


- 1、进入源码目录 E:\OpenSource\OpenLidar\CloudCompare\trunk-2.5.0\qCC\plugins ，在该文件夹下有一个 “ qDummyPlugin ”的文件夹，拷贝它到当前文件夹， 并改名为 “ qMyPlugin ”。进入文件夹内部，会发现有将头文件及 CPP名字都改为 “ qMyPlugin ”，将 CMakeLists.txt 文件中多有的 “ DUMMY ”都用自己的插件名代替， 最后进入 ..\plugins\CMakeLists.txt 中，在文件的最后添加插件的链接：**add_subdirectory (qMyPlygin);**
- 2、修改头文件以及 CPP文件，实现插件功能； 在头文件中把所有 Dummy 替换成 My，并将宏定义中的 DUMMY 替换成 MY；
在 CPP文件中也与 dummy 进行相关操作；
- 3、CMake 编译，在 VC2010 中生成工程

qPCL插件运行

getActions ()加载插件的各子项 ，调用 addFilter() 函数， addFilter() 函数以 PC插件的每一个工具函数的返回值为消息；

addFilter(new LoadPCD());

```
int qPCL::addFilter(BaseFilter* filter)
{
    assert(filter);
    filter->setMainAppInterface(m_app);

    QAction* action = filter->getAction();
    if (!action)
        return 0;

    //filter already inserted?
    if (std::find(m_filters.begin(), m_filters.end(), filter) != m_filters.end())
        return 0;

    m_filters.push_back(filter);

    //connect signals
    connect(filter, SIGNAL(newEntity(ccHObject*)), -----this, ---SLOT(handleNewEntity(ccHObject*)));
    connect(filter, SIGNAL(entityHasChanged(ccHObject*)), ---this, ---SLOT(handleEntityChange(ccHObject*)));
    connect(filter, SIGNAL(newErrorMessage(QString)), -----this, ---SLOT(handleErrorMessage(QString)));

    return 1;
}
```

调用 connect ()函数，这是 Q的消息，将 filter 的信号 newEntity() 与当前对象的槽函数

handleNewEntity() 链接，触发 handleNewEntity() ，将打开 PCD文档这一按钮添加到工具条中。

loadPCD() 在被点击到时，会弹出相关对话框，调用文件读取函数，它得到的知识 PC文件的路径，并将其存储在路径数组 m_filename 中；



```
int LoadPCD::openInputDialog()
{
    QSettings settings;
    settings.beginGroup("PclUtils/LoadPCD");
    QString currentPath = settings.value("currentPath", QApplication::applicationDirPath()).toString();

    //file choosing dialog
    //We store the result directly in 'a_filenames' as it is simpler.
    //Thus we also bypass getParametersFromDialog, but we avoid keeping
    //the QFileDialog as a member...
    QStringList a_filenames = QFileDialog::getOpenFileNames(0,
        tr("Open PCD file(s)"),
        currentPath,
        "PCD file (*.pcd)");

    if (a_filenames.isEmpty())
        return 0;

    //save file loading location
    currentPath = QFileInfo(a_filenames[0]).absolutePath();
    settings.setValue("currentPath", currentPath);
    settings.endGroup();

    return 1;
}
```

在函数 compute () 中，通过 loadSensorMessage() 函数将点云加载进来，多个点云的加载时通过 for 循环将 m_filenames 中的点云加载进来的。

```
int LoadPCD::compute()
{
    //for each selected filename
    for (int k=0; k<m_filenames.size(); ++k)
    {
        QString filename=m_filenames[k];

        boost::shared_ptr<sensor_msgs::PointCloud2> cloud_ptr_in=loadSensorMessage(filename);

        if (!cloud_ptr_in) //loading failed?
            return 0;
    }
}
```

loadSensorMessage() 函数如下：

```
sensor_msgs::PointCloud2::Ptr loadSensorMessage(const QString &filename)
{
    sensor_msgs::PointCloud2::Ptr out_cloud(new sensor_msgs::PointCloud2);

    //Load the given file
    if (pcl::io::loadPCDFile(filename.toStdString(), *out_cloud) < 0)
    {
        //loading failed
        out_cloud.reset();
    }

    return out_cloud;
}
```

至此，点云被加载进内存；

addFilter(new SavePCD());

加载保存 PCD文件的相关功能；

addFilter(new NormalEstimation());

NormalEstimation(): 实现对点云的法线进行估计

获取视图点云数据的



```
void qRansacSD::onNewSelection(const cchObject::Container& selectedEntities)
{
    if (m_action)
    {
        m_action->setEnabled(selectedEntities.size()==1 && selectedEntities[0]->isA(CC_POINT_CLOUD));
    }
}
```

在qPCL插件上添加其他功能

首先在 CloudCompare源文件路径：

E:\OpenSource\OpenLidar\CloudCompare\trunk-2.5.0\qCC\plugins\qPCL\PclUtils\filters

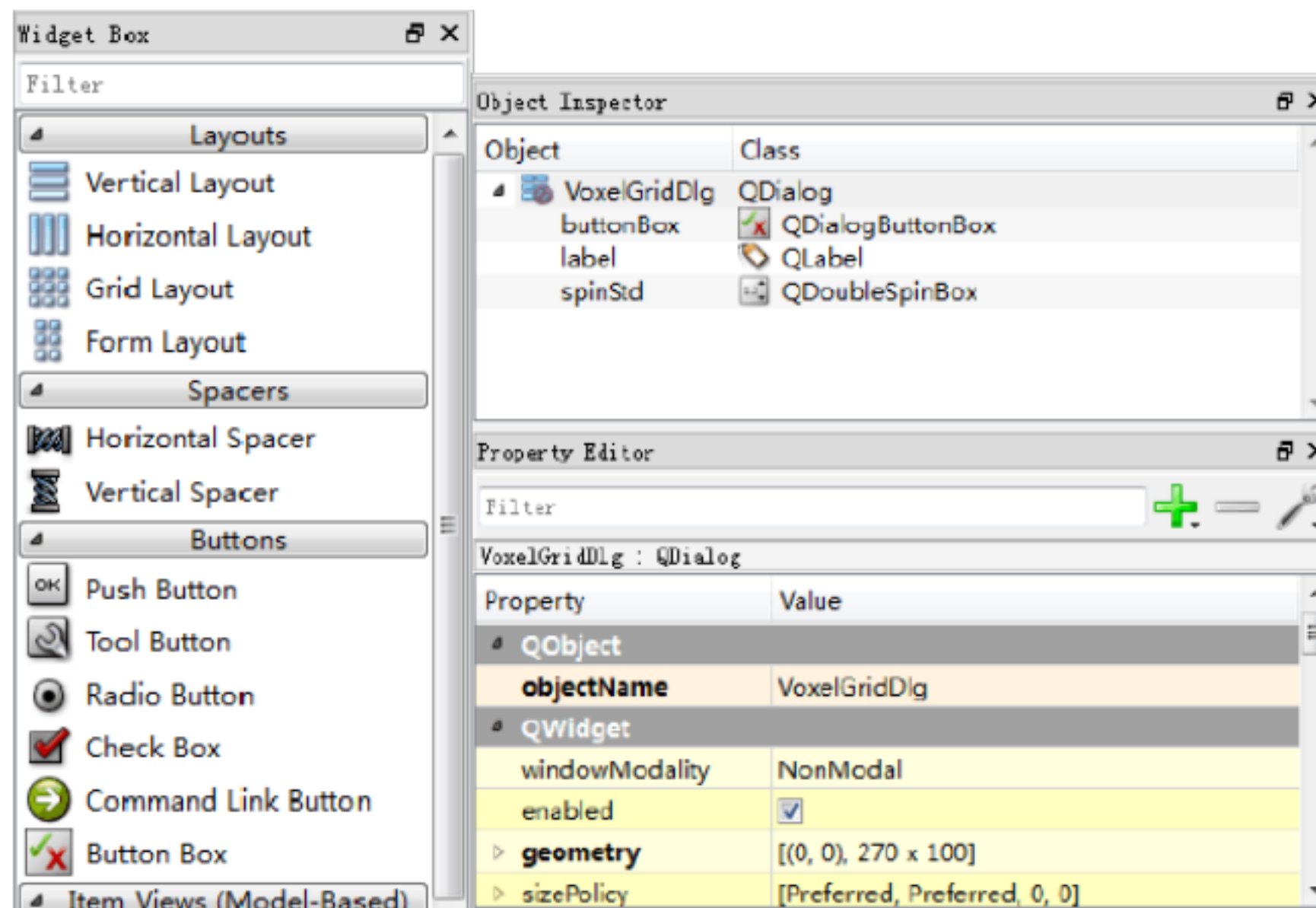
该文件下，添加自己想要的插件的头文件和源文件，可以直接创建其中某个插件的副本，然后更名为自己想要的插件，以添加 VoxelGrid 为例：

 VoxelGrid	2013/8/8 13:08	C++ Source	3 KB
 VoxelGrid	2013/8/8 10:49	C/C++ Header	2 KB

如果需要弹出对话框进行参数设置， 则需使用 Qt designer 进行对话框的设计， 保存文件生成 ui 文件



在左边的 Widget Box中可以拖动相关事件到对话框中，在右边的参数设置栏中可以修改其参数，比如类名等；



得到的 ui 文件在 VC2010中打开便可看到其对话框的标题设置、变量名及初始值。



在设置好对话框后将 ui 文件放入到如下路径中：

CloudCompare\trunk-2.5.0\qCC\plugins\qPCL\PclUtils\filters\dialogs ;

并建立源文件及头文件，也可以对已有的文件拷贝（推荐）

VoxelGridDlg	2013/6/8 11:47	C++ Source	2 KB
VoxelGridDlg	2013/6/8 11:48	C/C++ Header	2 KB
VoxelGridDlg.ui	2013/6/8 10:19	UI 文件	3 KB

资源建立好后，就开始修改 CloudCompare的代码。

在qPCL.cpp中的 getAction() 函数中添加 addFilter(newWoxelGrid()) ，并包含其头文件在 CPP文件中

#include <VoxelGrid.h>

如果是从原有文件考过来的副本，那么进入 VoxelGrid.h 中进行相关的代码修改即可。

```
#ifndef-VOXELGRID_H
#define-VOXELGRID_H

#include-<BaseFilter.h>

class-ComputeSPINImages;

class-VoxelGrid:-public-BaseFilter
{
----Q_OBJECT
public:
----VoxelGrid();

protected:
----int-compute();
----int-openInputDialog();
----void-getParametersFromDialog();
----ComputeSPINImages-*m_dialog;
//----int-m_k;
----float-m_std;
};

#endif-//-STATISTICALOUTLIERREMOVER_H
```

其中修改的部分由宏定义、类名、构造函数、以及成员变量（此处的成员变量是用来传递体素化栅格叶大小的浮点型的数据）。

然后是在源文件中进行修改：

利用查找替换功能，将原类名全部替换成 VoxelGrid ，并添加相关头文件以及对话框头文件：

```
#include-"VoxelGrid.h"
#include-"VoxelGridDlg.h"
#include-"ccPointCloud.h"
#include-"cc2sm.h"
#include-"sm2cc.h"
#include-"filtering.h"
```

在构造函数中进行插件说明修改，使得插件在鼠标移动到该处的时候其下方会出现相关的说明

```
VoxelGrid::VoxelGrid()
----: BaseFilter(FilterDescription("VoxelGrid-Remover",
-----"Remove point Using VoxelGrid",
-----"Remove point from point cloud to reduce the number of point and keep the shape of point cloud",
-----":/toolbar/PolUtils/icons/sr_outlier_removal.png")),
----m_dialog(0)
{
}
```

从对话框中获取参数的函数也需要正对具体情况进行修改。

```
void-VoxelGrid::getParametersFromDialog()
{
----//get-values-from-dialog

----m_std=m_dialog->spinStd->value();
}
```


在算法实现函数 compute() 函数中调用相关滤波算法的实现：

```
int VoxelGrid::compute()
{
    ....//get selected as pointcloud
    ....ccPointCloud* cloud = this->getSelectedEntityAsCCPointCloud();
    ....sensor_msgs::PointCloud2::Ptr tap_cloud = (new sensor_msgs::PointCloud2);

    ....//now as sensor message
    ....cc2smReader converter;
    ....converter.setInputCloud(cloud);
    ....converter.getAsSM(*tap_cloud);

    ....sensor_msgs::PointCloud2Ptr outcloud = (new sensor_msgs::PointCloud2);
    + //开始执行滤波处理
    ....voxelGrid(tap_cloud, n_std, outcloud);

    ....//get back outcloud as a ccPointCloud
    + ccPointCloud* final_cloud = sm2ccConverter(outcloud).getCCloud();
    ....if(!final_cloud)
    ....return -1;

    ....//create a suitable name for the entity
    ....final_cloud->setName(QString("%1_k%2_std%3").arg(cloud->getName()).arg(n_std));
    + final_cloud->setDisplay(cloud->getDisplay());

    ....//disable original cloud
    ....cloud->setEnabled(false);
    + if (cloud->getParent())
    +     cloud->getParent()->addChild(final_cloud);

    + emit newEntity(cloud);

    + return 1;
}
```

工具条相关图标添加（这个很重要，如果添加错误可能导致图标对应的功能紊乱）



之后就是在 filtering.cpp 中添加 voxelgrid() 函数，在此之前要在该 cpp 文件中添加相关头文件

```
#include <filtering.h>
#include <pcl/filters/statistical_outlier_removal.h>
#include <pcl/filters/voxel_grid.h>
```

```
//体素化网格方法实现下采样的函数实现
int voxelGrid(const sensor_msgs::PointCloud2ConstPtr incloud, const float inStd, sensor_msgs::PointCloud2Ptr outcloud)
{
    + pcl::VoxelGrid<sensor_msgs::PointCloud2> remove;
    + remove.setInputCloud(incloud);
    + remove.setLeafSize(nStd, nStd, nStd);
    + remove.filter(*outcloud);

    + return 1;
}
```

最重要的是要在头文件对该函数进行声明，然后在 qPCL.cpp 中声明 filtering.h （否则 voxelgrid() 函数无法识别）

最后一步，对 VoxelGridDlg.h 和 VoxelGridDlg.cpp 文件进行修改

VoxelGridDlg.h 修改如下，主要是修改宏定义、添加头文件，类名可以不做修改，但类从何处继承而来就需要改好：

```
#ifndef Q_PCL_PLUGIN_VOXELGRID_DIALOG_HEADER
#define Q_PCL_PLUGIN_VOXELGRID_DIALOG_HEADER

#include <ui_VoxelGridDlg.h>

class ComputeSPINImages1 : public QDialog, public Ui::VoxelGridDlg
{
    Q_OBJECT
public:
    ComputeSPINImages1(QWidget* parent=0);

public:
};

#endif // Q_PCL_PLUGIN_VOXELGRID_DIALOG_HEADER
```

VoxelGridDlg.cpp 修改如下，也只需要改写 ui 的继承关系即可。

```
#include "VoxelGridDlg.h"

ComputeSPINImages1::ComputeSPINImages1(QWidget* parent) : QDialog(parent), Ui::VoxelGridDlg()
{
    setupUi(this);
}
```

至此重新生成工程，在生成 INSTALL就可以把 EXE文件及相关的动态库都导入到一个文件夹中。
效果如下：

