快速3D建模原型系统

目录

[快速3D建模原型系统 1](#_Toc522026457)

[简介 2](#_Toc522026458)

[1 功能 2](#_Toc522026459)

[1.1近期 2](#_Toc522026460)

[1.1.1近期功能描述 2](#_Toc522026461)

[1.2远期 2](#_Toc522026462)

[2 开发平台及编译方法 3](#_Toc522026463)

[2.1 开发平台 3](#_Toc522026464)

[2.2 编译方法 3](#_Toc522026465)

[3 开发流程 3](#_Toc522026466)

[3.1 CC开发规则 4](#_Toc522026467)

[3.1.1 命名 4](#_Toc522026468)

[3.1.2编译 5](#_Toc522026469)

[3.1.3文件头 5](#_Toc522026470)

[3.2如何制作一个插件 6](#_Toc522026471)

[3.3 算法库CCLib和qCC的参考文档 6](#_Toc522026472)

[3.4 熟悉用户手册 7](#_Toc522026473)

[4 注意事项 8](#_Toc522026474)

[4.1 版权问题 8](#_Toc522026475)

[4.2 可持续性 8](#_Toc522026476)

[4.2.1 编写类参考文档 8](#_Toc522026477)

[4.2.2 编写插件参考文档 10](#_Toc522026478)

[5 其他 10](#_Toc522026479)

[5.1 仓库建设 10](#_Toc522026480)

# 简介

建设一个方便我们研究和使用的快速3D模型系统。每一届的研究生或者本科生（毕设或者srp）都可以使用这个系统来做研究或者为这个系统做贡献。

# 1 功能

## 1.1近期

* 基本的点云交互功能
* 自动识别点云对应的3D模型（面表示）
* 渐进式合并点云。
* Ransac算法的改进
* 梁柱提取算法
* 开口门窗定位算法
* 类纱窗的提取算法
* 模型求交算法
* 多站点模型拼接算法？

### 1.1.1近期功能描述

1. 基本的点云交互功能

能够显示和旋转点云；

能够用某种工具部分选取点云，比如用一个正方体盒子把兔子点云的头部选取出来。

能够对选择的点云进行修改（是改变点的坐标，还是修改选到的点的范围？）

（2）自动识别点云对应的3D模型（面表示）

用深度学习算法（或其他算法）查找最相似的点云模型来自动识别点云对应的3D模型（面表示）。

（3）渐进式合并点云

依次循环，获取第二个3D模型之后，能自动合并3D模型，这样就可以进行渐进式的建模。

## 1.2远期

* 纹理库的准备
* 纹理贴图算法
* 常用建筑物基本单元点云库的建立
* PointNet算法的集成
* Bim模型的生成，与常见建模工具之间进行数据交换。

# 2 开发平台及编译方法

## 2.1 开发平台

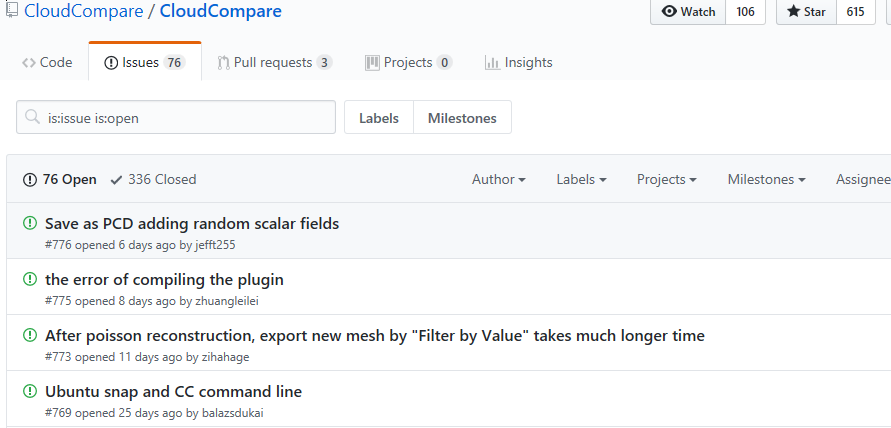
使用Windows版本的开源点云交互软件CloudCompare进行二次开发。用到的其他相关软件待补充。

## 2.2 编译方法

待补充。

# 3 开发流程

以插件的形式，在CloudCompare中加入相关的功能，还能拥有自己的界面。平时可以多阅读CloudCompare的github文档，https://github.com/CloudCompare/CloudCompare。其中的CONTRIBUTING.md介绍了CC的一些开发规则和注意事项。Github项目上的Issues分组里，是各位建设者对一些bug的讨论，在开发过程中遇到问题可以在这个板块参与讨论，以便寻求帮助。



以下是对CONTRIBUTING.md的一个简单翻译。

## 3.1 CC开发规则

### 3.1.1 命名

CloudCompare中的名称应该尽可能具有描述性，没有缩写，除非是非常明确或常见的（例如fw而不是forward等）。 大多数变量名称应以小写字母开头。 如果名称由多个单词组成，则每个单词的第一个字母应为大写（当然，第一个单词除外）。  
示例：numberOfPoints，ptsCount

**（1）特殊情况下：**

* 静态变量：应始终以前缀s\_开头（小写 - 如s\_defaultFilename
* 静态方法：应始终以大写字母开头（如InitGLEW）
* 类：应始终以前缀cc开头（小写 - 如ccConsole）
* enumerators:：

所有字母应大写  
应始终以前缀CC\_开头  
单词由下划线分隔（如CC\_OBJECT\_FLAG）

* 宏：以前缀MACRO\_开头，后跟标准方法名称（如MACRO\_SkipUnselected）
* const变量：  
  所有字母应大写  
  单词由下划线分隔（如NORMALS\_QUANTIZE\_LEVEL）
* macro const（#define）：应该避免; 与const的语法相同

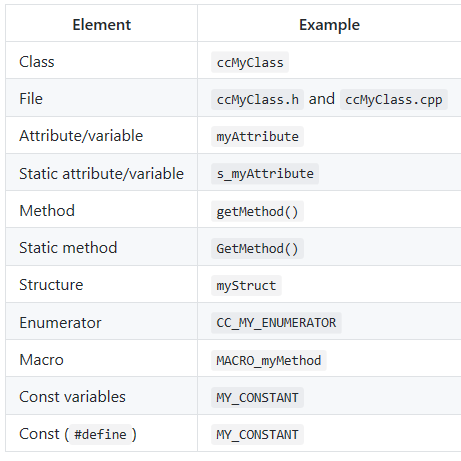
**（2）文件**

* 文件命名遵循与大多数CloudCompare元素相同的规则（小写的第一个字母等）
* 每个类应该单独保存在标头+源文件中。在特殊情况下，单个类使用的非常小的类可以与此类一起保存。标头+源文件名应与主类相同。

示例：ccConsole保存在ccConsole.h和ccConsole.cpp中

* 文件名不应包含任何空格字符，可以用下划线代替。
* 所有与数据相关的类（数据模型，数据库等）都应保存在db目录中。
* 图像（图标）应全部保存在images目录（或其子目录之一）中。
* GUI模板（主要是.ui Qt文件）应保存在ui\_templates目录中

示例小结：

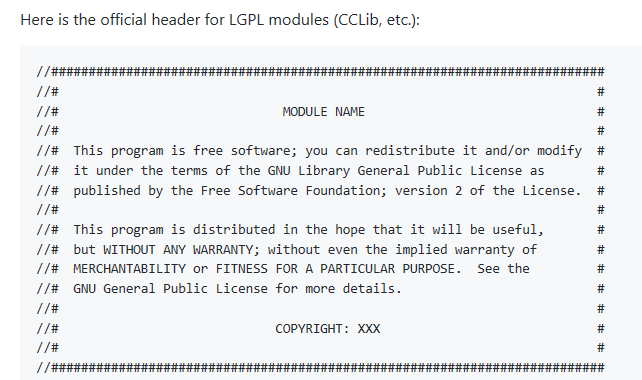


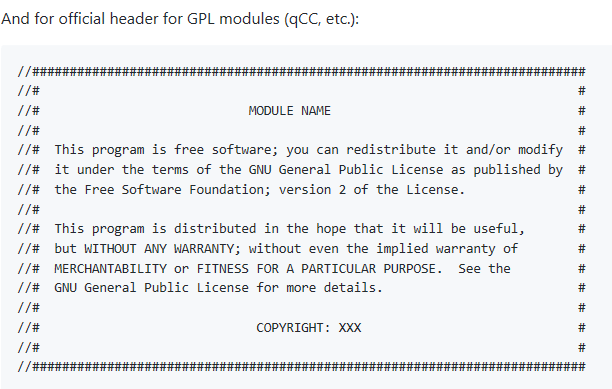
### 3.1.2编译

为了避免与Unix环境不兼容的语法，必须遵守以下规则：¬仅对包含路径使用“/”。  
示例：include“../db/ccPointCloud.h”

### 3.1.3文件头

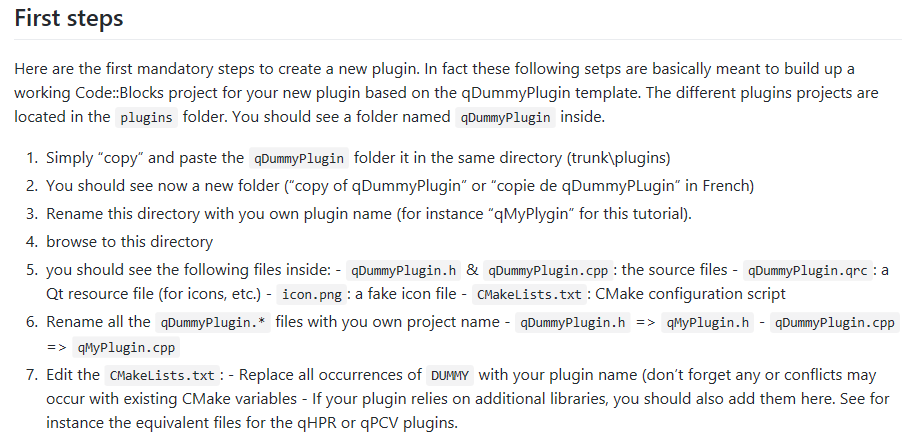
集成到任何CloudCompare模块（CCLib，qCC等）的任何新源文件（.h，.cpp等）都必须显示正式标头。使用时请到CONTRIBUTING.md文件下copy.





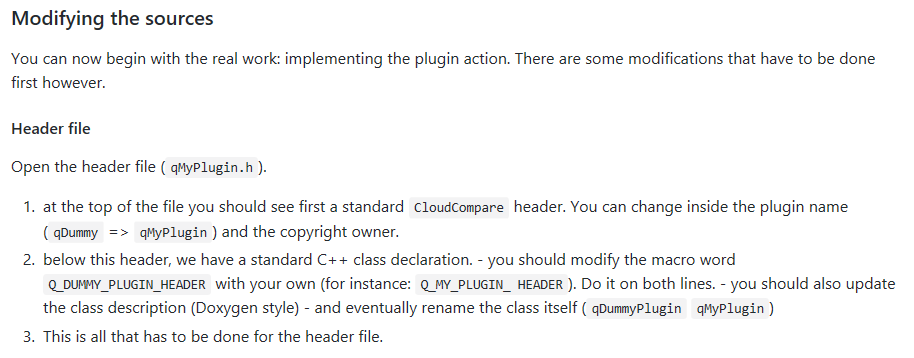
## 3.2如何制作一个插件

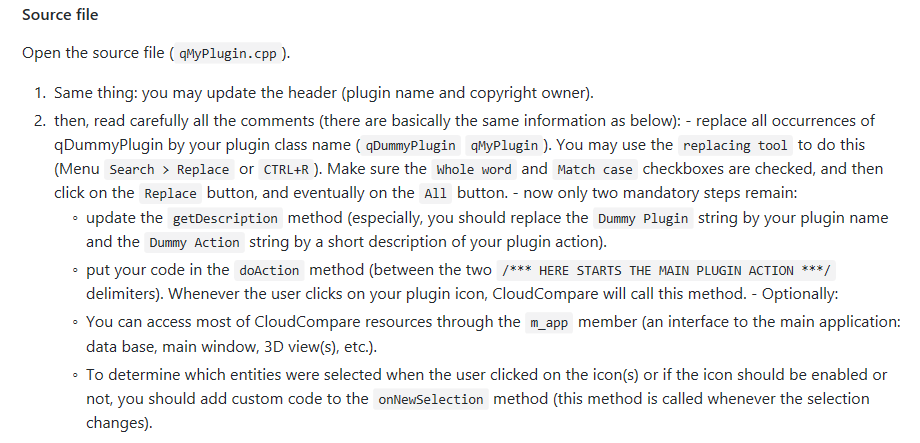
设计新插件是扩展qCC（CloudCompare）功能的简单方法，无需修改其核心并完成所有连接，就可以轻松设计一个新功能，该功能可以应用于当前在CloudCompare中加载的一个或多个实体。 而且，插件可以显示自己的对话框。 提供了一个虚拟插件结构（源代码和相应的Code :: Blocks项目）作为模板。



## 3.3 算法库CCLib和qCC的参考文档

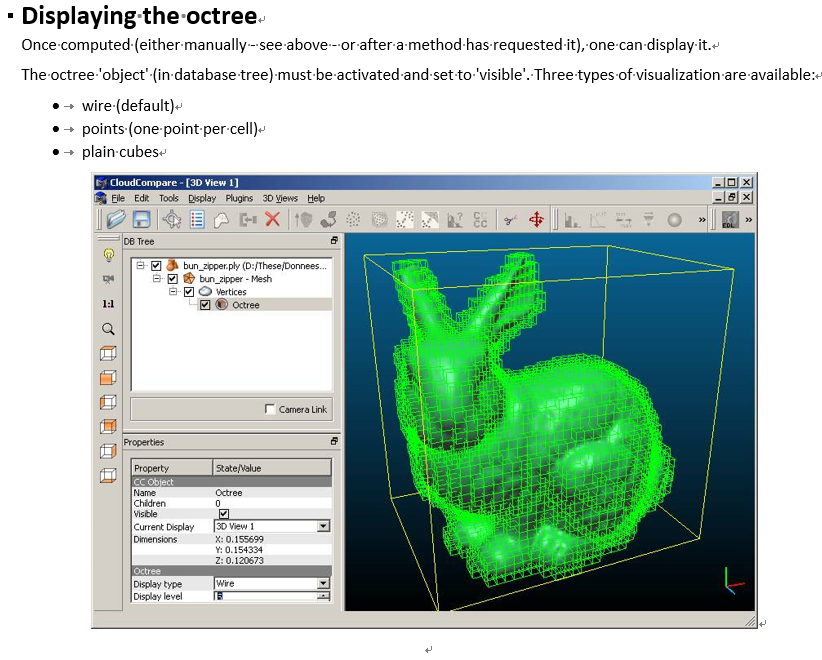
CCLib doxygen文档https://www.cloudcompare.org/doc/CCLib/html/index.html  
qCC doxygen文档 https://www.cloudcompare.org/doc/qCC/html/index.html





## 3.4 熟悉用户手册

CC在文件夹doc/en\_2.6.1 目录下有一个非常详尽的用户手册 CloudCompare User manual 2.6.1.docx，没事多翻翻，熟悉功能。



# 4 注意事项

## 4.1 版权问题

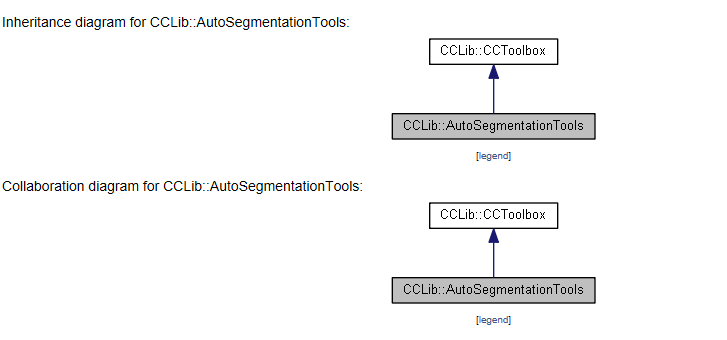
注意使用到的外部算法库是否能够任意使用，需要版权的就不太适合商用。

## 4.2 可持续性

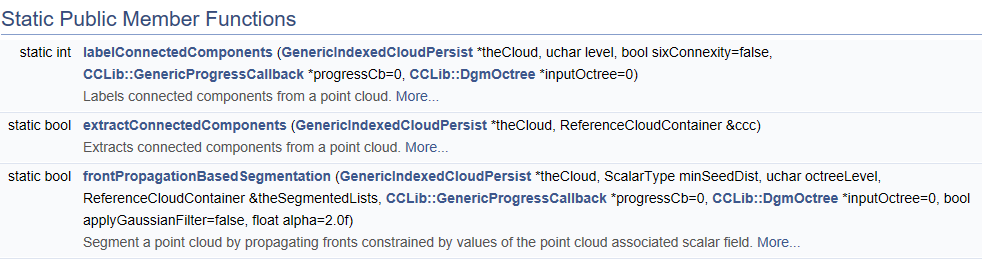
为了方便阅读和使用二次开发的代码，需要写参考文档。

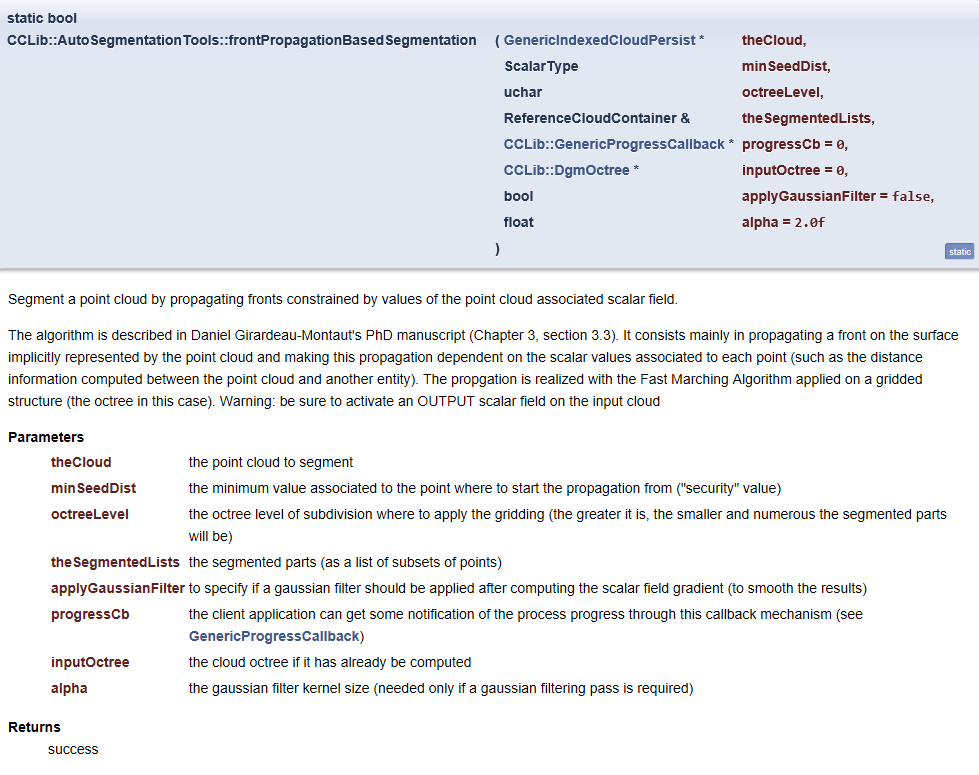
### 4.2.1 编写类参考文档

如果是增加了一个子类，格式参考CCLib的某一个类的reference，其中包括（1）继承示意图：



（2）对类成员的描述：





https://www.cloudcompare.org/doc/CCLib/html/class\_c\_c\_lib\_1\_1\_auto\_segmentation\_tools.html

### 4.2.2 编写插件参考文档

如果是增加了一个插件，要详细写出：

1. 这个插件使用了什么算法，简单介绍这个算法的思想，每一次更新都有什么改进。
2. 如何使用这个插件，注意多一点截图；

# 5 其他

## 5.1 仓库建设

建立一个我们的github仓库（如果不能关闭分享功能，那么用什么其他仓库比较好？），并把使用方法和update的规则统一一下。github仓库主要用来管理项目和保存讨论记录，且方便更新和下载。（晓凤来管理？）