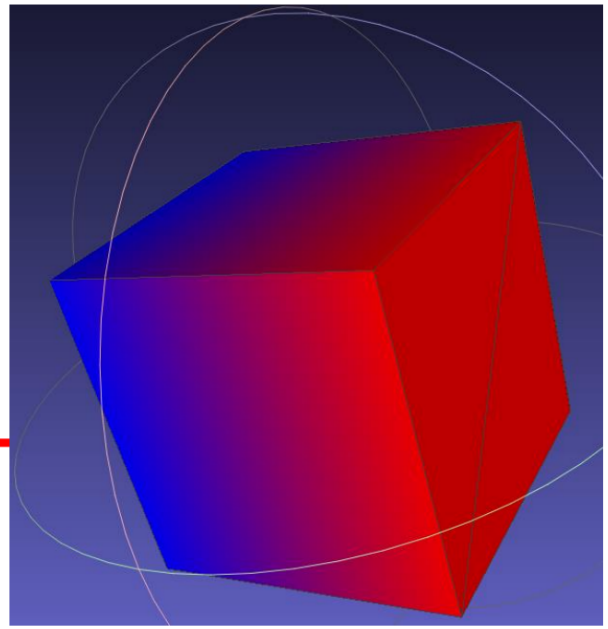
8行顶点数据

7行面数据

数据部分

点坐标			色彩		
0	0	0	255	0	0
0	0	1	255	0	0
0	1	1	255	0	0
0	1	0	255	0	0
1	0	0	0	0	255
1	0	1	0	0	255
1	1	1	0	0	255
1	1	0	0	0	255

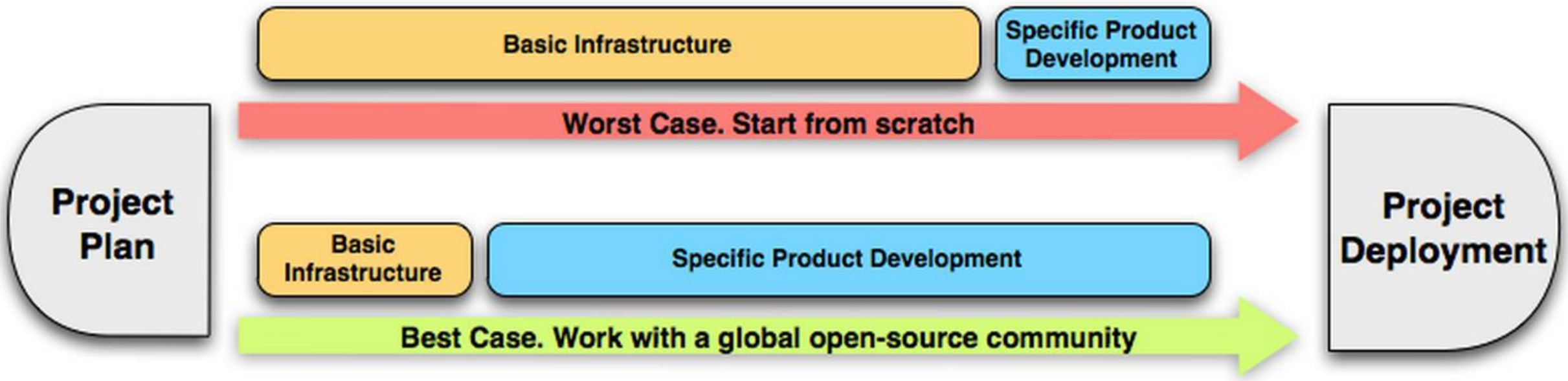
列表长度	顶点序号				
3	0	1	2		三角形
3	0	2	3		
4	7	6	5	4	四边形
4	0	4	5	1	
4	1	5	6	2	
4	2	6	7	3	
4	3	7	4	0	



目录

CONTENTS

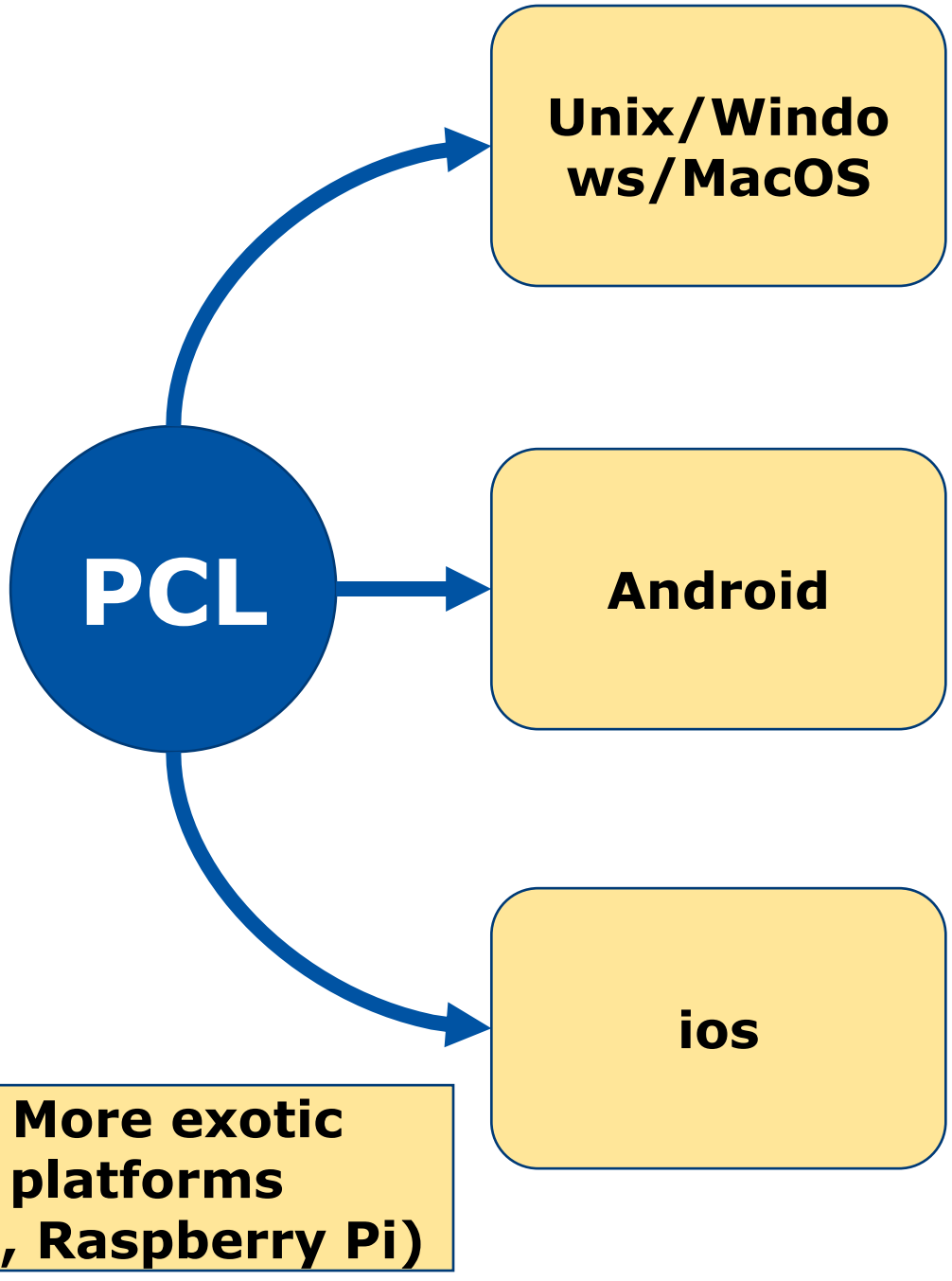
Why PCL


Basic Infrastructure
Specific Product Development
Worst Case. Start from scratch
Project Plan
Basic Infrastructure
Specific Product Development
Best Case. Work with a global open-source community
Project Deployment

pointcloudlibrary

The Point Cloud Library (PCL) is a large scale, open project for point cloud processing. The PCL framework contains numerous state-of-the art algorithms including **filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation**.

- 超过六百多个开发者
- 3-clause BSD license
- PCL is **cross-platform**, and has been successfully compiled and deployed on Linux, MacOS, Windows, and Android/iOS.

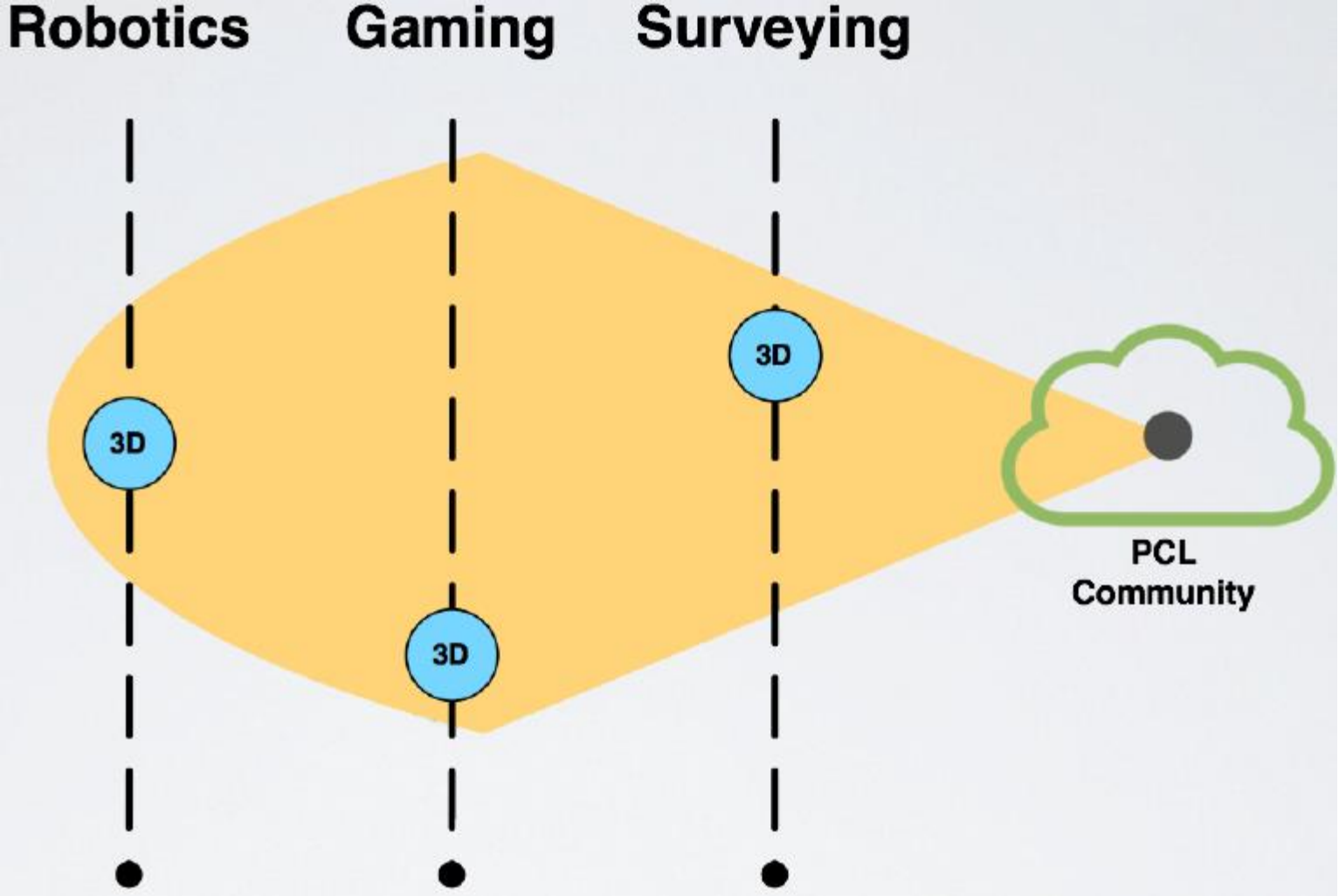


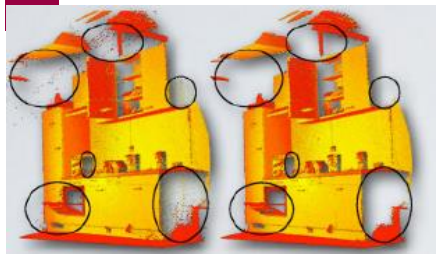
- ✓ support for multiple sensors
- ✓ Processing
- ✓ Visualization
- ✓ CUDA implementations

- ✓ visualization with OpenGL ES and VES

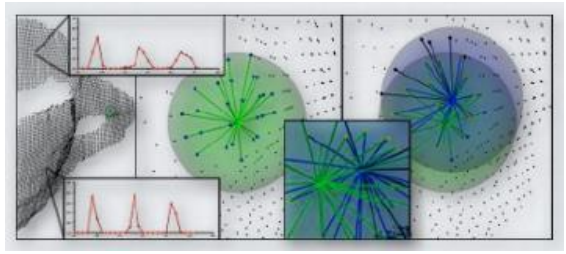
- ✓ Apple does not allow unlicensed 3rd party accessories
- ✓ visualization with OpenGL ES and VES

Motivation

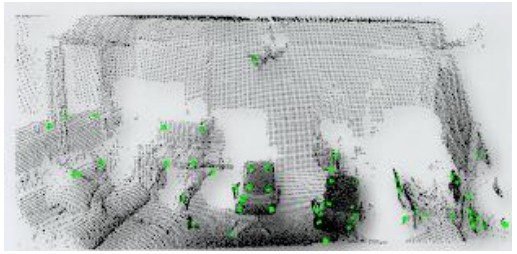




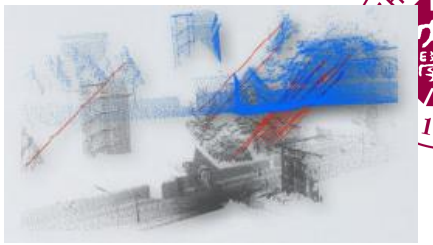
Features



Filters

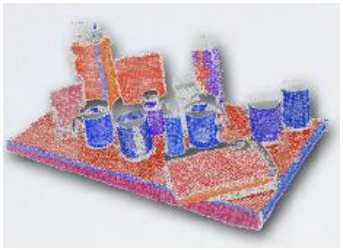


Keypoints

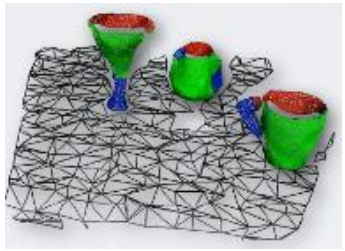


Registration

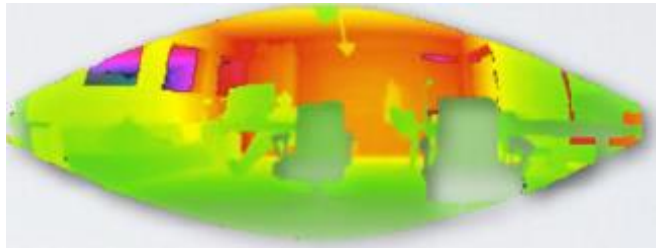
Segmentation



Sample Consensus



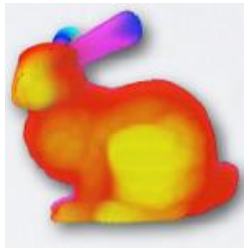
Surface



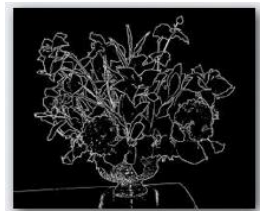
Range Image

I/O

People

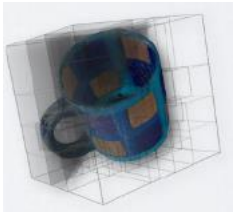


Visualization



2D

ML

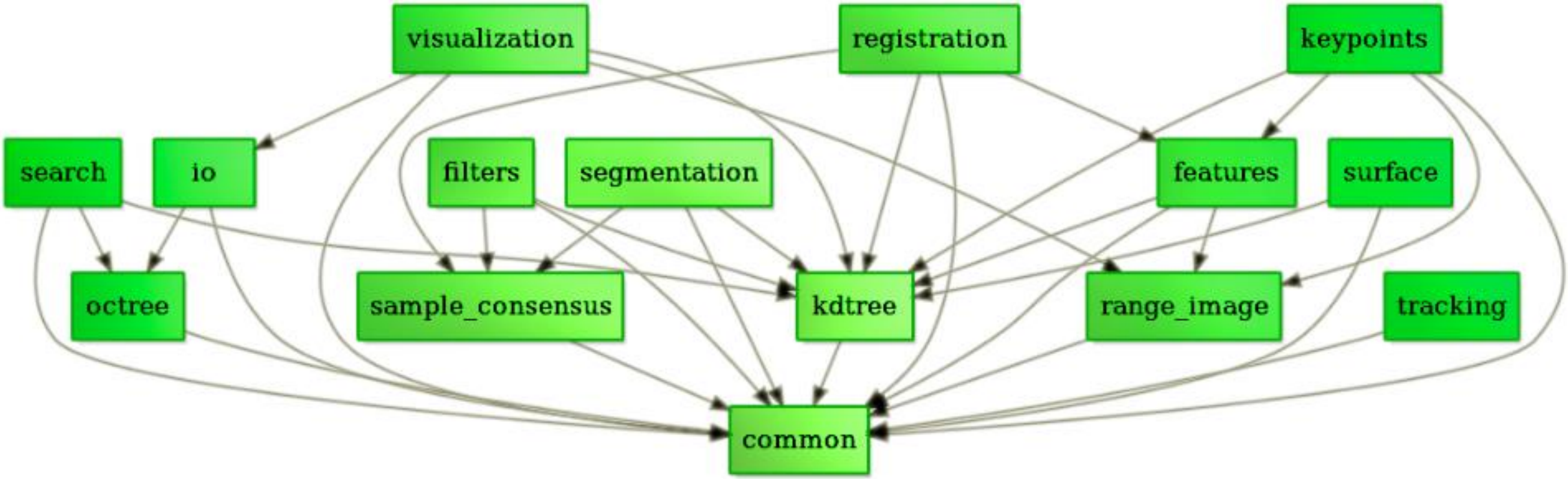


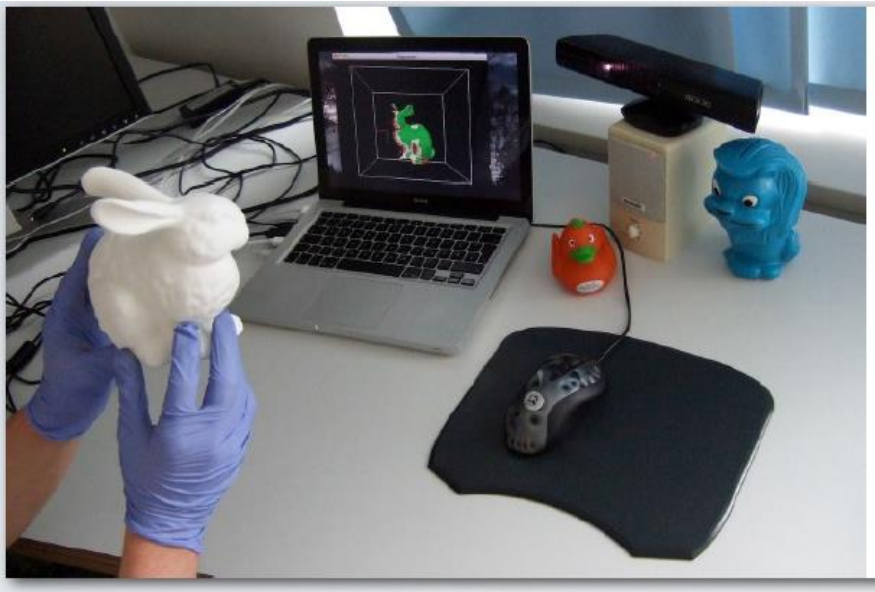
Kdtree

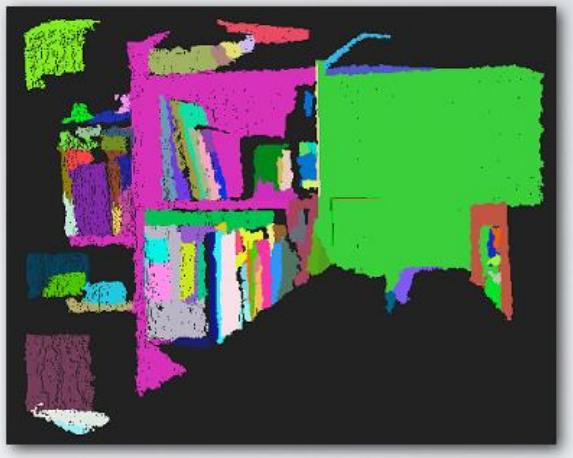
Octree

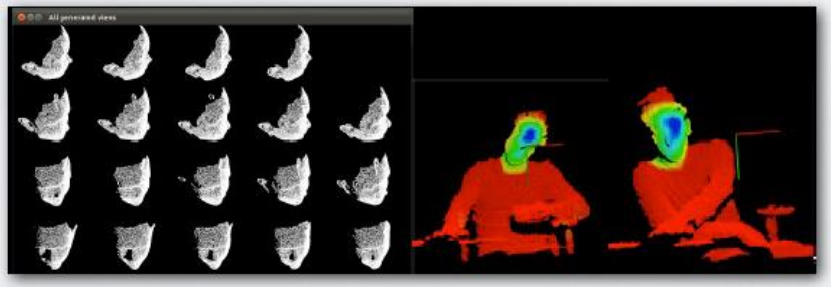
Modules

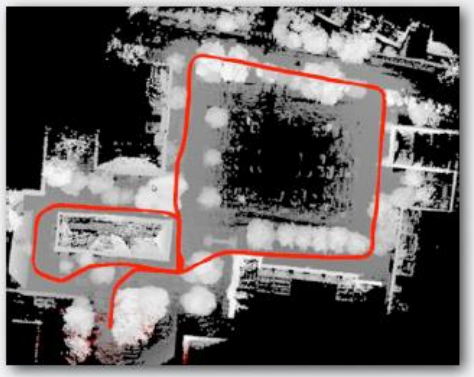
Module dependencies

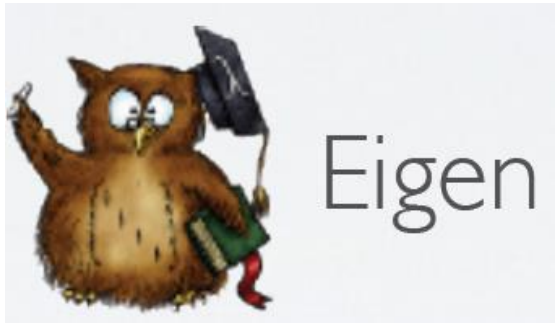


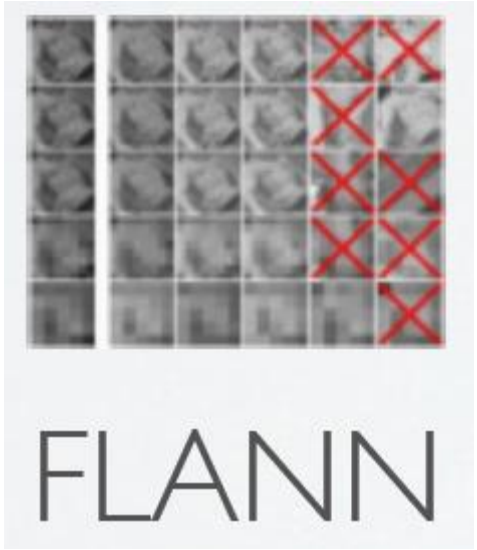






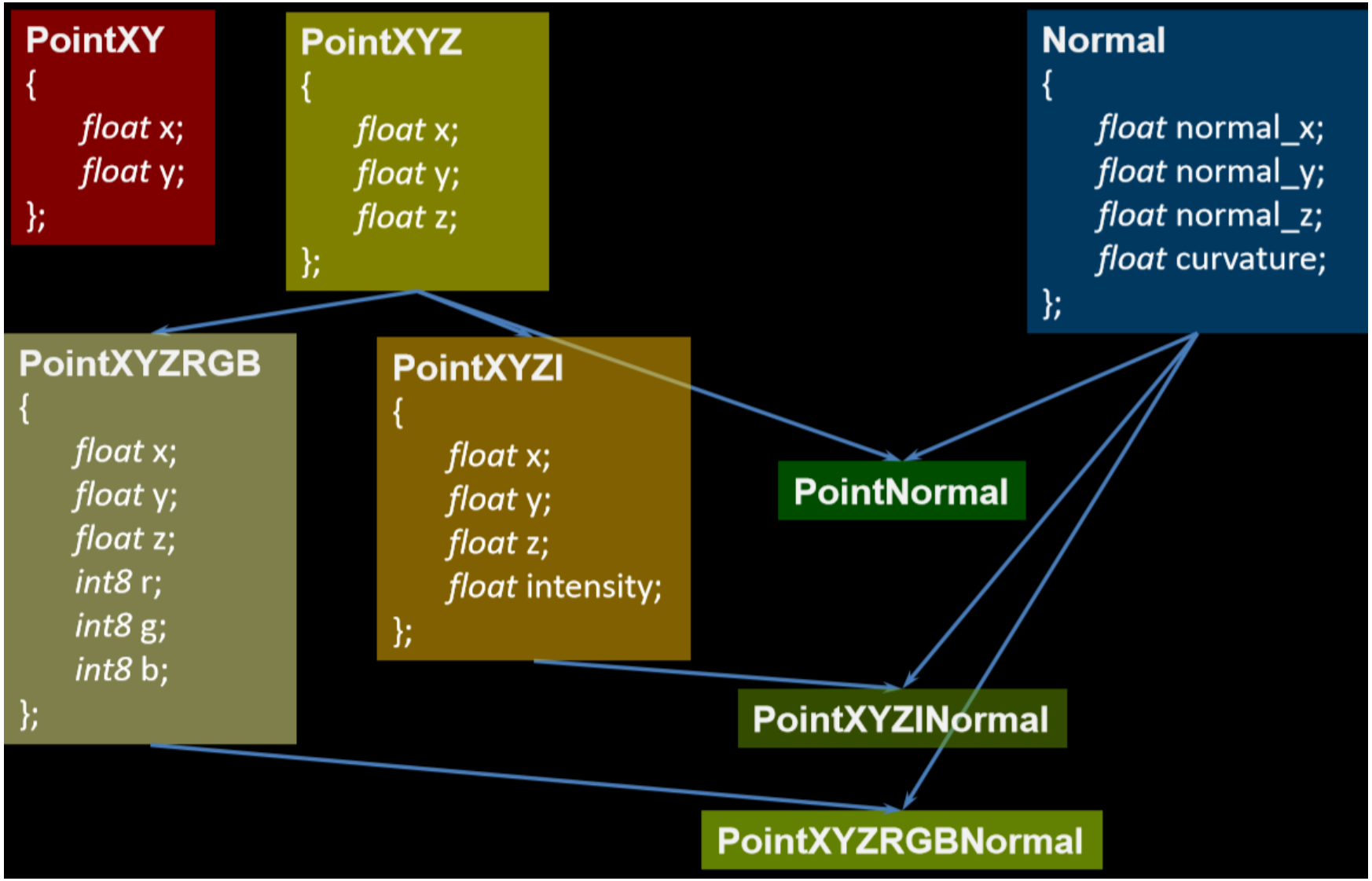




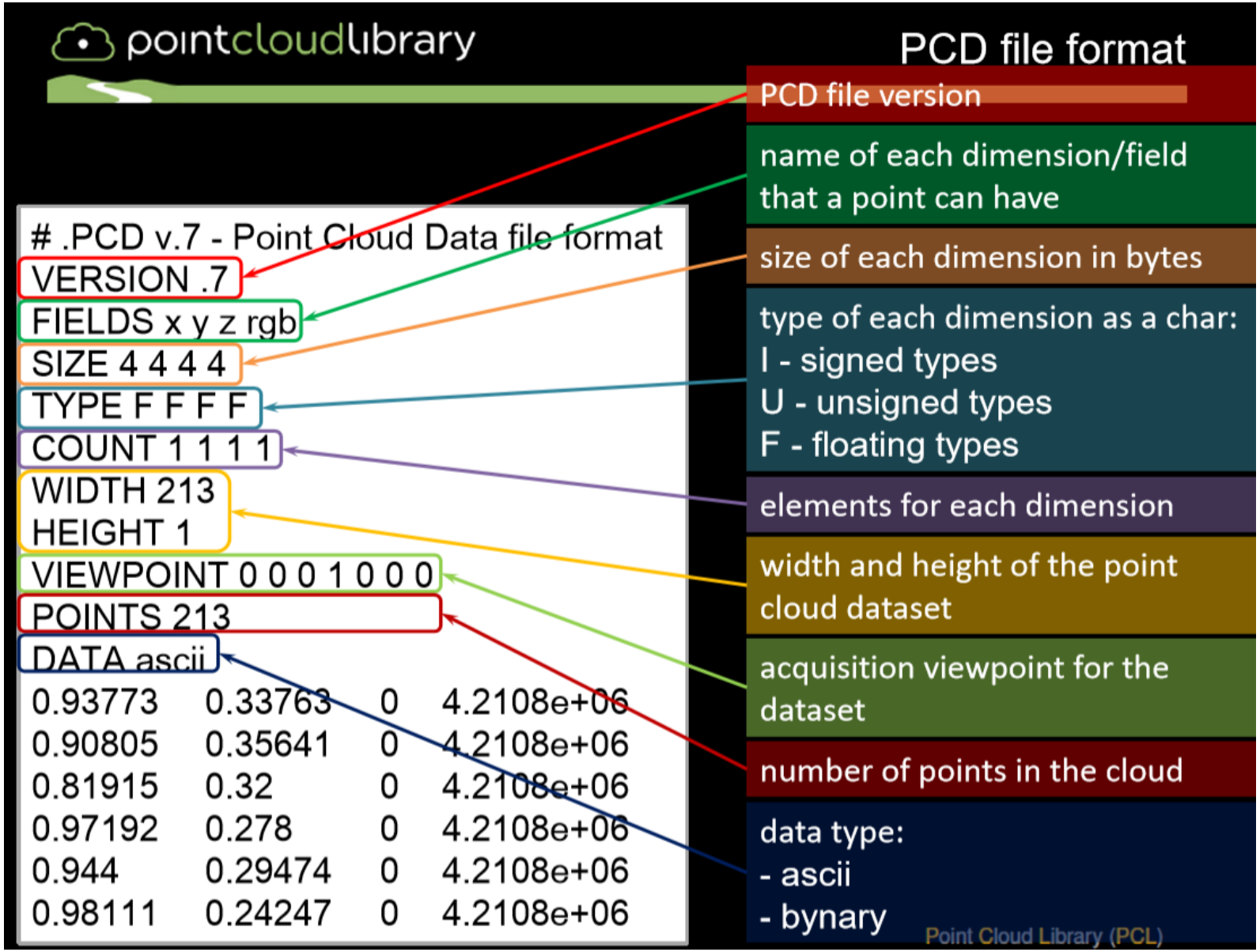


Optional

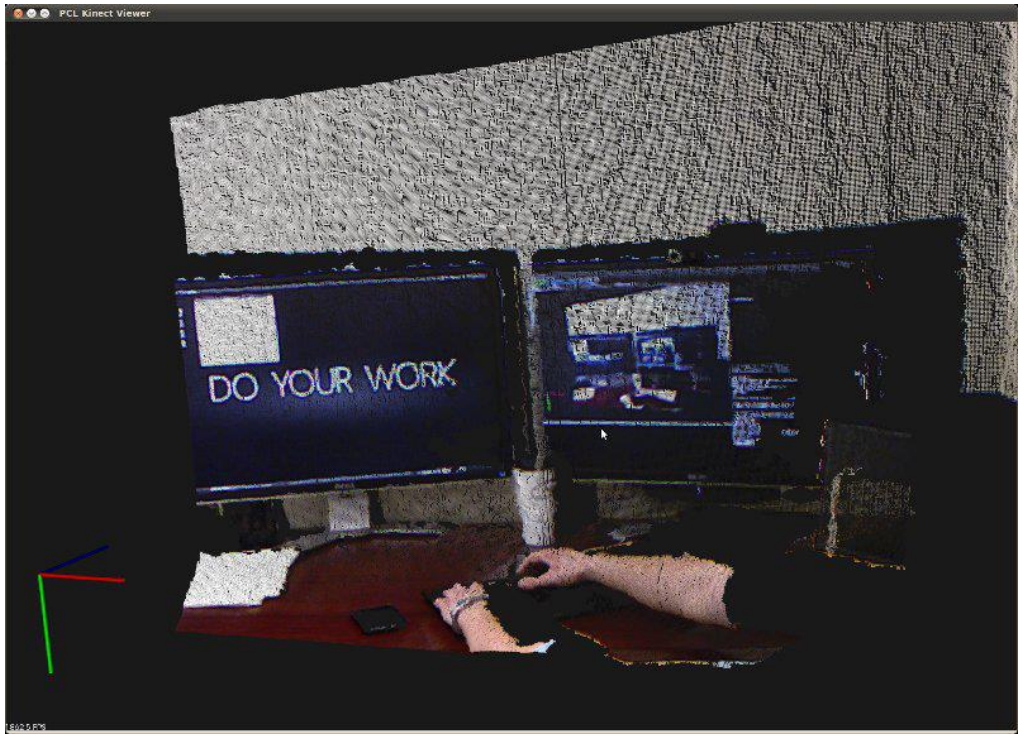
Basic structures



Basic structures



I/O

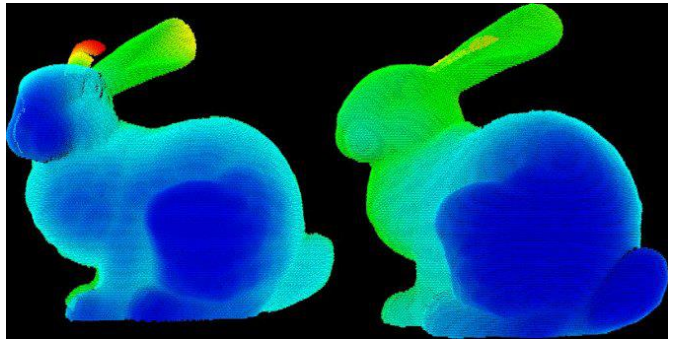


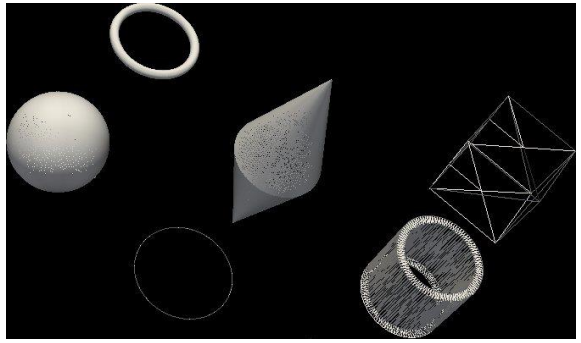
pcl::OpenNIGrabber

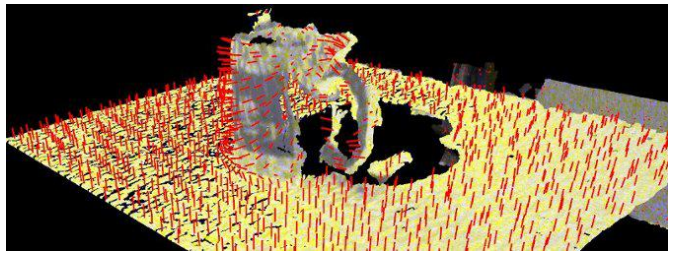
Documentation: http://docs.pointclouds.org/trunk/groupio.html

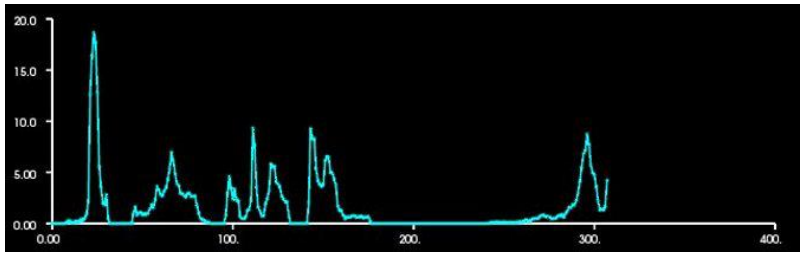
Tutorials: http://pointclouds.org/documentation/tutorials/#i-o

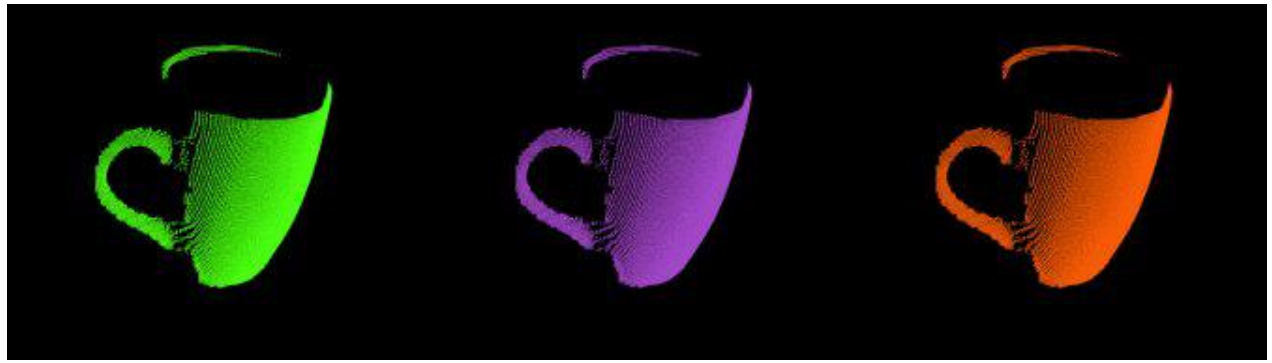
Visualization







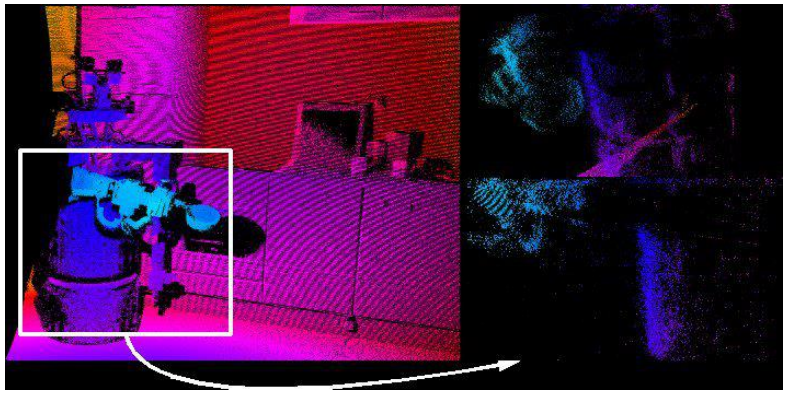


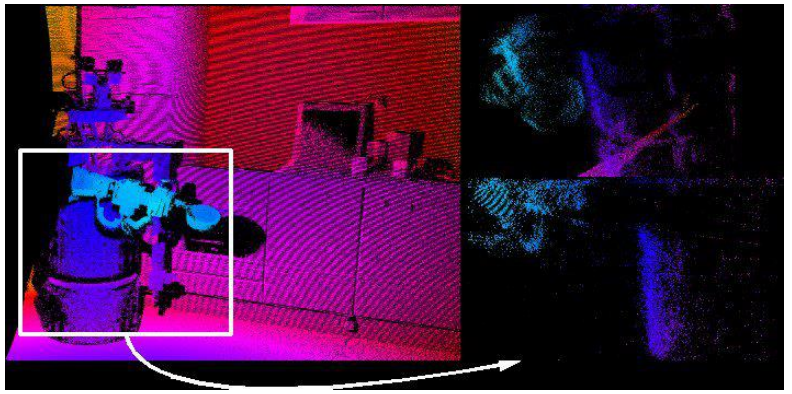


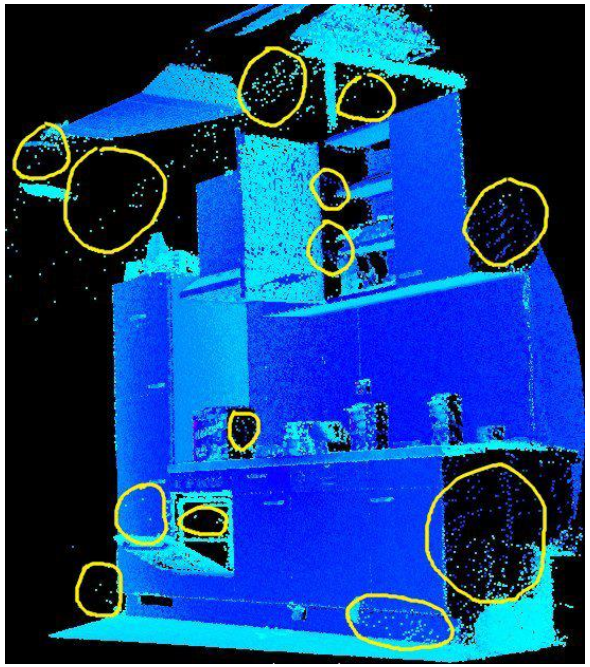
Documentation: http://docs.pointclouds.org/trunk/group__visualization.html
Tutorials: http://pointclouds.org/documentation/tutorials/#visualization-tutorial

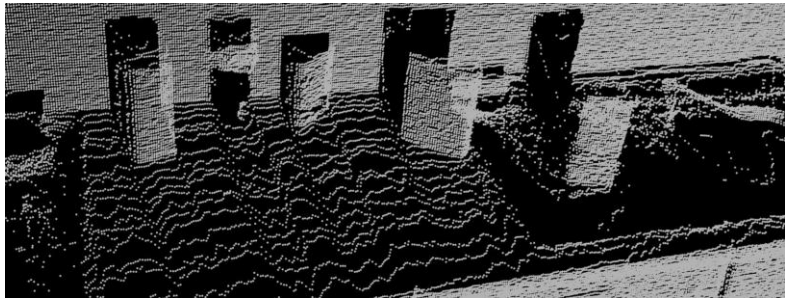
Filtering 1/6

- irregular density (2.5D)
- Occlusions
- massive amount of data
- noise









Filtering 2/6

- 用于点云数据修改或者属性修改
  - ✓ Removing Points:
    - Conditional Removal
    - Radius/Statistical Outlier Removal
    - Color Filtering
    - Passthrough
  - ✓ Downsampling:
    - Voxelgrid Filter
    - approximate Voxelgrid filtering
  - ✓ Modifying Other Point Attributes:
    - Contrast
    - Bilateral Filtering

Filtering 3/6

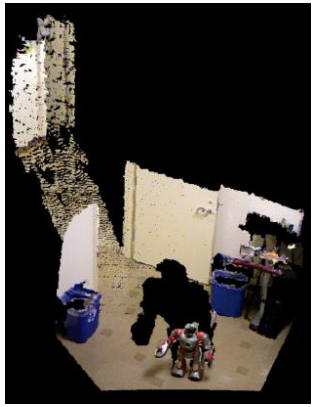
All filters are derived from the **Filter** base class with following interface:

```
template<typename PointT> class Filter : public PCLBase<PointT>
{
  public:
    Filter (bool extract_removed_indices = false);
    inline IndicesConstPtr const getRemovedIndices ();
    inline void setFilterFieldName (const std::string &field_name);
    inline std::string const getFilterFieldName ();
    inline void setFilterLimits (const double &limit_min, const double &limit_max);
    inline void getFilterLimits (double &limit_min, double &limit_max);
    inline void setFilterLimitsNegative (const bool limit_negative);
    inline bool getFilterLimitsNegative ();
    inline void filter (PointCloud &output);
};
```

Filtering 4/6

Example: Passthrough Filter

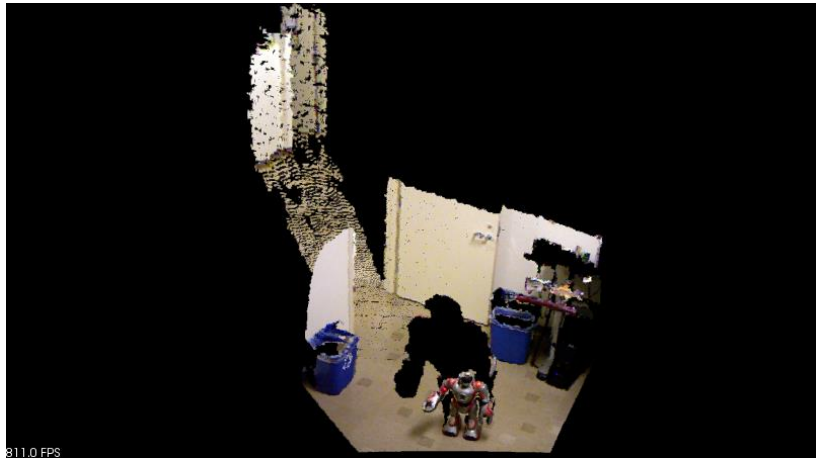
```
// point cloud instance for the result
PointCloudPtr thresholded (new PointCloud);

// create passthrough filter instance
pcl::PassThrough<PointT> pass_through;

// set input cloud
pass_through.setInputCloud (input);

// set fieldname we want to filter over
pass_through.setFilterFieldName ("z");

// set range for selected field to 1.0 - 1.5 meters
pass_through.setFilterLimits (1.0, 1.5);

// do filtering
pass_through.filter (*thresholded);
```

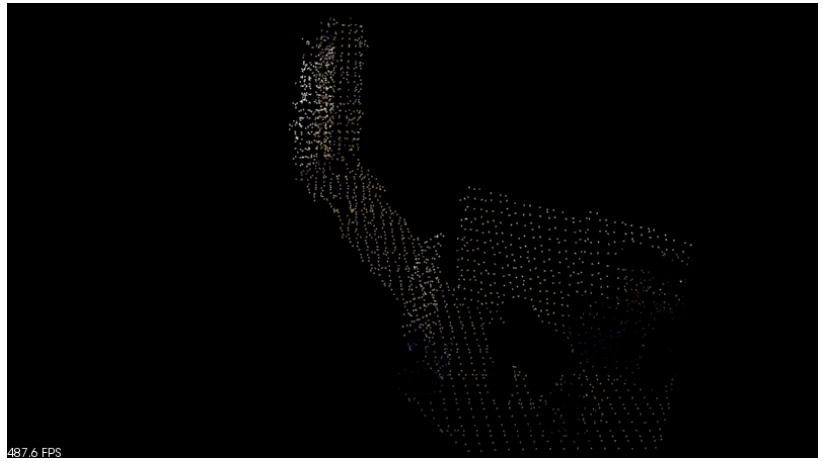


Filtering 5/6

Example: VoxelGrid Filter

```
// point cloud instance for the result
PointCloudPtr downsampled (new PointCloud);

// create passthrough filter instance
pcl::VoxelGrid<PointT> voxel_grid;

// set input cloud
voxel_grid.setInputCloud (input);

// set cell/voxel size to 0.1 meters in each dimension
voxel_grid.setLeafSize (0.1, 0.1, 0.1);

// do filtering
voxel_grid.filter (*downsampled);
```

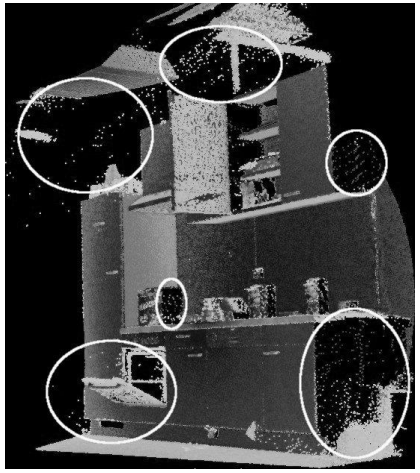


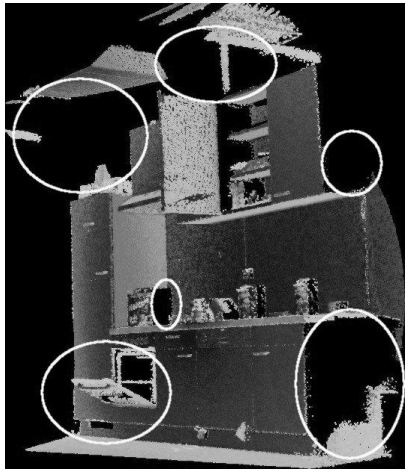


Filtering 6/6

Example: Radius Outlier Removal

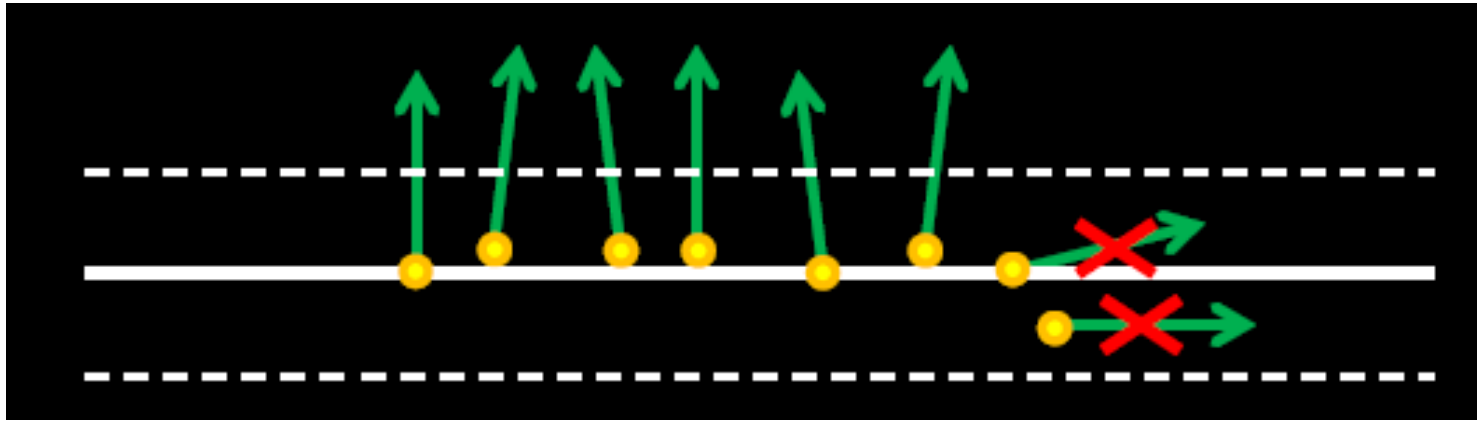
```
// point cloud instance for the result
PointCloudPtr cleaned (new PointCloud);

// create the radius outlier removal filter
pcl::RadiusOutlierRemoval<pcl::PointXYZRGB> radius_outlier_removal;

// set input cloud
radius_outlier_removal.setInputCloud (input);

// set radius for neighbor search
radius_outlier_removal.setRadiusSearch (0.05);

// set threshold for minimum required neighbors neighbors
radius_outlier_removal.setMinNeighborsInRadius (800);

// do filtering
radius_outlier_removal.filter (*cleaned);
```





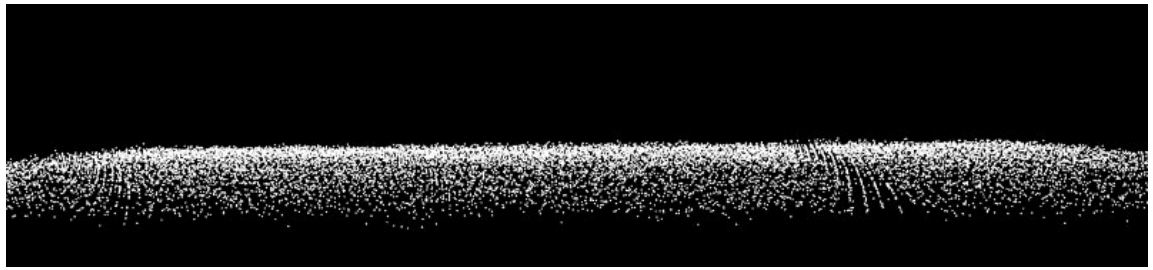
Normal Estimation 1/3

- Plane fitting can be supported by surface normals.



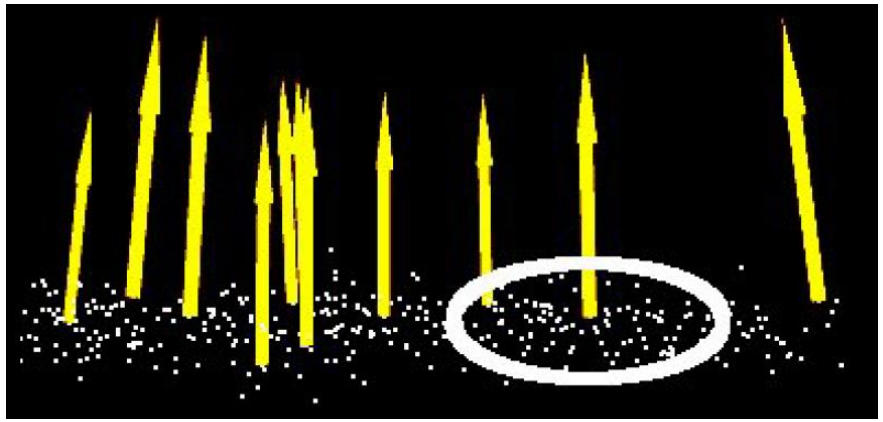
Normal Estimation 2/3

输入: point cloud $\mathcal{P}$ of 3D points $p = (x, y, z)^T$



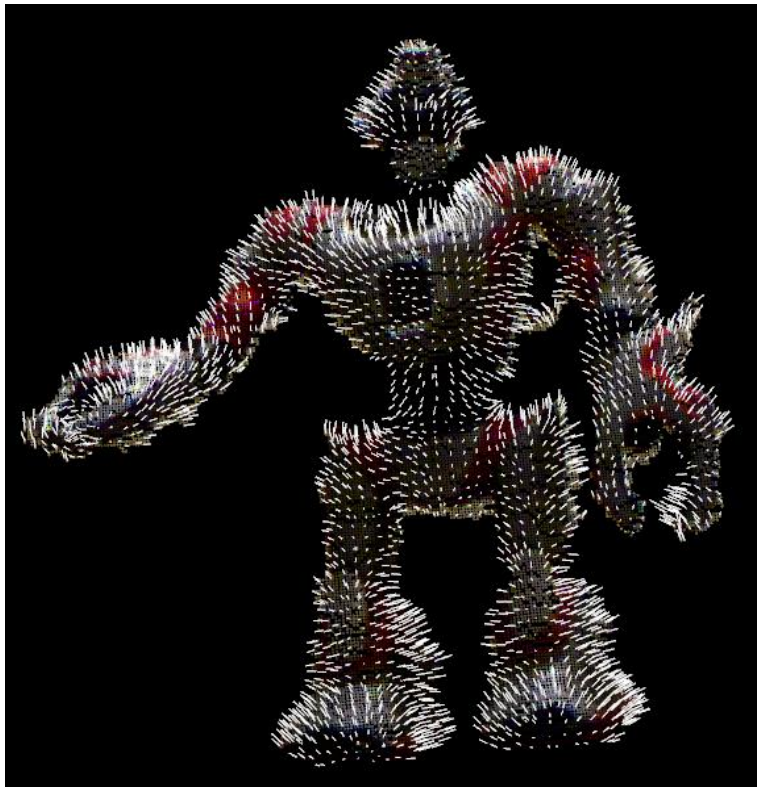
表面法向量计算:

1. Select a set of points $\mathcal{Q} \subseteq \mathcal{P}$ from the neighborhood of p.
2. Fit a local plane through $\mathcal{Q}$.
3. Compute the normal $\vec{n}$ of the plane.



Normal Estimation 3/3

```
pcl::PointCloud<pcl::Normal>::Ptr normals_out
  (new pcl::PointCloud<pcl::Normal>);

pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> norm_est;

// Use a FLANN-based KdTree to perform neighborhood searches
norm_est.setSearchMethod
  (pcl::KdTreeFLANN<pcl::PointXYZRGB>::Ptr
    (new pcl::KdTreeFLANN<pcl::PointXYZRGB>));

// Specify the size of the local neighborhood to use when
// computing the surface normals
norm_est.setRadiusSearch (normal_radius);

// Set the input points
norm_est.setInputCloud (points);

// Set the search surface (i.e., the points that will be used
// when search for the input points' neighbors)
norm_est.setSearchSurface (points);

// Estimate the surface normals and
// store the result in "normals_out"
norm_est.compute (*normals_out);
```



Segmentation 1/4

RANSAC (Random Sample Consensus) is a randomized algorithm for robust model fitting.

基本的迭代策略

- select sample set
- compute model
- compute and count inliers
- repeat until sufficiently confident

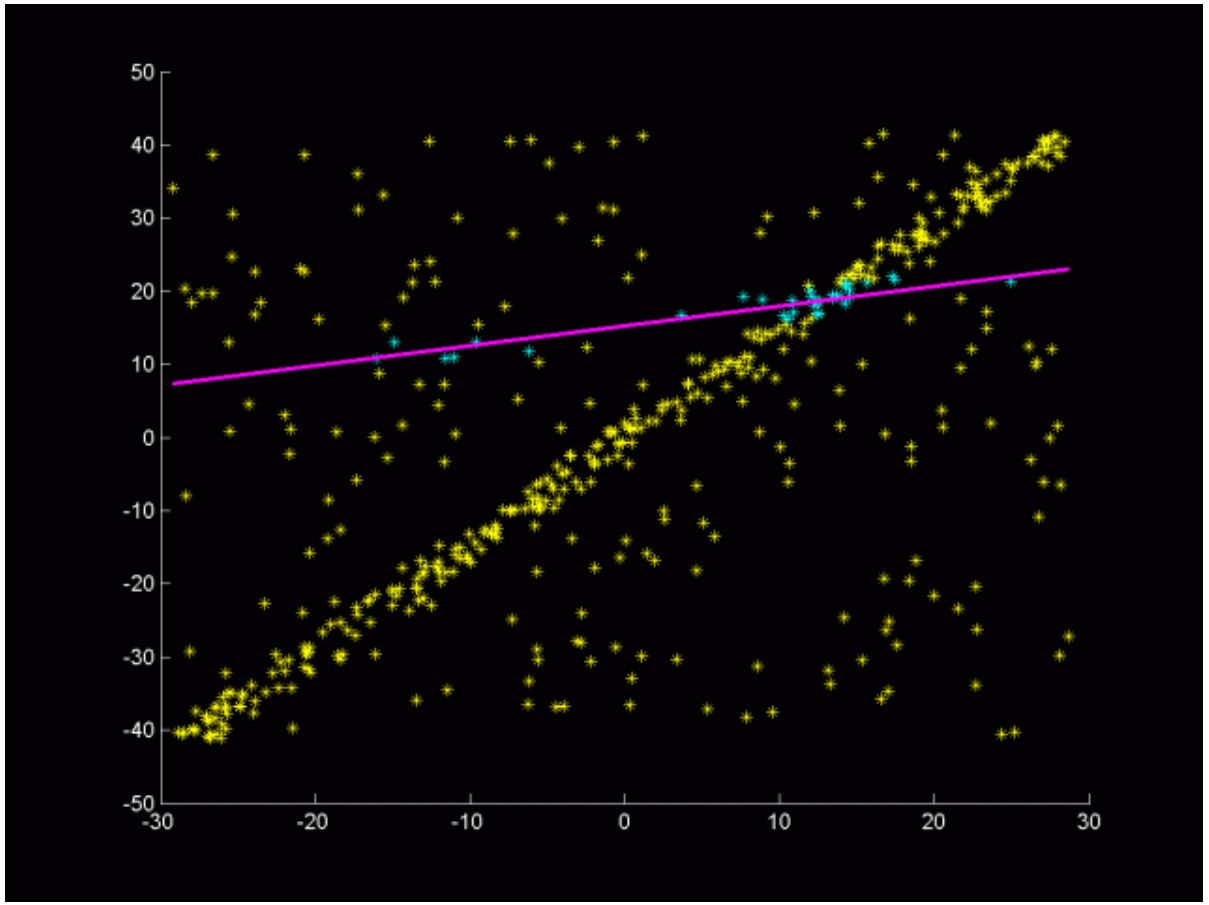
Segmentation 2/4

- **RANSAC (Random Sample Consensus)** is a randomized algorithm for robust model fitting.

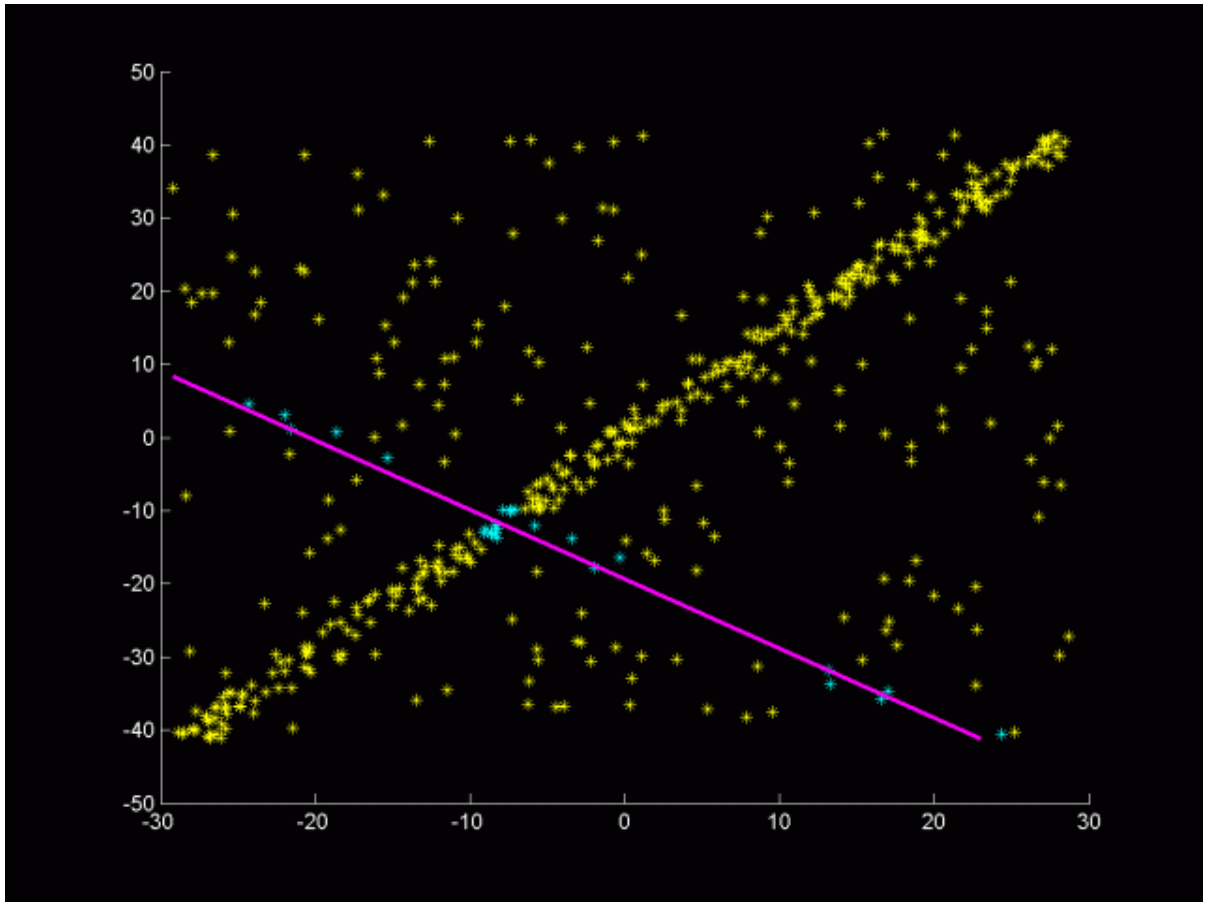
基本的迭代策略: **Line segmentation**

- select sample set **->2 points**
- compute model **-> line equation**
- compute and count inliers **-> errors**
- repeat until sufficiently confident **-> 95%**

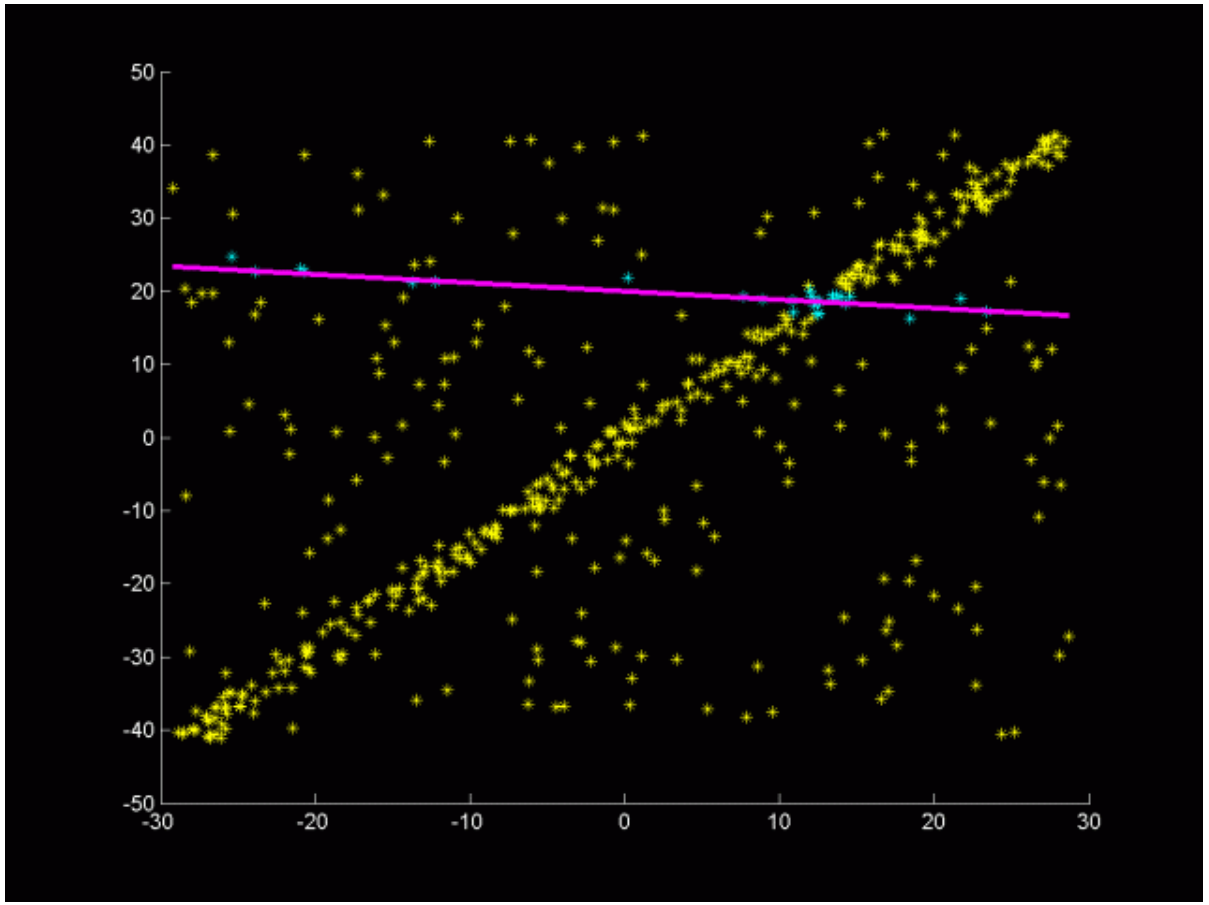
Segmentation 2/4



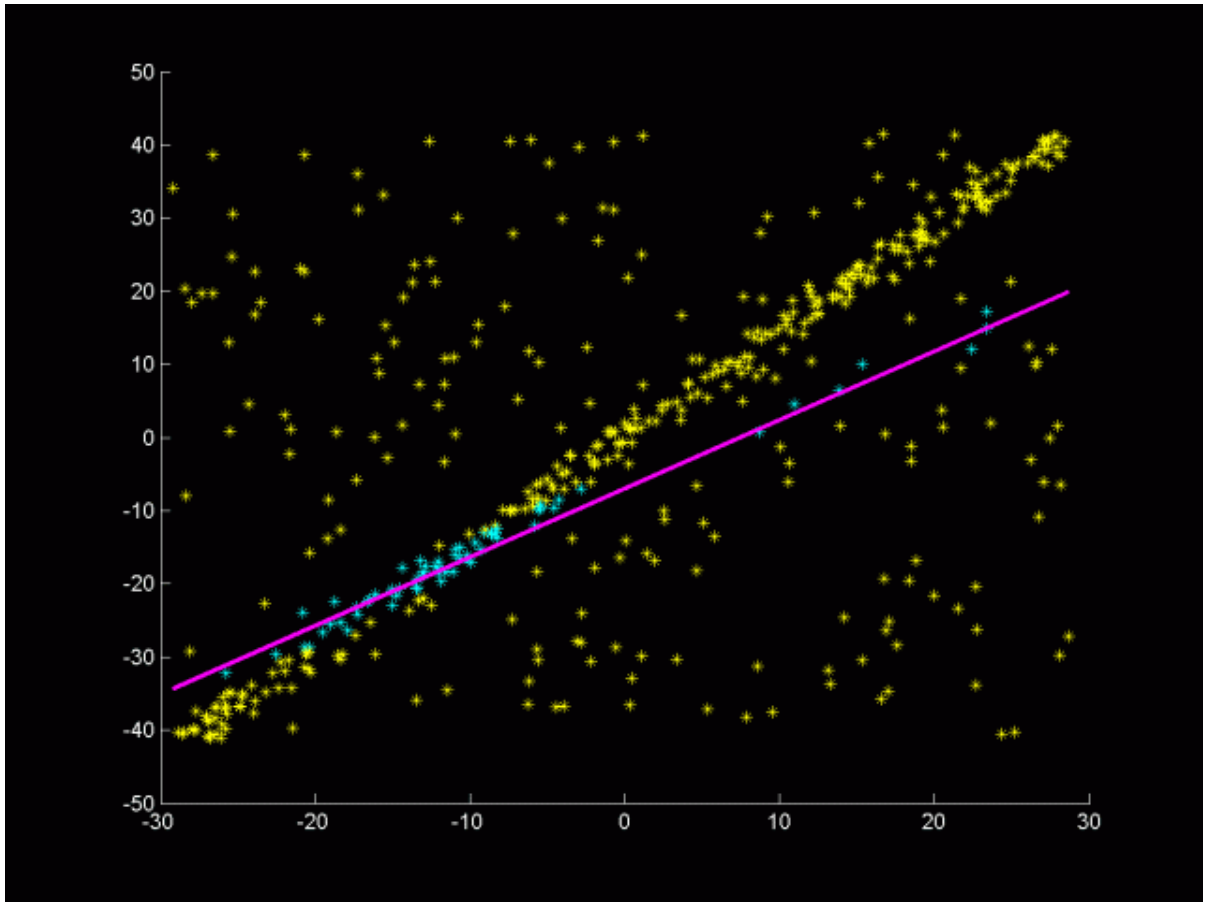
Segmentation 2/4



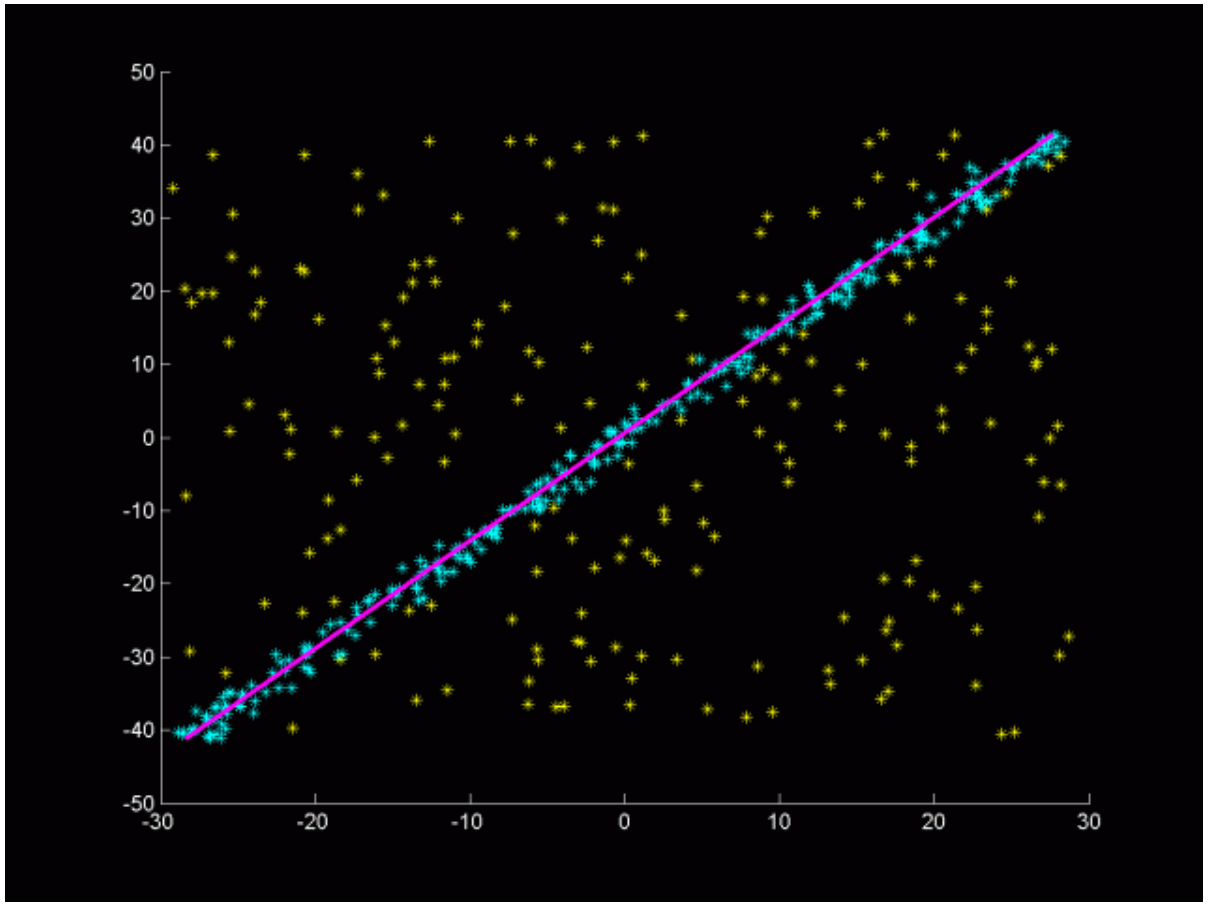
Segmentation 2/4



Segmentation 2/4



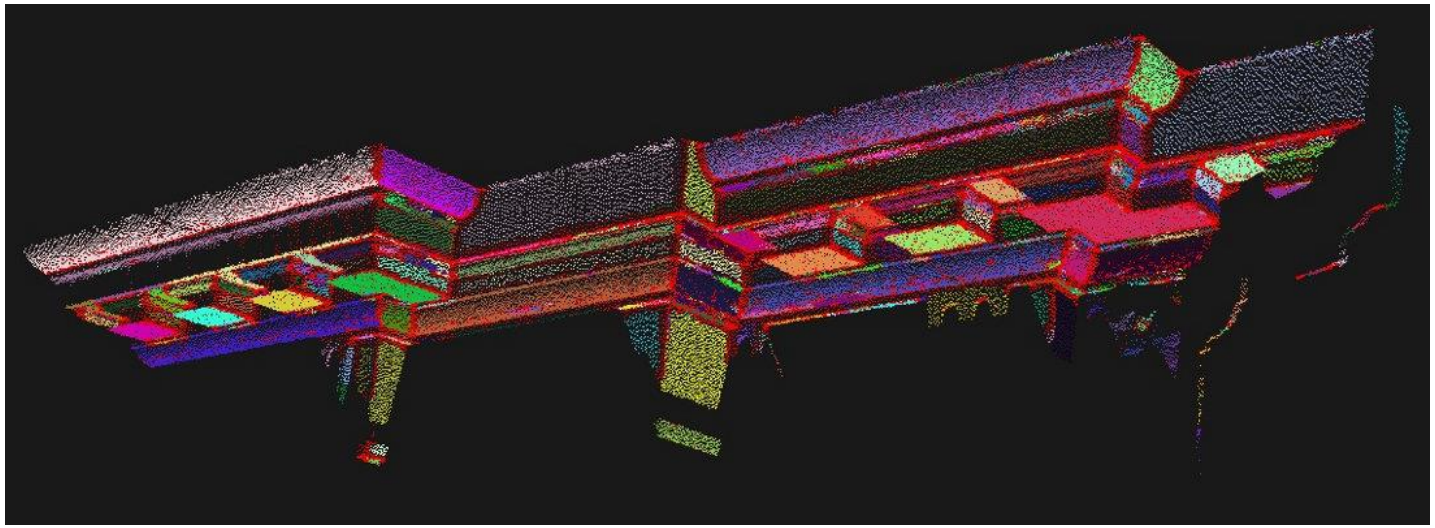
Segmentation 2/4



Segmentation 3/4

- **PCL对分割算法进行了扩展**
  - MSAC (weighted distances instead of hard thresholds)
  - MLESAC (Maximum Likelihood Estimator)
  - PROSAC (Progressive Sample Consensus)

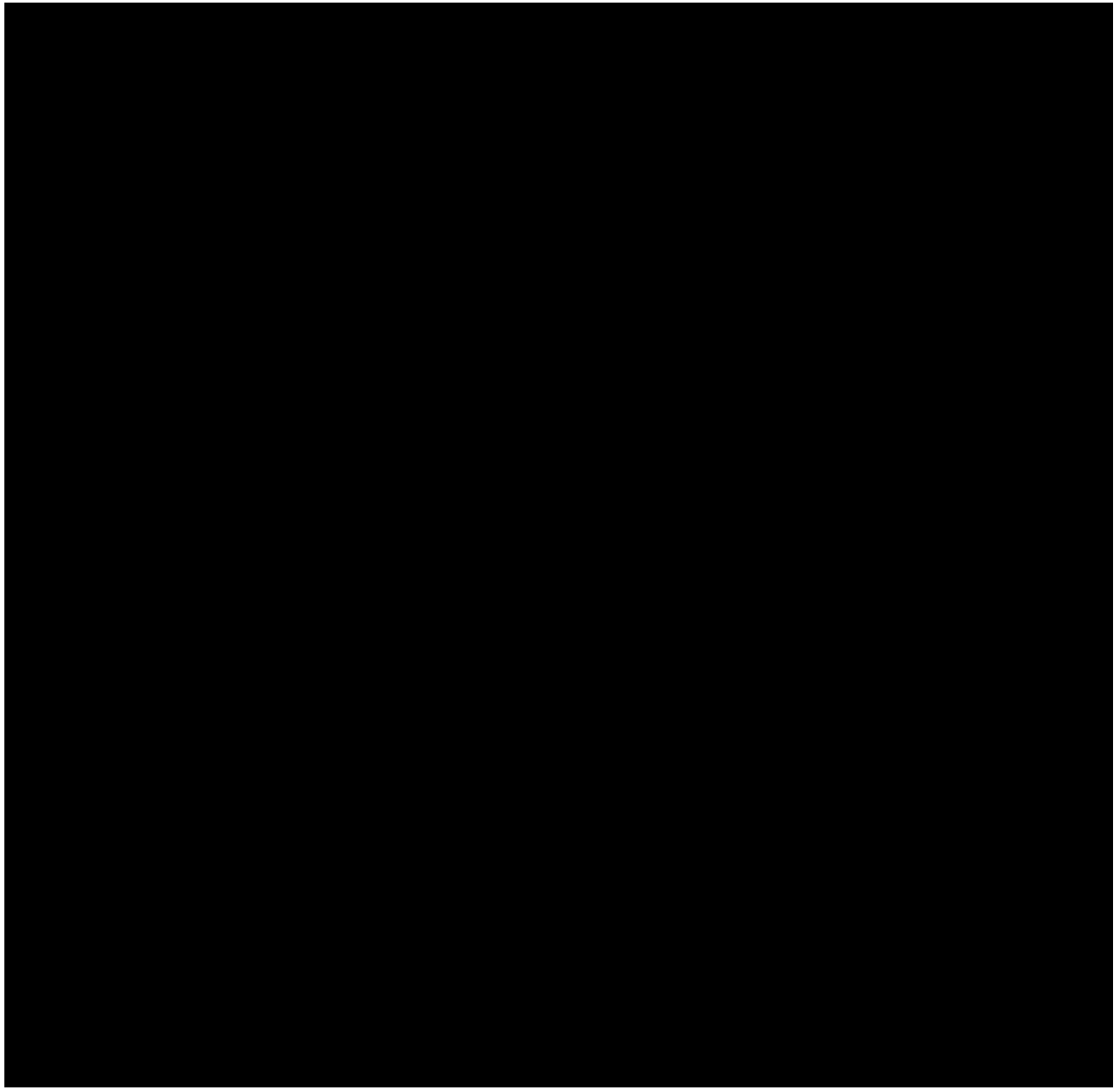
- **同时PCL可以对不同形状的物体进行分割**
  - Plane models (with constraints such as orientation)
  - Cone
  - Cylinder
  - Sphere
  - Line
  - Circle
  - …



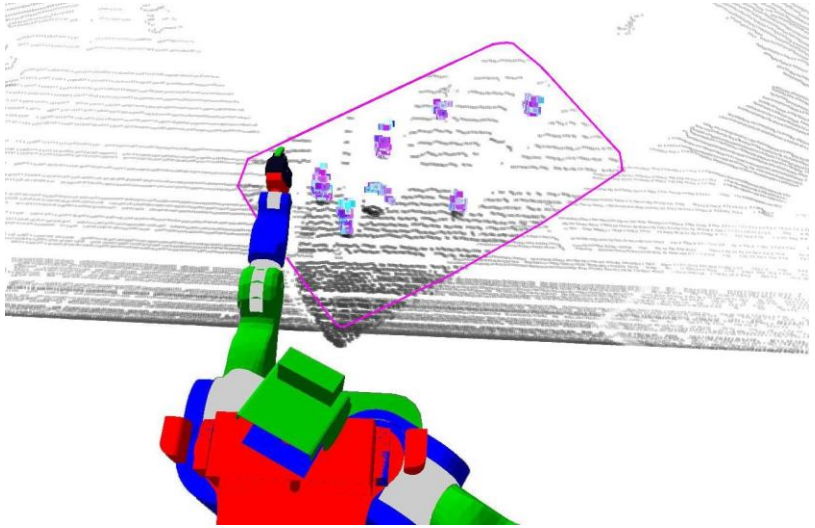
```
// necessary includes
#include <pcl/sample_consensus/ransac.h>
#include <pcl/sample_consensus/sac_model_plane.h>

// ...

// Create a shared plane model pointer directly
SampleConsensusModelPlane<PointXYZ>::Ptr model
  (new SampleConsensusModelPlane<PointXYZ> (input));

// Create the RANSAC object
RandomSampleConsensus<PointXYZ> sac (model, 0.03);

// perform the segmenation step
bool result = sac.computeModel ();
```

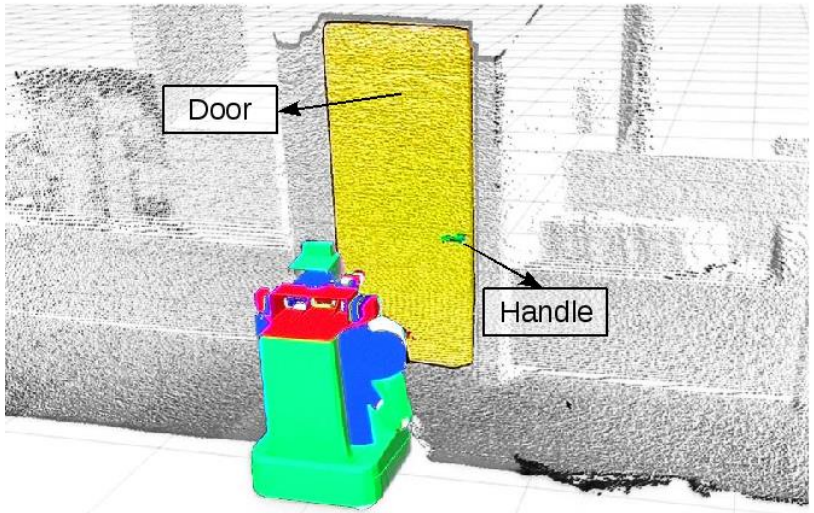
```
// get inlier indices
boost::shared_ptr<vector<int> > inliers (new vector<int>);
sac.getInliers (*inliers);
cout << "Found model with " << inliers->size () << " inliers";

// get model coefficients
Eigen::VectorXf coeff;
sac.getModelCoefficients (coeff);
cout << ", plane normal is: " << coeff[0] << ", " << coeff[1] << ", " << coeff[2] << "."
```

Segmentation 4/4



Clustering 1/3



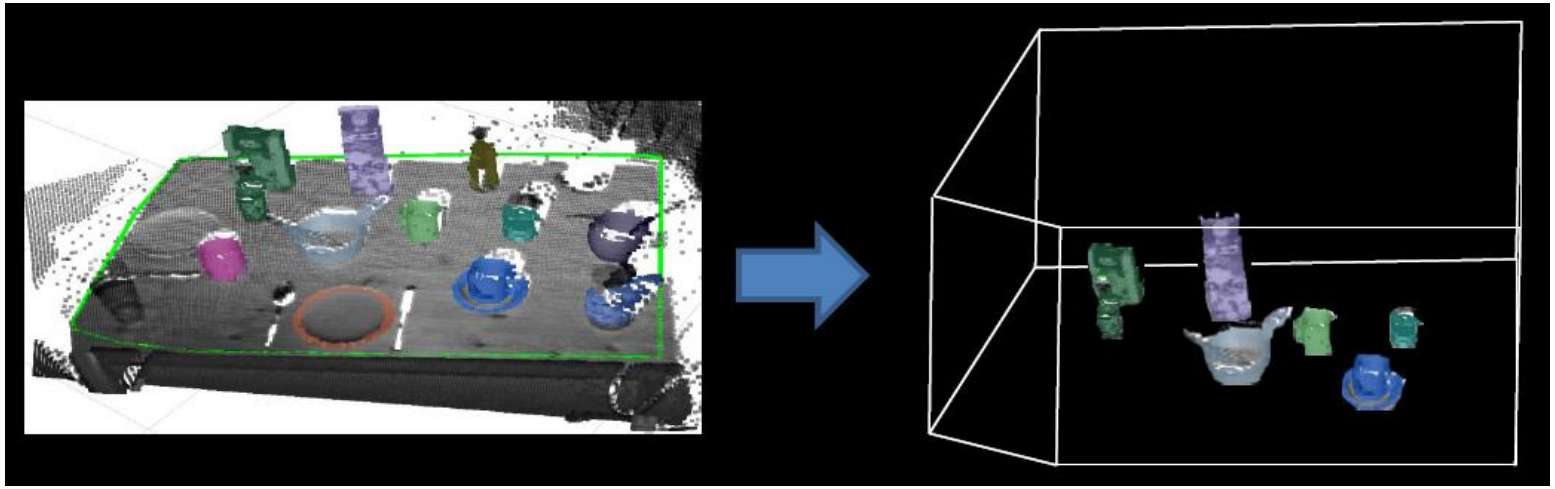


- 当我们将平面要素分割出来之后，我们可以获取到
  - 桌面或者柜子上的物件
  - 门上的门把手
- 怎么获取
  - 计算平面点的凸包
  - 将平面法方向点云筛选出来

Clustering 2/3

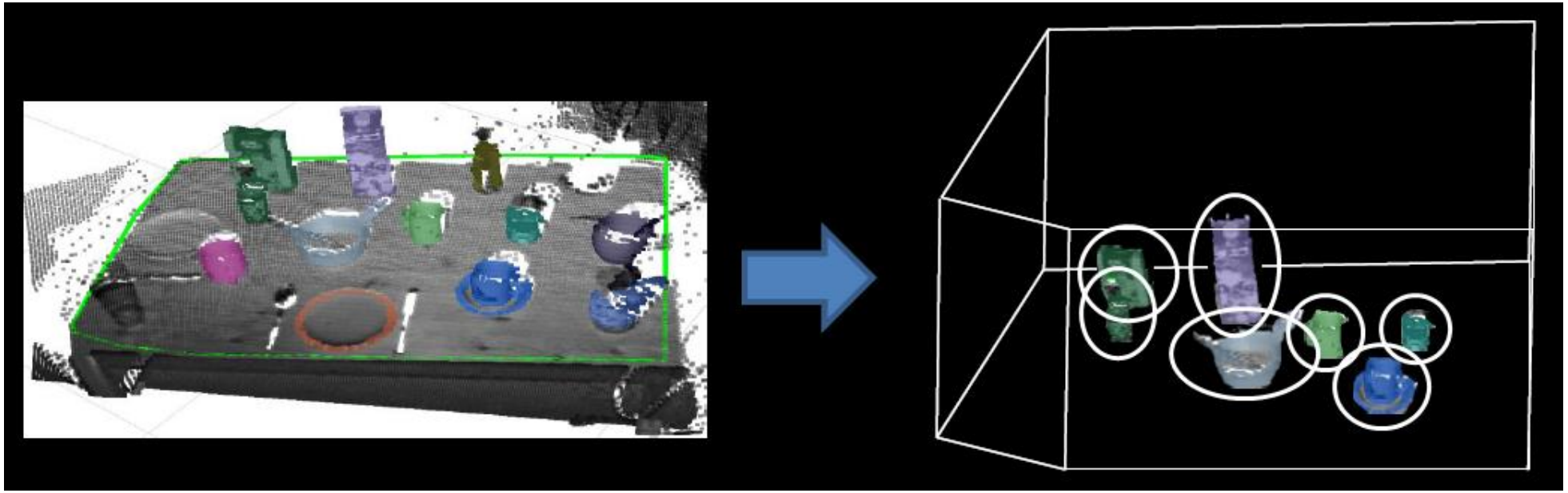
```
// Create a Convex Hull representation of the projected inliers
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_hull
  (new pcl::PointCloud<pcl::PointXYZ>);
pcl::ConvexHull<pcl::PointXYZ> chull;
chull.setInputCloud (inliers_cloud);
chull.reconstruct (*cloud_hull);
```

```
// segment those points that are in the polygonal prism
ExtractPolygonalPrismData<PointXYZ> ex;
ex.setInputCloud (outliers);
ex.setInputPlanarHull (cloud_hull);

PointIndices::Ptr output (new PointIndices);
ex.segment (*output);
```



Clustering 3/3

```
// Create EuclideanClusterExtraction and set parameters
pcl::EuclideanClusterExtraction<PointT> ec;
ec.setClusterTolerance (cluster_tolerance);
ec.setMinClusterSize (min_cluster_size);
ec.setMaxClusterSize (max_cluster_size);

// set input cloud and let it run
ec.setInputCloud (input);
ec.extract (cluster_indices_out);
```



练习

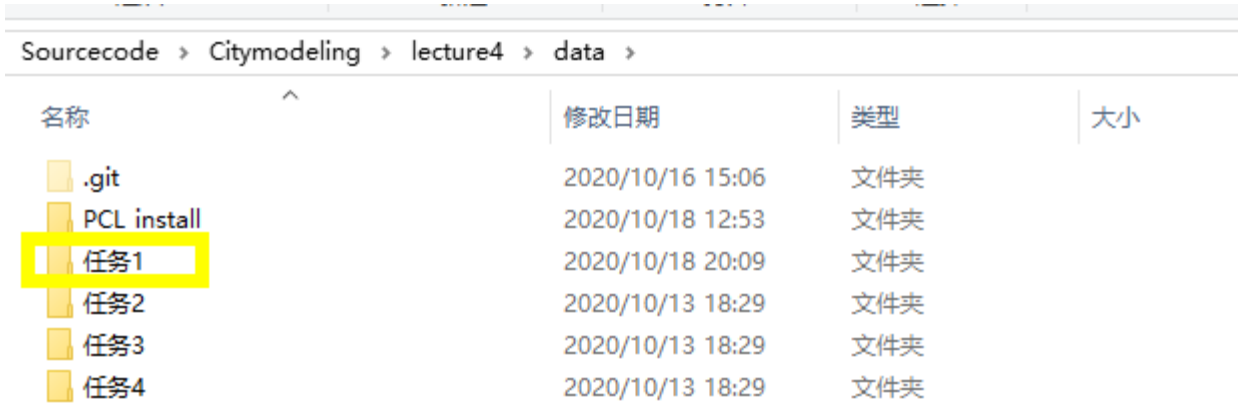
参考资料：https://pcl.readthedocs.io/projects/tutorials/en/latest/#i-o

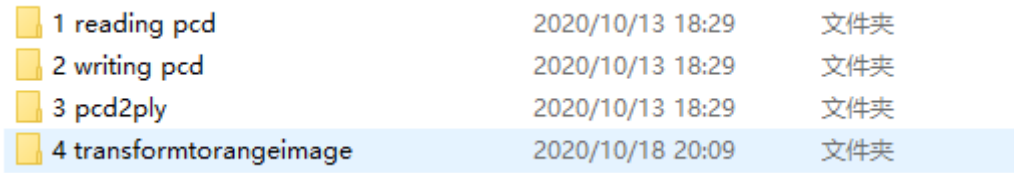
1、PCD数据的读取：读取PCD点云文件，并打印出所有的点X,Y,Z（参考任务1中reading pcd文件夹）

2、PCD数据的写入：随机创建一个点云序列，保存为PCD文件，并在cloudcompare中检查是否正确（参考任务1中writing pcd文件夹）

3、PCD转Rangeimage：读取PCD文件，将其转换为rangeimage，尝试在X，Y,Z三个方向上进行旋转再重新投影。（参考任务1中TranstoRangeimage文件夹）

4、PCD与PLY数据互相转换：读取PCD文件，将其转换为PLY数据格式。（参考任务1中3 pcd2ply 文件夹）





Example-PCD read

参考资料：https://pcl.readthedocs.io/projects/tutorials/en/latest/#i-o

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

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

  if (pcl::io::loadPCDFile<pcl::PointXYZ> ("test_pcd.pcd", *cloud) == -1) //* load the file
  {
    PCL_ERROR ("Couldn't read file test_pcd.pcd \n");
    return (-1);
  }
  std::cout << "Loaded "
            << cloud->width * cloud->height
            << " data points from test_pcd.pcd with the following fields: "
            << std::endl;
  for (const auto& point: *cloud)
    std::cout << "    " << point.x
              << " "    << point.y
              << " "    << point.z << std::endl;

  return (0);
}
```

creates a PointCloud<PointXYZ> boost shared pointer and initializes it.

```
1  #include <iostream>
2  #include <pcl/io/pcd_io.h>
3  #include <pcl/point_types.h>
4
5  int
6  main (int argc, char** argv)
7  {
8    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
9
10   if (pcl::io::loadPCDFile<pcl::PointXYZ> ("test_pcd.pcd", *cloud) == -1) //* load the file
11   {
12     PCL_ERROR ("Couldn't read file test_pcd.pcd \n");
13     return (-1);
14   }
15   std::cout << "Loaded "
16             << cloud->width * cloud->height
17             << " data points from test_pcd.pcd with the following fields: "
18             << std::endl;
19   for (const auto& point: *cloud)
20     std::cout << "    " << point.x
21               << " "    << point.y
22               << " "    << point.z << std::endl;
23
24   return (0);
25 }
```

loads the PointCloud data from disk (we assume that test_pcd.pcd has already been created from the previous tutorial) into the binary blob.

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

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

  if (pcl::io::loadPCDFile<pcl::PointXYZ> ("test_pcd.pcd", *cloud) == -1) //* load the file
  {
    PCL_ERROR ("Couldn't read file test_pcd.pcd \n");
    return (-1);
  }
  std::cout << "Loaded "
            << cloud->width * cloud->height
            << " data points from test_pcd.pcd with the following fields: "
            << std::endl;
  for (const auto& point: *cloud)
    std::cout << "    " << point.x
              << " "    << point.y
              << " "    << point.z << std::endl;

  return (0);
}
```

show the data that was loaded from file.

谢谢