从零开始一起学习SLAM | 给点云加个滤网

小白：师兄，上次你讲了点云拼接后，我回去费了不少时间研究，终于得到了和你给的参考结果差不多的点云，不过，这个点云“可远观而不可近看”，放大了看就只有一个个稀疏的点了。究竟它能干什么呢？

师兄：这个问题嘛。。。基本就和SLAM的作用一样，定位和建图
小白：定位好理解，可是师兄说建图，这么稀疏的地图有什么用呢？
师兄：地图分很多种，稀疏的，稠密的，还有半稀疏的等，你输出的这个稀疏的地图放大了看就是一个个离散的空间点，不过我们可以把它变成连续的稠密的网格，这个过程也叫点云的网格化
小白：哇塞，听起来好高大上呢，具体怎么做呢？
师兄：点云网格化需要对点云进行一系列处理，今天我们先说说点云处理流程的第一步，叫做点云滤波

为什么要对点云滤波？

小白：滤波是什么鬼？
师兄：滤波最早来自在数字信号处理里的概念，你可以理解为是一个过滤器，是对点云的一种预处理方法
小白：哦哦，想起来中学学的滤纸，就是可以过滤掉杂质那种
师兄：哈哈，对，这个很形象了，你看下面这个图，左侧就是原来的点云，右侧是经过滤波后滤掉的“杂质”

小白：师兄，那是所有的点云一开始都要滤波吗？
师兄：如果你的点云本来就非常好了，就不需要了。一般下面这几种情况需要进行点云滤波处理：
（1） 点云数据密度不规则需要平滑
（2） 因为遮挡等问题造成离群点需要去除

（3） 大量数据需要下采样
（4） 噪声数据需要去除

小白：前三点还能勉强理解，这第四点中点云中噪声数据从哪里来的呢？
师兄：这个很多因素啦！
一方面来自设备。比如我们用激光扫描仪、RGB-D相机等设备获取点云数据时，由于设备精度，电磁波的衍射特性等都会引入噪声的。
另一方面来自环境因素带来的影响，比如被测物体表面性质发生变化。
还有一个重要的方面就是操作者经验带来的影响，比如在处理点云数据拼接配准等操作过程中引入的一些噪声等。

小白：嗯嗯，原来噪声数据这么容易混进来啊。那怎么去掉他们呢？或者说怎么样滤波呢？说滤波好像显得更专业一点哈
师兄：点云中的噪声点对后续操作的影响比较大。就像盖房子一样，地基有很多瑕疵，如果不加以处理最终可能会导致整个房子坍塌的。不过别担心，PCL中有一个专门的点云滤波模块，可以将噪声点去除，还可以进行点云压缩等操作，非常灵活实用，例如：双边滤波，统计滤波，条件滤波，随机采样一致性滤波等。这样才能够更好的进行配准，特征提取，曲面重建，可视化等后续应用处理。
小白：那太好啦，PCL都帮我们想到啦，我迫不及待的想要实践一下啦。具体怎么操作呢？
师兄：滤波模块主要是调用一些封装好的滤波函数，然后根据需要设定一下参数，还是很直观的。

一般来说，滤波对应的方案有如下几种：
（1）按照给定的规则限制过滤去除点
（2） 通过常用滤波算法修改点的部分属性
（3）对数据进行下采样

小白：哦哦，这么多函数啊，哪里有这个滤波函数大全呢？方便我需要的时候去查查用哪个的那种？

师兄：有的，PCL中关于点云滤波的所有函数都在这里：
http://docs.pointclouds.org/trunk/group__filters.html
下面我们举两个例子介绍一下，简单的熟悉一下滤波的过程

点云下采样

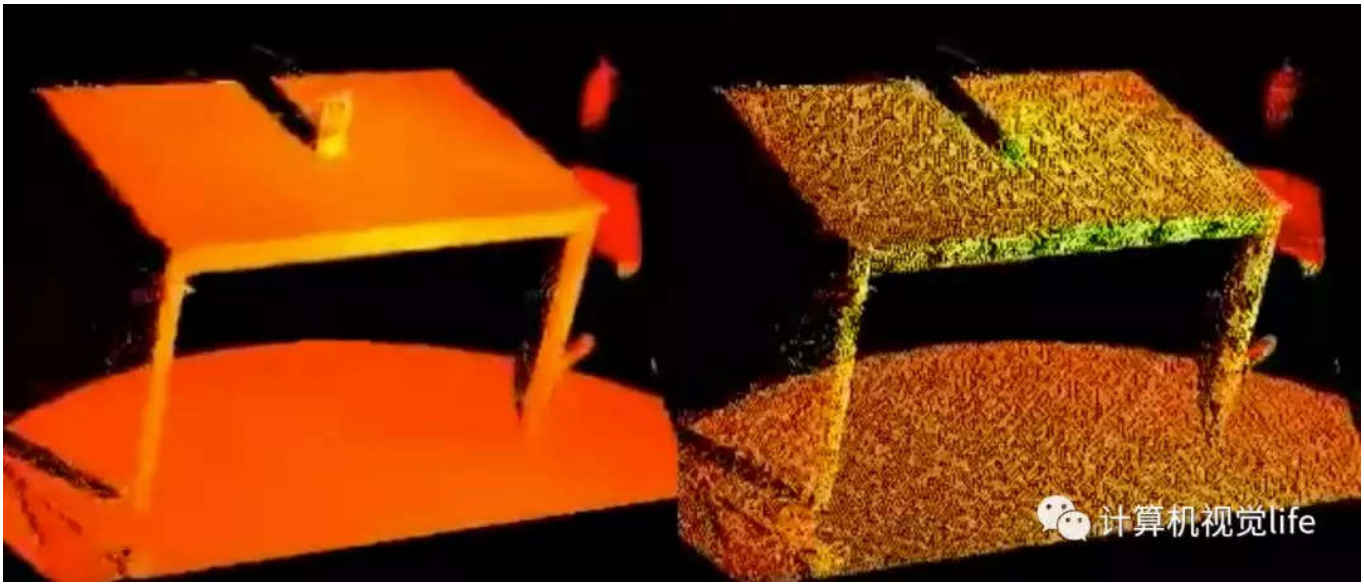
师兄：我先说一下数据下采样吧，这个最简单
小白：师兄，能不能问下为啥要下采样？我有强迫症，每做一件事情前都想知道原因。。。
师兄：嗯，这个不算强迫症啦，是个好习惯！了解原因了就知道什么时候用嘛！我举个例子，比如我们上次点云融合，一张640x480 的Depth图，假如每个地方都有深度值，可以转化为30万个

点组成的点云，如果有几十张上百张图这样暴力融合，那这个融合的点云会越来越大，储存、操作都是个大问题！

小白：是不是相当于我有个大容量的样本，我按一定的规则从里面抽取有代表性的样本，可以代替原来的样本，是这样吗？
师兄：对，理解的很到位。这个下采样PCL中有专门的类，叫做

```
class  pcl::ApproximateVoxelGrid< PointT >
```

它比较适合对海量的点云在处理前进行数据压缩，就像我们上次讲的点云融合后的数据那样，而且可以在特征提取等处理中选择合适的体素（voxel）大小等参数，提高算法效率。该函数对输入的点云数据创建一个三维体素栅格，每个体素内用体素中所有点的重心来近似显示体素中其他点，这样该体素内所有点都用一个重心点最终表示。它的优点是可以在下采样的时候保存点云的形状特征。
看下面是它的结果，左边是待处理的右边是处理前，右边是处理后



小白：看起来确实形状保持的挺好，这个怎么编程实现？

师兄：其实关键就下面几行代码

```
pcl::VoxelGrid<PointT> downSampled;  //创建滤波对象
downSampled.setInputCloud (cloud);            //设置需要过滤的点云给滤波对象
downSampled.setLeafSize (0.01f, 0.01f, 0.01f);  //设置滤波时创建的体素体积为1cm的立方体
downSampled.filter (*cloud_downSampled);           //执行滤波处理，存储输出
```

每行代码的意义都注释好了，应该不难理解
小白：看起来好像也不麻烦
师兄；嗯，上面只是经常用的成员函数，还有一些其他成员函数，我这里只重点介绍两个
一个是

```
setLeafSize( float lx, float ly, float lz)
```

setLeafSize后面的三个参数表示体素栅格叶大小，分别表示体素在XYZ方向的尺寸，以米为单位，上面就是设置为长宽高都为1cm
另外一个是设置是否对所有的字段进行下采样，成员函数为

```
setDownsampleAllData(bool downsample)
```

小白：师兄，这个字段是啥意思?
师兄：哦，不好意思，忘记解释了。我们知道点云有不同的类型，比如有的是 PointXYZ，有的是PointXYZRGB，还有其他类型，也就是一个点包含多种不同信息，比如空间位置XYZ，颜色信息RGB，或者强度信息等，如果想要对所有信息（字段）下采样则设置为true，只对XYZ下采样的话设置为false

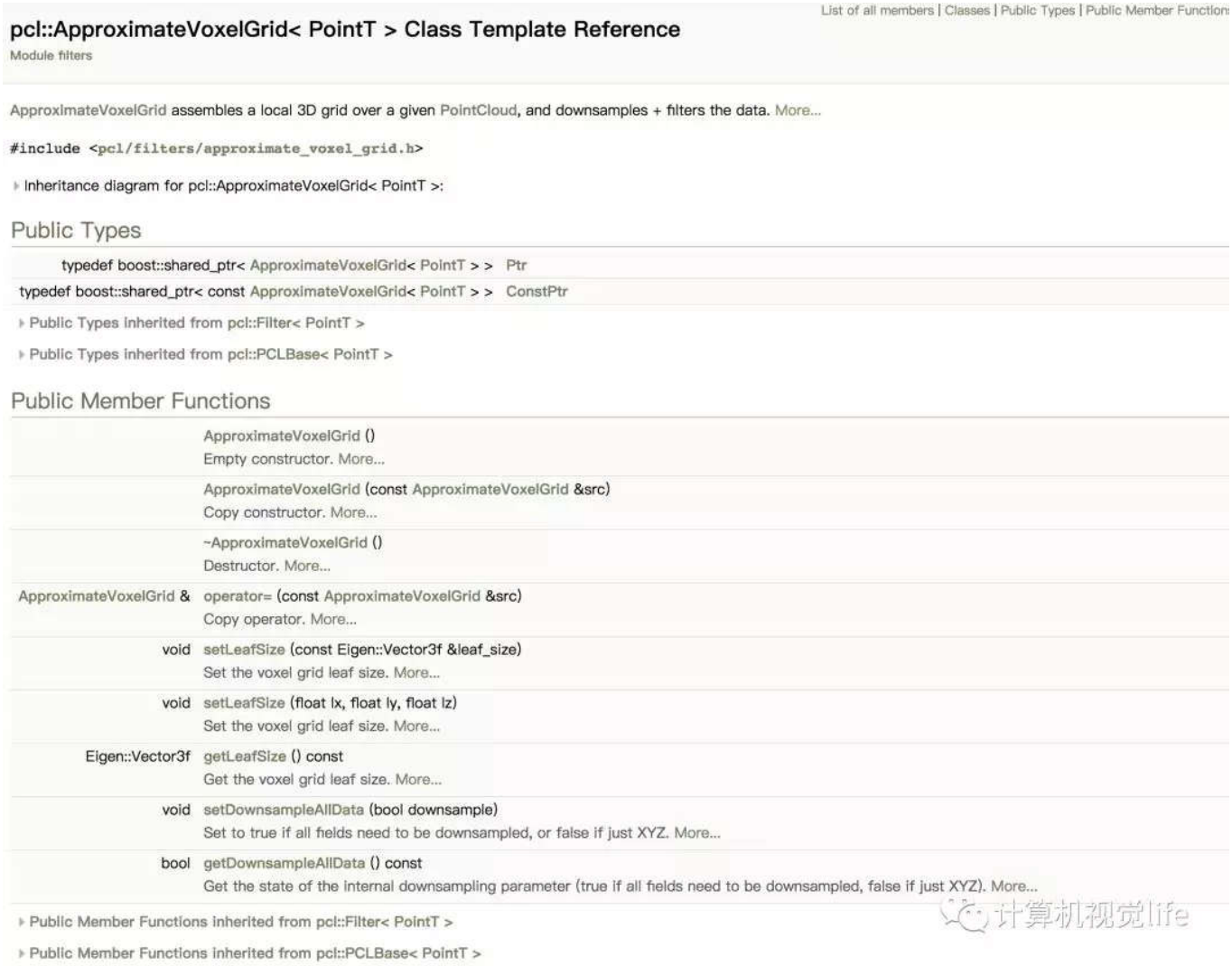
如何充分了解每个类的功能?

小白：嗯，了解啦，不过我有个问题，我想要用这个下采样类时怎么知道它有哪些函数可以用呢?
师兄：这个好问题，PCL官网上一般都有例程，但是例程包含的成员函数是很有限的，所以如果你想要了解这个滤波模板类的所有功能，或者说内联成员函数的话，最好的办法就是去官网查询。我们还是以pcl::ApproximateVoxelGrid 模板类为例进行说明

第一步，登录PCL API documentation http://docs.pointclouds.org/trunk/

第二步：在右上角搜索框内输入我们要查询的模板类名称，这里我们输入ApproximateVoxelGrid，如下所示，会自动跳出来匹配的结果

第三步：选择你要找的类名，点击进入，就到了下面这个界面，列出了所有的成员变量和成员函数，点击每个成员函数，会跳到对应的解释界面



pcl::ApproximateVoxelGrid< PointT > Class Template Reference

List of all members | Classes | Public Types | Public Member Function

Module filters

ApproximateVoxelGrid assembles a local 3D grid over a given PointCloud, and downsamples + filters the data. More...

#include <pcl/filters/approximate_voxel_grid.h>

Inheritance diagram for pcl::ApproximateVoxelGrid< PointT >:

Public Types

typedef boost::shared_ptr< ApproximateVoxelGrid< PointT > > Ptr

typedef boost::shared_ptr< const ApproximateVoxelGrid< PointT > > ConstPtr

Public Types inherited from pcl::Filter< PointT >

Public Types inherited from pcl::PCLBase< PointT >

Public Member Functions

ApproximateVoxelGrid ()
Empty constructor. More...

ApproximateVoxelGrid (const ApproximateVoxelGrid &src)
Copy constructor. More...

~ApproximateVoxelGrid ()
Destructor. More...

ApproximateVoxelGrid & operator= (const ApproximateVoxelGrid &src)
Copy operator. More...

void setLeafSize (const Eigen::Vector3f &leaf_size)
Set the voxel grid leaf size. More...

void setLeafSize (float lx, float ly, float lz)
Set the voxel grid leaf size. More...

Eigen::Vector3f getLeafSize () const
Get the voxel grid leaf size. More...

void setDownsampleAllData (bool downsample)
Set to true if all fields need to be downsampled, or false if just XYZ. More...

bool getDownsampleAllData () const
Get the state of the internal downsampling parameter (true if all fields need to be downsampled, false if just XYZ). More...

Public Member Functions inherited from pcl::Filter< PointT >

Public Member Functions inherited from pcl::PCLBase< PointT >

你看除了我们前面提到的几个成员函数，其实还有好几个我们没用到的都在这里了，需要的话可以使用啦，就是这样简单

小白：授人以鱼不如授人以渔，谢谢师兄！以后我可以自己查函数啦

去除点云的离群点

师兄：刚才下采样只是万里长征第一步，下面说一下去除离群点方法。
小白：等下，师兄，什么是离群点啊？
师兄：哦，忘了解释这个术语了，抱歉，我简单说一下，离群点对应的英文是outliers，也叫外点，就是明显偏离“群众”的点，比如我们用激光扫描一面平坦的墙壁，正常情况下得到的应该是差不多也位于同一个平面的点云，但是由于设备测量误差等原因，会产生少量脱离群众的空间点，离本来的墙壁过远，我们就叫这部分点为离群点。

小白：哈哈，离群点就是脱离群众的坏点，明白啦！不过这些点不是很少吗？有必要赶尽杀绝吗？
师兄：“赶尽杀绝”，很形象的比喻！哈哈，还是很有必要的，因为离群点会使局部点云特征(如表面法线或曲率变化)的估计复杂化，从而导致错误的值，从而可能导致点云配准失败。而且这些离群点还会随着积累进行传导，不早点消灭会有很大隐患的。

小白：原来危害这么大，那师兄赶快告诉我该怎样消灭它们吧！
师兄：好，我这里列举两个常用的去除离群点的类，第一个类叫做 StatisticalOutlierRemoval，顾名思义，使用统计分析技术，从一个点云数据中集中移除测量噪声点，也就是离群点啦！

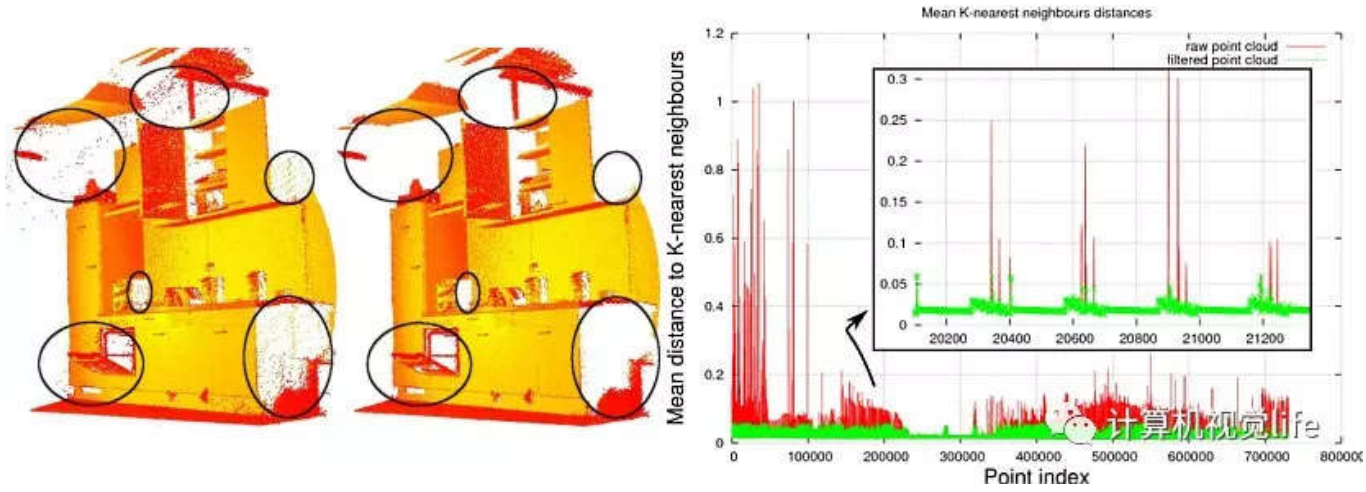
小白：听起来很高大上啊，那具体是什么统计分析技术呢？
师兄：主要是对每个点的邻域进行统计分析，剔除不符合一定标准的邻域点。具体来说，对于每个点，我们计算它到所有相邻点的平均距离。假设得到的分布是高斯分布，我们可以计算出一个均值 μ 和一个标准差 σ，那么这个邻域点集中所有点与其邻域距离大于μ + std_mul * σ 区间之外的点都可以被视为离群点，并可从点云数据中去除。std_mul 是标准差倍数的一个阈值，可以自己指定。

小白：嗯，听起来还是挺科学的，枪打出头鸟嘛！师兄，这个原理我懂啦，重点是怎么编程呢？
师兄：这里有个示例代码

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;    //创建滤波器对象
sor.setInputCloud (cloud);                            //设置待滤波的点云
sor.setMeanK (50);                                //设置在进行统计时考虑的临近点个数
sor.setStddevMulThresh (1.0);                       //设置判断是否为离群点的阀值，用来倍乘标
sor.filter (*cloud_filtered);                     //滤波结果存储到cloud_filtered
```

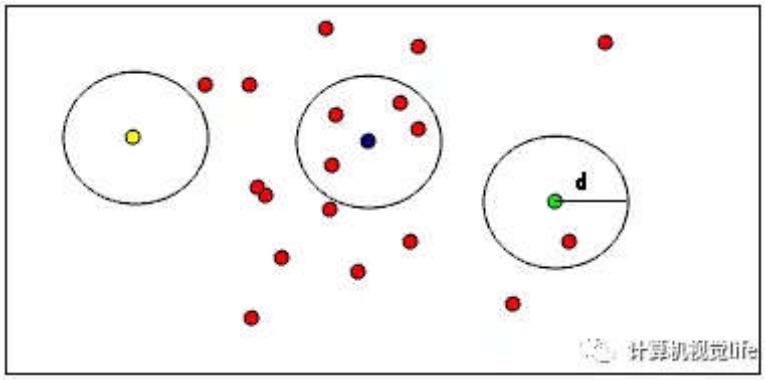
小白：和前面下采样的形式很像哎
师兄：嗯，PCL都帮你写好了，你只需要创建对象，设置参数就行啦。
小白：师兄可以稍微解释一下上面代码吗？
师兄：好，这个应该不难看懂。你看，我们先创建统计分析滤波器，然后设置滤波器输入是cloud，也就是我们待处理的点云，然后设置对每个点分析的临近点的个数设置为50 ，并将标准差的倍数设置为1， 这意味着如果一个点的距离超出了平均距离加上一个标准差以上，则该点被标记为离群点，并将它移除。最后统计分析滤波后，输出的结果就是cloud_filtered
小白：师兄这么一解释感觉容易理解多了，这个方法效果怎么样？

师兄：效果还是挺不错的，你看下图展示了稀疏离群值分析和移除的效果:原始数据集显示在左边，结果集显示在右边。图中红色表示滤波前的平均k近邻距离，绿色表示滤波后的平均k近邻距离，可以看到毛刺明显少了很多。



小白：嗯嗯，效果明显，看的见！
师兄：另外，还有一个比较简单常用的方法就是根据空间点半径范围临近点数量来滤波，对应的类名是 RadiusOutlinerRemoval，这个很容易理解，它的滤波思想非常直接，就是在点云数据中，设定每个点一定半径范围内周围至少有足够多的近邻，不满足就会被删除。
小白：这个半径范围是怎么定的?
师兄：因为空间点的具体坐标都是知道的，所以我们可以很方便的计算某个点和它周围所有点的欧氏距离，这些都是以米为单位的，可以直接指定具体的数值，对于三维建模很实用。

下图就是该类的筛选方法。比如你指定了一个半径d，然后指定该半径内至少有1个邻居，那么下图中只有黄色的点将从点云中删除。如果指定了半径内至少有2个邻居，那么黄色和绿色的点都将从点云中删除。



小白：确实很直观，编程怎么实现呢?
师兄：编程实现的简单例子在这里：

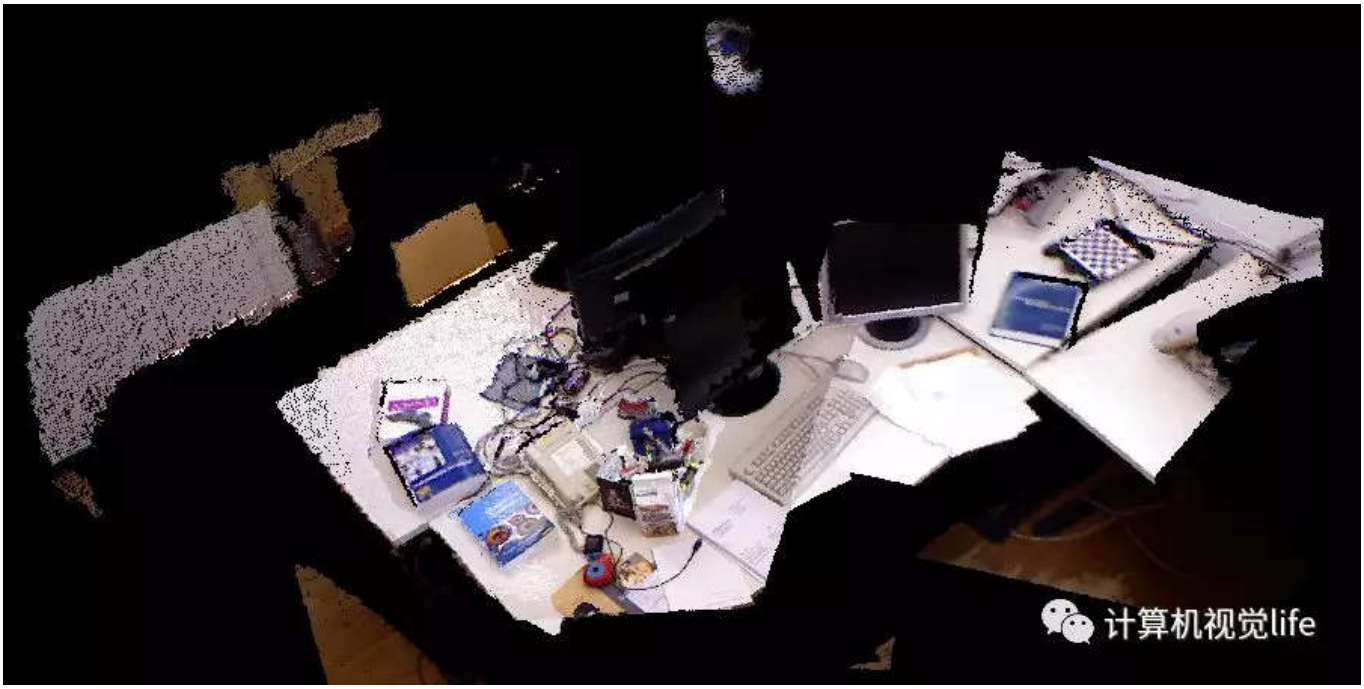
```
pcl::RadiusOutlierRemoval<pcl::PointXYZ> pcFilter;   //创建滤波器对象
pcFilter.setInputCloud(cloud);                 //设置待滤波的点云
pcFilter.setRadiusSearch(0.8);                 // 设置搜索半径
pcFilter.setMinNeighborsInRadius(2);          // 设置一个内点最少的邻居数目（见上面解释）
pcFilter.filter(*cloud_filtered);             //滤波结果存储到cloud_filtered
```

小白：确实用法都差不多，我按照前面师兄讲的方法充分了解每个类的功能，就会用啦！
师兄：嗯，今天篇幅已经比较长了，虽然感觉才讲了一点，下次我们再讲网格化吧！
小白：好，那下次再聊啦，谢谢师兄！

编程练习

给定一个融合后的点云（结果来自《从零开始一起学习SLAM | 你好，点云》），请先对其进行下采样，再进行滤波，最后输出滤波后的结果及被滤掉的离群点。

输入点云如下：



如果你进行了滤波，滤掉的噪音大概如下，你发现什么问题了吗？

本文参考：PCL官网