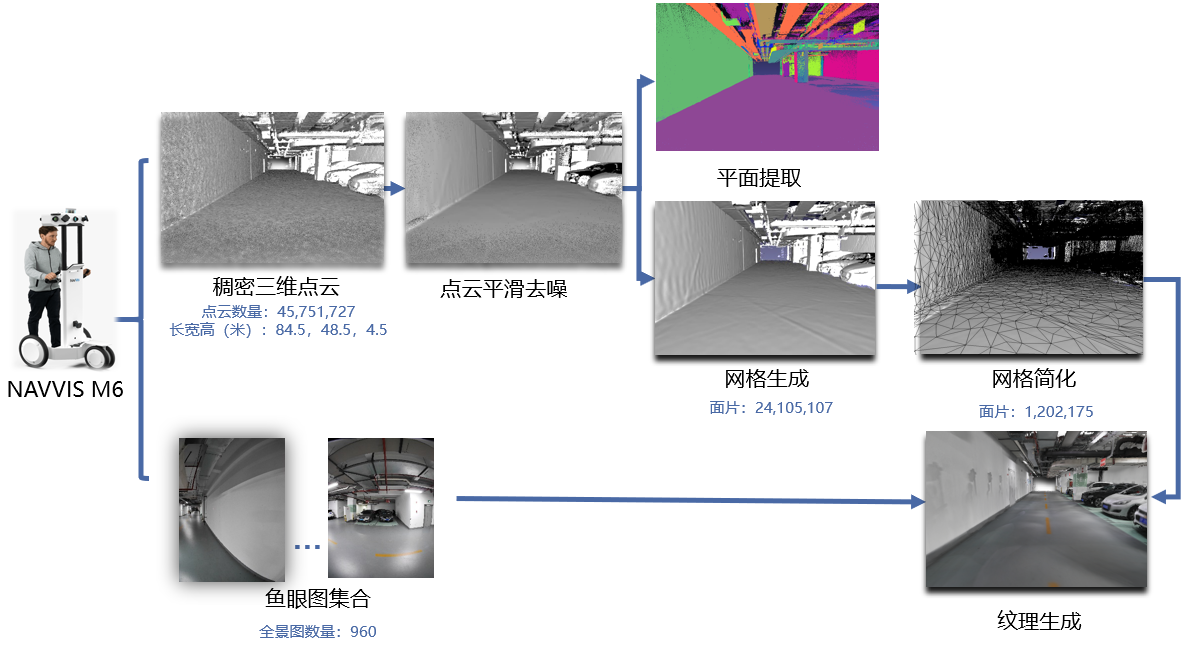
## 高精度室内场景建模和小物体移除算法说明文档

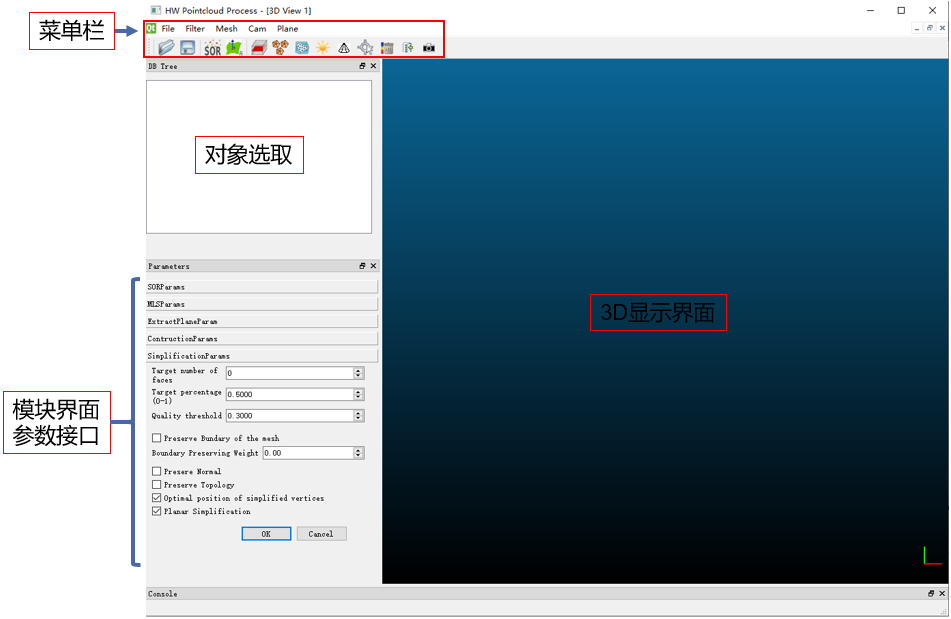
1. **需求规格说明**

服务于三维世界渲染项目，同时满足虚拟现实和对虚拟世界进行编辑的目的，需要建立一个关于真实世界的三维场景网格，排除三维场景中其它物体的影响，渲染主体场景。所以需要一个系统能够对场景进行网格重建，对场景网格进行纹理映射，同时具有去除场景中的可移动物体，对主体场景进行纹理映射。

1. **系统设计**
2. **系统流程：**



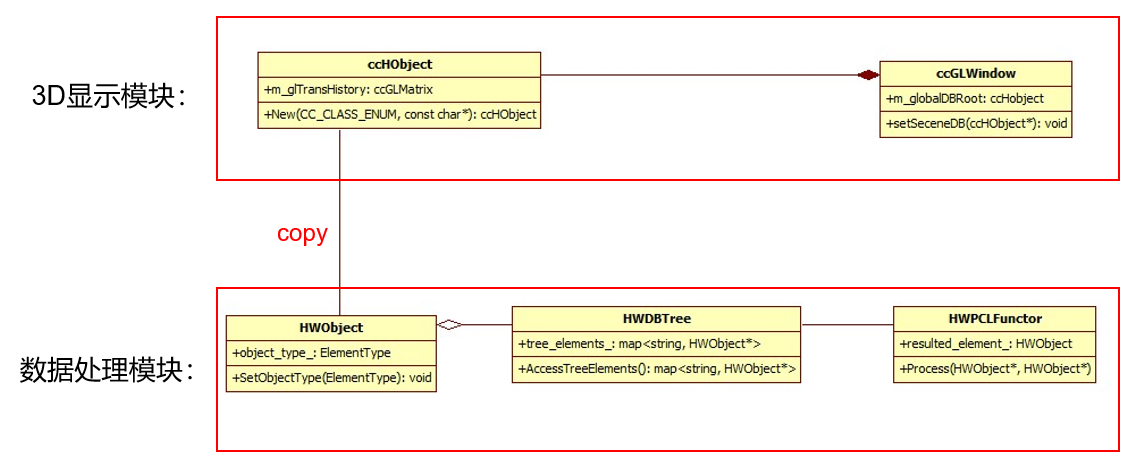
1. **界面设计**



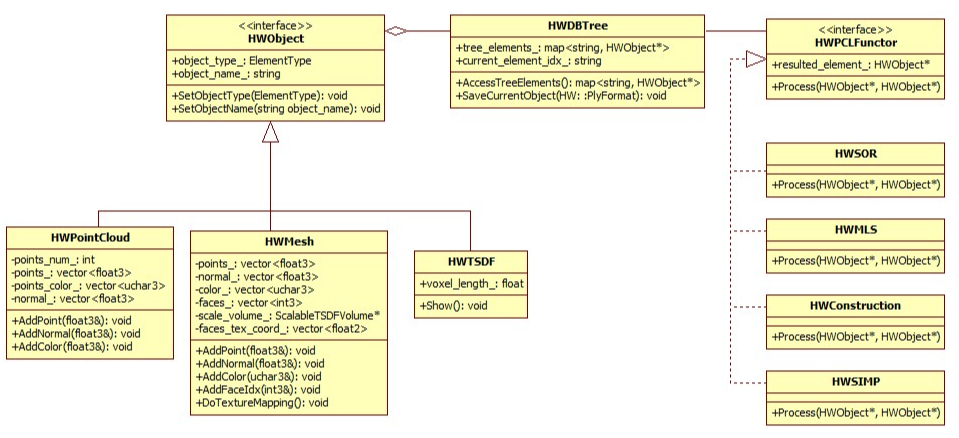
主要包括四个部分，它们提供的图形接口分别如下：

1. 菜单栏：对象的加载，对象保存，SOR，MLS，点云网格化，纹理生成，对象选取等等。
2. 对象选取：对加载的多个对象提供图形句柄，供用户操作等等。
3. 模块界面参数接口：提供各个模块的参数设置接口，用于对这些模块的控制。
4. 3D显示界面：提供对象在空间三维的显示，并且对这些对象在三维空间进行操作，包括旋转，平移等等。
5. **各模块设计**

系统模块主要包括两大块，分别是：三维显示模块，除了显示模块外的数据处理模块。



1. **数据处理模块：**

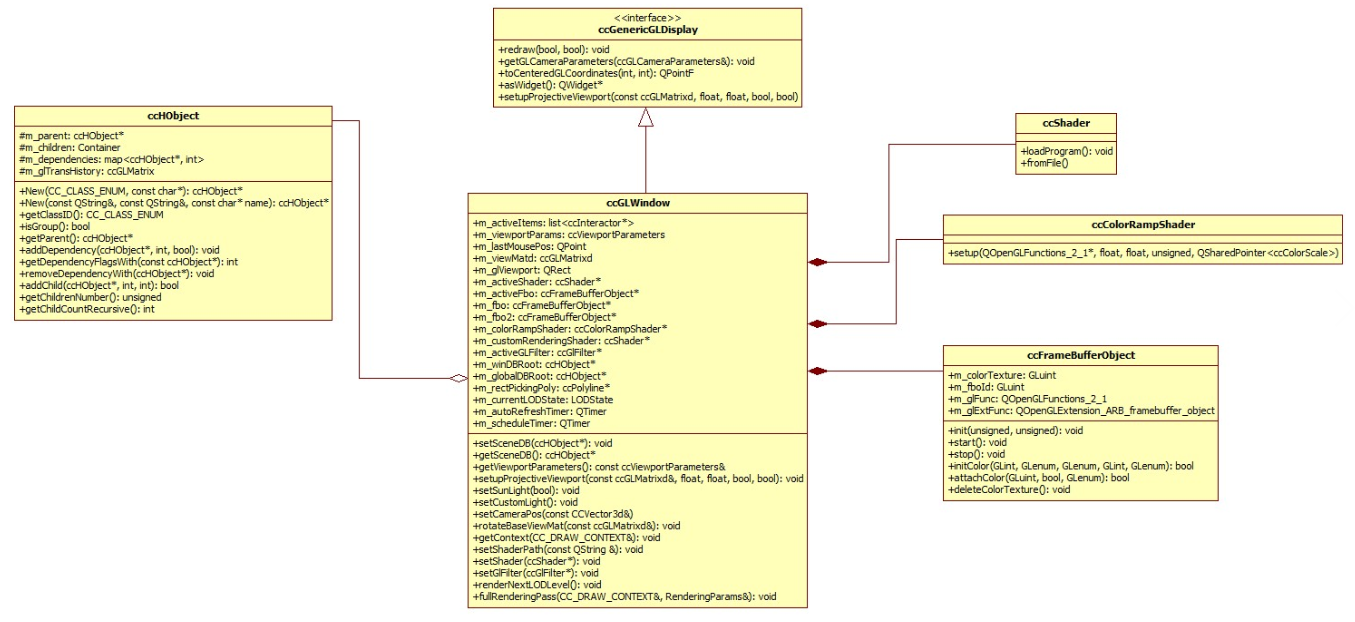


数据模块主要包括三个实体类HWPointCloud，HWMesh，HWTSDF，以及它们的功能类HWSOR，HWMLS，HWConstruction，HWSIMP。

1. HWPointCloud类，主要是用于存储点云数据。并提供一些基本的对于点云的操作。
2. HWMesh类，主要用于存储网格数据，并且提供一些简单对网格的操作。
3. HWTDF类，这个类主要存储体术表达的数据，暂且还没有用到。
4. HWSOR类，这个类是提供SOR 去噪功能，对点云进行操作，它可以通过功能类接口HWPLFunctor类调用其功能。
5. HWMLS类，这个类提供平滑点云功能，它主要是对点云进行操作，它可以通过功能类接口HWPLFunctor类调用其功能。
6. HWConstruction类，这个类提供生成网格功能，它主要是对点云进行操作，它可以通过功能类接口HWPLFunctor类调用其功能。
7. HWSIMP类，这个类提供对网格的简化功能，它主要是对点网格进行操作，它可以通过功能类接口HWPLFunctor类调用其功能。

实体类的指针都存在HWDBTree类中，HWDBTree类可以通过HWPCLFunctor类去完成各个功能。

1. **三维显示模块：**



3D显示模块，主要是用CloudCompare的显示模块，用它提供的一系列在3D空间的操作。

1. ccHObject主要存储需要显示的数据，如点云和网格等等。
2. ccGLWindow这个类主要是用于显示三维数据，所以的关于显示都在这个类中。可以通过它控制三维显示的上下文。
3. **各模块介绍**
4. **离群点去噪(SOR)**

**3.1.1：功能需求**

随着扫描设备种类越来越多，促进了商业的应用。由于扫描场景的复杂，设备扫描出来点云总是一些噪声，对于点云的后续处理影响较大，所以需要对点云进行去噪。

**3.1.2：算法原理**

离群点去噪称Statistical outlier removal filter，用于剔除异常值。它的思想为：对每一个点邻域进行统计分析，计算它到所有临近点的平均距离。假设得到的结果是一个高斯分布（由均值和标准差决定），那么平均距离在标准范围内（由全局距离平均值和方差得到）之外的点，被定义为离群点，并从数据集中删除。

具体的算法流程算法，它有两个参数，分别是，（表示顶点数量，表示乘子）：

1. 计算点云集合中每个点到领域的平均距离，保存到中。
2. 计算点云集合的平均距离集合平均值和样本标准差，即的平均值和标准差，表示为和，再计算距离阈值。
3. 对比点云集合的和，将小于距离阈值的顶点全部设置为删除。

**3.1.3:结果输入输出**



图1 左图：原始点云 右图：SOR去噪的点云

本次实验，用的是场景中的部分点云，SOR参数设置，number of points设置为8，Standard deviation multiplier设置为3.00。得到的结果为右图。

1. **MLS**

**3.2.1：功能需求**

扫描设备扫描场景出来的点云一般在平面处会有凹凸出现，它主要包含下面两个方面因素：1）扫描设备的精度；2）场景融合一般为多个视角得到的，因此会有很多视角的点云之间相互注册加融合，注册的误差可能导致场景中平面的点云凹凸不平。点云凹凸不平会影响后续的点云处理。因此需要一个功能，能够平滑这些点云。

**3.2.2：算法原理**

MLS类名“移动最小二乘”，它的思想是点云局部化，对每一个局部区域先计算支撑平面，通过支撑平面和局部区域的点云，拟合多项式曲面。

其中。拟合出多项式曲面后，将点云投影到多项式曲面上，即将点移动到多项式曲面上。

具体算法，算法主要输入的参数有搜索半径，流程如下：

1. 对点云中每一个顶点搜索它的空间半径为的点，放入邻近点云中。(mls.hpp中searchForNeighor)
2. 用PCA对邻近点拟合一个平面。设置平面为轴，它的法向量为轴的坐标系，将邻近点转到新的坐标系上，在新的坐标系上的点云，拟合出曲面，并且让顶点投影到曲面上并保存. （mls.hpp文件，函数computeMLSPointNormal）。
3. 重复1）2)过程直到搜索所有的顶点（mls.hpp文件，函数computeMLSPointNormal)。

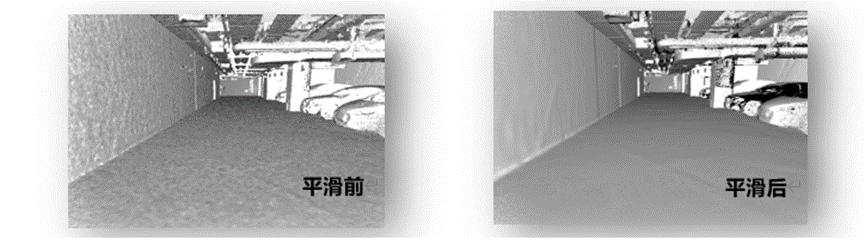
**3.2.3：测试结果对比**

图 2 MLS测试，左图平滑前，右图平滑后

通过结果实验对比，输入没有经过平滑得点云有一些凹凸不平，经过了MLS平滑后，点云变得平滑。

1. **网格生成**

**3.3.1：功能需求**

在应用场景中点云表达缺少了点云之间的关系，它们之间的关系只有空间邻近关系，没有网格表达空间场景信息丰富。需要将场景中的点云重建为网格场景。

**3.3.2：算法原理**

* 1. **TSDF表达**

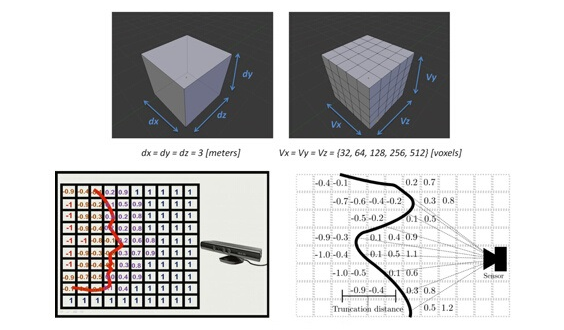
体表达的概念首次提出是在1996年，Curless等人在KinectFusion中提出了用符号距离函数（SDF）来表达模型的方法。他们将物体表面定义为零，自由空间（即物体外部）为一个正值，值的大小随着其到最近表面的距离的增大而增大，而占有空间（即物体内部）则是一个相似的负值。这些函数值被存储在一个栅格立方体中。由于只有靠近实际表面的体素是重要的，研究者们通常采用截断符号距离函数（TSDF）来表达模型。在设计算法时，必须定义立方体的体素大小以及空间范围。以图1上为例所示，图中定义了一个3x3x3米的立方体，并把立方体分为不同分辨率的小立方体网格（体素）。

图3 KinectFusion的立方体网格形式以及TSDF

常规的体素网格在内存消耗方面非常低效，并且受限于预定义的立方体和分辨率。因此，我们采用了体素哈希结构。在这种方法中，体素的空间位置所对应的索引存储在线性化的空间哈希表中，并通过空间哈希函数进行寻址。由于内存中只保存那些包含空间信息的体素，这种策略大大减少了内存的消耗，在理论上允许设定一定大小和分辨率的体素块来表示无限大的空间。与体素层次结构相比，哈希结构在数据插入和访问上具有巨大优势，其时间复杂度均为O(1)。

* 1. **MarchingCube生成网格**

每个立方体栅格的每个顶点都有对应的TSDF值，根据集合关系可以很方便的推断出：

1. 如果一个立方体栅格的每个顶点的TSDF值都<0，那么可以认为这个立方体完全位于曲面的一侧。
2. 同样的，如果每个顶点的TSDF值都>0，那么可以认为这个立方体完全位于曲面的另一侧。

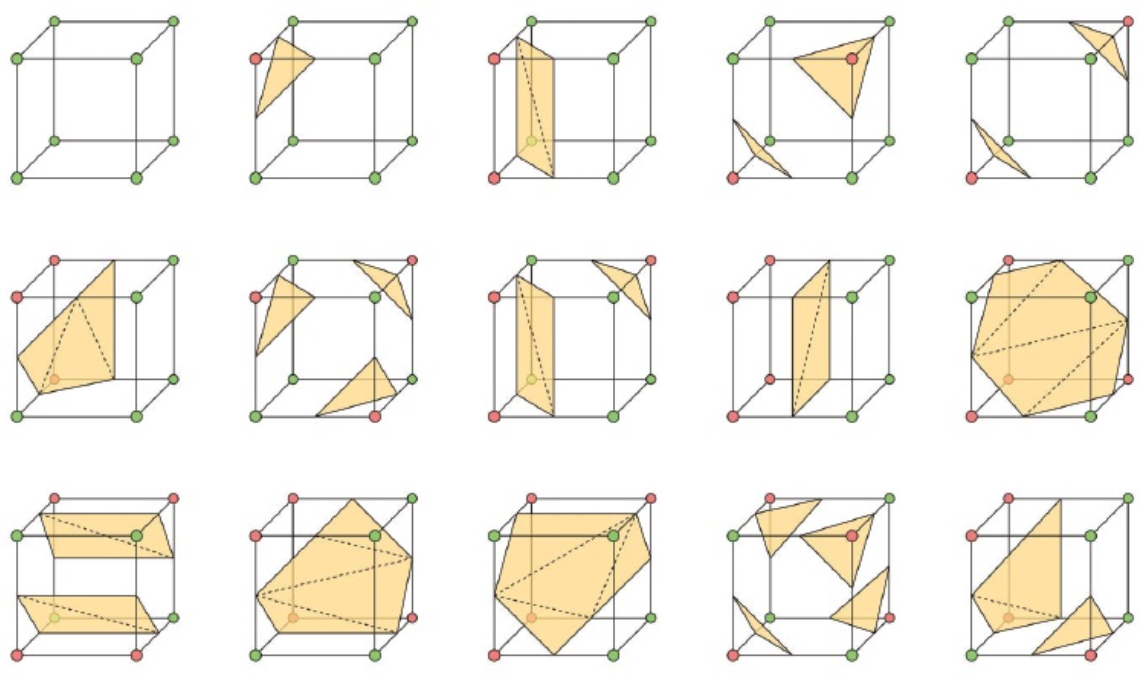
如果有部分顶点值<0，同时有部分顶点值>0，那么整个曲面与这个立方体相交。此时，通过分析各个顶点的TSDF值，在对应的边上进行线性插值找到对应的顶点位置，从而构建出这个立方体对应的三角形集

图4 Marching Cubes算法的15种基本情形

**3.3.3：测试结果**

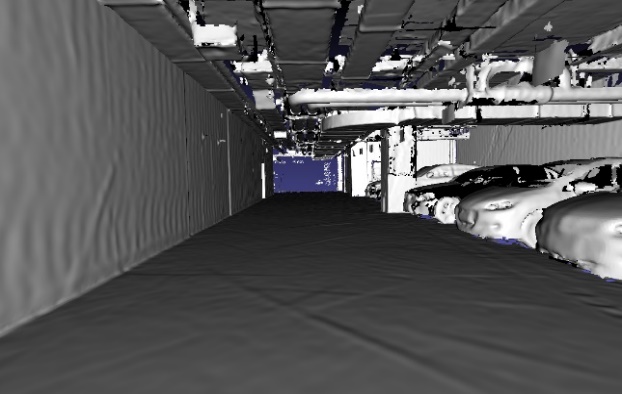


图5 左图：原始场景点云 右图：Marching Cube生成的网格

本次测试应对于停车厂场景进行实验，它有45,751,727点云数量，长宽高（米）：84.5，48.5，4.5

对于场景重建的参数Voxel length 为0.02米（表示精度），场景中最大Volume Resolution为10000（它表示Marching Cube 里面空间每个维度最大可以有10000个voxel）。Volume trunc length为0.04表示在建立Volume时候离场景表面的距离大于0.04的voxel设置为 0。

时间花费为：78.246s

1. **平面提取**

**3.4.1：功能需求**

扫描设备得到的三维场景非常复杂，需要场景区域进行分割，以便后续对不同的区域不同得处理。现在分割算法有很多，主要分成传统的方法和神经网络的方法。

传统的方法有：基于边缘的分割方法，基于区域分割方法，基于属性分割方法，基于模型分割方法等等。

1. 基于边缘的分割方法，是通过检测点云的一些区域边界来进行分割，这种分割对于噪声和非常敏感，不适合对大型场景分割。
2. 基于区域分割方法，它是利用邻近区域的信息对具有相似属性的点归类，从而达到分割的目的。对于相似属性的定义以及不同区域的边界归属问题等等，对于复杂场景容易产生过度分割。
3. 基于属性的方法，它首先要计算属性，然后根据点的属性进行聚类，它的分割比较依赖于属性选着和计算。另外如果输入点有多维属性的时间效率不高。不能用于大型场景中。
4. 基于模型的方法，这类分割是一些比较简单的几何形状（球形，圆锥，平面等等）来对点云进行分组，从而达到分割目的。它有数学原理，分割速度快。

神经网络算法：有PointNet++等等，很多处于研究阶段，它们对于大型场景受限于它们的存储，对于百万级别以上的点云分割的神经网络分割算法，暂且还未找到。

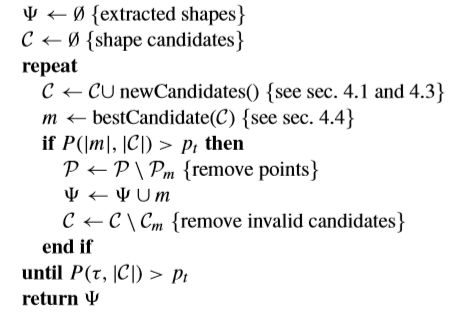
考虑到场景复杂性和时间问题。所以对于大型场景先进行简单的分割，使用基于模型的分割方法，最简单的分割就是将场景中分割成平面和非平面。然后通过对平面区域的筛选处理，提取场景中轮廓。从而达到粗略的分割目的。

**3.4.2：算法原理**

选取的平面提取算法，它的主要思想是：预先选取平面基元，在点云中采样局部区域，然后用ransac进行平面拟合，得到基元平面，在通过最大化平面点云原则，将基元平面周围的点云投影到基元平面上，选取最大的满足条件的平面点云。将得到平面点云从点云中删除，重复提取剩余点云的平面点云。为了理解平面提取算法，需要说明一下几个概念。

1. 平面基元，根据空间几何定理，空间平面可以通过三个不在同一条直线上的顶点唯一确定，所以预先定义平面的，最少的点集为3个，它们的表示方式可以通过平面上的一个顶点和它的法向量确定。
2. Ransac平面拟合，它是在采样的局部区域，对这个空间局部区域拟合一个平面，得到拟合局部区域的平面。
3. 选取满足条件的平面点云，它需要满足以下条件：在平面附近点云（离平面），点云的法向量和平面的法向量不能过大()，最大的点云数量。表示为

算法流程：



其中NewCandiates()为新建立的平面基元，根据上面标准3)，选择最好平面点云，并将平面点云删除掉。重复直到没有局部满足最小平面的需求。(mainWindows.cpp, DoExtractPlaneAction())

以上是对点云的提取平面点云。提取平面后，需要对这些平面进行分析，判断哪些平面点云是属于场景的轮廓，哪些平面属于其它种类。对于场景中，一般的较大的平面都是属于轮廓，特别是建筑型场景。通过对平面区域的面积，同时对于因为法向量约束或者离平面距离等的原因，对于场景中的边缘区域可能会被剔除在平面之外。会有很多细长的平面点云出现。所以在两个平面边缘的地方会有很多细长的平面点云，如下图红色区域：



通过对平面区域的长宽比，选择平面区域的细长点云，同时这个细长的平面点云具有一定的长度这些都是提供出接口供人设置，如果不需要这类平面点云可以将它设值足够大即可，赋值它属于场景轮廓，这样有效的减少了轮廓之前的平面之间的缝隙。

**3.4.3：测试结果对比**

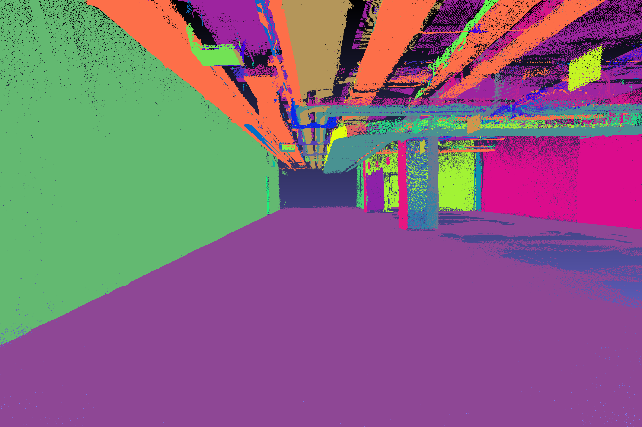
 

图6 左图：输入的场景点云 右图：提取平面后的场景点云

通过实验可以得到将场景中的平面区域提取出来，再通过筛选非平面区域，从而能把平面区域合在一起。

1. **网格简化**

**3.5.1：功能需求**

为了控制程序处理时间，我们需要对过密的网格进行网格简化。我们采用了论文Surface Simplification Using Quadric Error Metrics提出的方法。（quadric\_simp.cpp,28行）

**3.5.2：算法原理**

该算法利用顶点对的迭代收缩来简化模型，并使用二次矩阵来保持曲面误差近似。通过收缩任意顶点对(不仅仅是边)，该算法能够连接模型的非连通区域（如图7）。无论是在视觉上还是在几何误差方面，这都可以促进更好的近似。

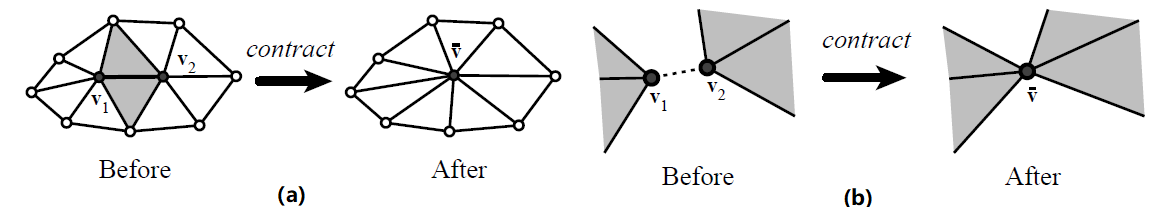
为了在每次迭代中选择要执行的收缩，需要定义顶点对收缩的代价。为了定义这个代价，该文尝试描述每个顶点的误差。为此，作者将每个顶点关联一个4\*4

图 7（a）边的收缩（b）非边的收缩

的矩阵,将顶点的误差定义二次型。对于给定的收缩，必须推导出一个新矩阵，用于逼近处的误差。作者选择使用简单加法规则。的位置通过最小化的值得到。这等价于解决：

，

如果这个矩阵是不可逆的，尝试沿着段寻找最优顶点。如果这也失败了，就从端点和中点中选择。

接下来讨论如何计算每个顶点的初始矩阵。，在原始模型中，每个顶点都是一组平面的交点的解，即在该顶点上相交的三角形的平面。作者把一组平面和每个顶点联系起来，把顶点相对于这个集合的误差定义为到它的平面距离的平方和：

，

代表由等式定义的平面（）。

中给出的误差度量可以写成二次形式:

，

，

因此，为了计算点对收缩算法所需的初始矩阵，每个顶点必须叠加在该顶点相交的三角形平面。对于每个顶点，这组平面定义了几个基本误差的平方。这个顶点的误差二次曲面是基本二次曲面的和。注意，每个顶点的初始误差估计值为0，因为每个顶点都位于其所有关联三角形的平面上。

算法总结如下：

1. 计算所有顶点的初始矩阵。
2. 选择所有有效的对（边或者点对距离小于一定阈值）。
3. 计算每个有效点对的最优收缩目标。这个目标顶点的误差变成了对其进行收缩的代价。
4. 把所有的点对放在堆中，将代价最低的点对在顶部。

迭代地删除堆中代价最小的点对，收缩这一对，并更新涉及的所有有效对的代价。

**3.5.3：测试结果**

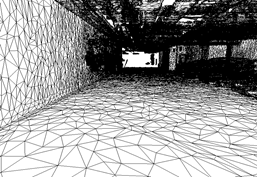
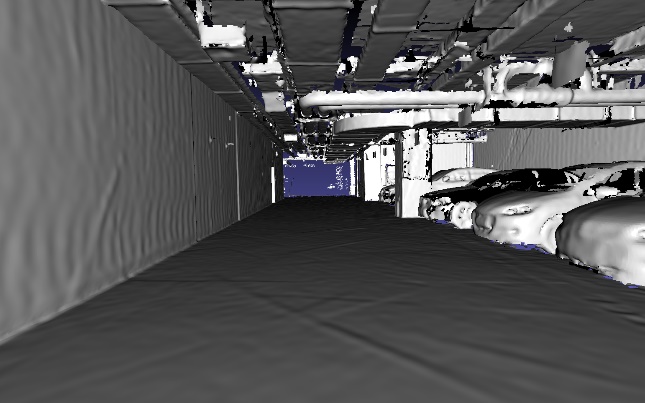


图8 左图：March Cube生成的网格 右图：简化后的网格

本次测试使用的是系统生成的网格，然后进行简化后得到的网格。

1. **纹理生成**

**3.6.1：功能需求**

应用场景中仅有网格，不能很好表示场景中的信息，需要添加纹理信息使得场景表达更为丰富，所以需要对网格进行纹理贴图。

**3.6.2：算法原理**

1. **鱼眼相机模型**

NavVis采用是鱼眼镜头，而纹理生成需要使用正常图片。因此我们需要根据鱼眼相机模型，将NavVis输出的鱼眼图片转化为正常图片(hw\_mesh.cpp,1401行)。

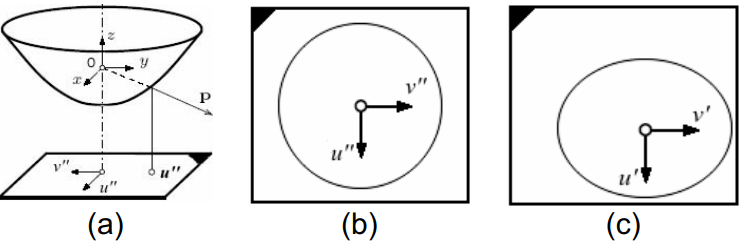
鱼眼镜头是一种超广角镜头，视角一般可达到220°或230°，这为近距离拍摄大范围景物创造了条件。NavVis采用的是Scaramuzza提出的全向相机模型。

图9 （a）反折射情况下的坐标系。（b）相机平面。（c）图像平面。(b)和(c)由仿射变换相联系。

为场景中一点，设为投影到相机平面上的点，为图像平面上的点。其中

，，。接着，引入图像投影函数，它代表相机平面上的点与从视点O发散到场景点的向量（见图9a）之间的关系。通过这样做，一个全向相机的完整模型是

函数可以是与镜面或透镜结构有关的各种形式。Scaramuzza将定义为

其中系数, =0,1,2...N为模型参数;是到相机中心的距离。因此，(1)可以改写为

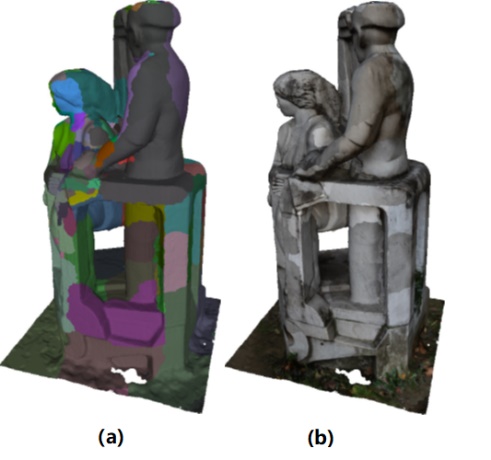
1. **纹理生成（hw\_mesh.cpp,1601行）**

图 10 （a）网格的标签，每种颜色表示一个视角（b）颜色调整前的纹理

网格的纹理生成主要分为两步：（1）为网格的每个面片选择一个视角生成初步的纹理；（2）由于不同的视角可能存在强烈的光照、曝光和尺度差异（见图3b），所以需要对纹理进行一致性优化，来避免相邻纹理块之间的缝隙。

1. 1.2.1视角选择（view\_selection.cpp,108行）

视角选择首先需要确定每个视角下每个面片的可见性。接下来，使用成对马尔科夫随机场能量公式来为每个面片分配一个视角:

数据项更喜欢用“好”视图来生成每个面的纹理。平滑项最小化了接缝(例如，相邻面之间的边使用不同的视角生成纹理)可见性。通过图割和Alpha-expansion最小化。

我们将面片在图像投影的梯度值作为我们的数据项（calculate\_data\_costs.cpp,327行）。当投影面积大(高分辨率的近距离正交图像)或梯度幅值大(聚焦图像)时，此项为大。我们使用Sobel算子计算面片投影到图片上的像素的梯度值，并投影范围内的所有像素的梯度值相加：

使用这个数据项会存在一个问题。如果一个视角存在遮挡，比如一个未被重建的行人（如图11所示），显然不应该选择此视角来对遮挡后面的面片进行纹理处理。然而对于梯度的数据项，经常会选择这个视角，因为遮挡物会产生比背景更大的梯度值。因此我们需要进行图片的一致性检查。

图 11 未被重建的行人

我们使用改进的均值漂移算法来进行图片的一致性检查（calculate\_data\_costs.cpp,37行），包含以下步骤：

1.对于每个可以看到面片的视角，计算面片投影的颜色均值。

2.将所有视角视为inliers。

3.计算所有inliers的颜色均值ci的均值和协方差矩阵。

4. 对每个可见面的视角计算一个多变量高斯函数。

5. 清除inliers列表并插入所有函数值大于一定阈值的视角(我们使用)。

6.重复3~5。对于10次迭代，或者直到的所有的项都降到以下，或者inliers的数量降到4以下。

我们使用基于Potts模型的平滑项(view\_selection.cpp,167行)：

1. 颜色调整

从视角选择阶段获得的模型(如图10b所示)包含许多块之间的颜色不连续性。需要进行颜色调整，以最小化缝隙的可见性。我们先进行全局调整(global\_seam\_leveling.cpp,139行)，然后使用泊松编辑进行局部调整(local\_seam\_leveling.cpp,106行)。

全局调整。首先，必须确保每个网格顶点只属于一个纹理块。因此，一调缝隙上的每个顶点被复制成两个顶点:顶点属于左侧的纹理块, 属于右侧的纹理块。现在每个顶点都有一个唯一的颜色。然后，通过最小化以下表达式，计算每个顶点的附加修正：

,

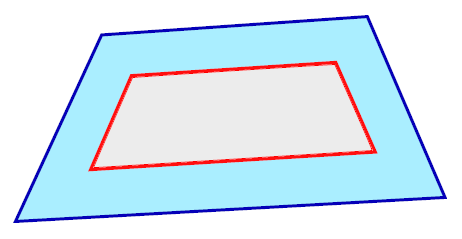
第一项确保缝的左和右边的颜色尽可能相似。第二项最小化同一纹理块内相邻顶点之间的颜色调整的差异。这有利于颜色调整在一个纹理块内是尽可能渐进的。在确定了所有顶点的最优后，每个面片的纹理修正将使用重心坐标从其周围顶点的中插值得到。最后，对输入图像进行修正，将纹理块打包到纹理贴图集中，并将纹理坐标附加到顶点上。

图 12（a）面片（b）随着到v1距离的增加，采样权值从1减小为0。

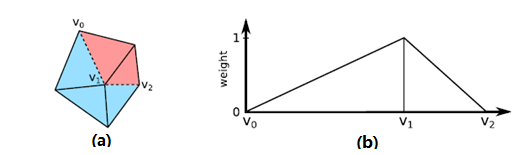
如果和仅通过顶点在两幅图像上的投影取值，会存在一定误差。因为相机的位姿存在一定误差，两个投影并不完全对应于真实物体上的同一点。同时，由于相机的远近不同，查找的单个像素在三维中跨越不同的长度。为了减轻这个问题，我们不仅在顶点的投影中查找一个顶点的颜色值，而且沿着所有相邻的接缝边缘查找。如图12a所示，我们通过对红色图像沿和两条边的颜色样本进行平均，来评估它在红色块中的颜色，即。在上的样本权值为1，随着样本到的距离增大，权值线性减小。这是因为优化后的会根据重心坐标应用到纹理的其它点上，而沿接缝的重心坐标形成从1到0的过渡。我们得到边和的平均颜色后，根据边的长度加权平均得到。我们用同样的方法得到,并将它们带入公式。

图 13纹理块有一个边缘带(浅蓝色)，包含一个外边缘(深蓝色)和一个内边缘(红色)。

为了优化，将公式写为矩阵形式：

,

是公式中堆叠成的向量。和为包含的稀疏矩阵，用于从中选择正确的,,,。公式是的二次型，非常稀疏、对称、正定。我们使用Eigen的共轭梯度法（CG）实现对的求解。我们同时优化了RGB三个通道。

全局调整不能消除所有可见的接缝，因此在全局调整之后，我们执行了局部的泊松图像编辑。为了节省时间和提高内存效率，我们将一个纹理块的泊松编辑限制在20像素宽的边带上(图13中的浅蓝色区域). 我们使用这个条带的外缘(图13，深蓝色)和内缘 (图13，红色) 作为泊松编辑的边界条件: 我们将每个外缘像素的值固定为边缘两侧纹理块颜色的平均值。每个内缘像素的值被固定为它的当前颜色。如果补丁太小，我们就忽略内缘。泊松方程的引导场是条带的拉普拉斯方程。

**3.6.3：测试结果**

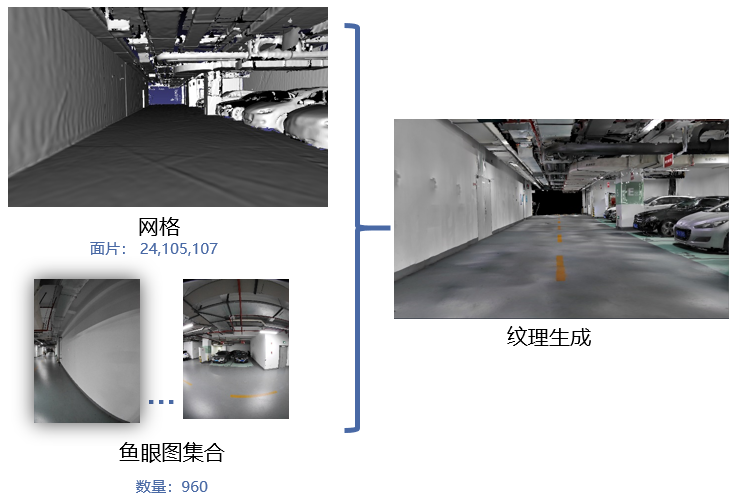
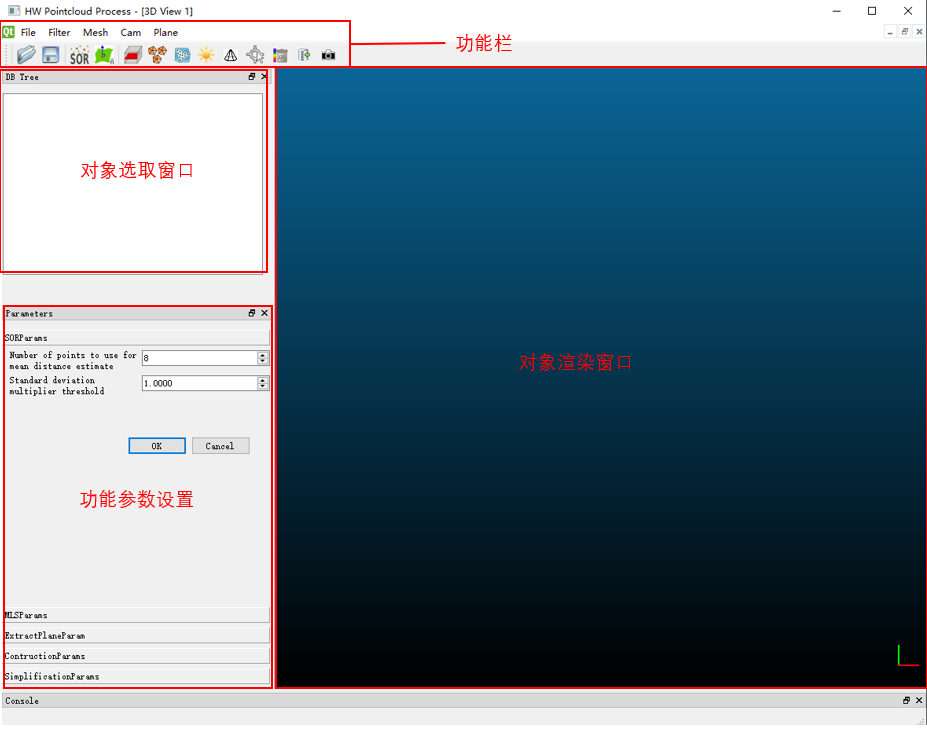


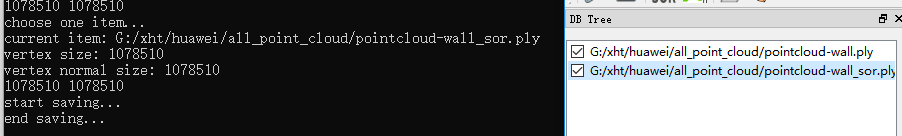
图13 左图：网格图加上鱼眼图 右图：贴上纹理的网格图

本次实验用的是车库的点云。这个模块的输入是系统重建得到的网格，和Navvis提供的960张鱼眼图，最后渲染得到带有纹理的场景，并且渲染出来。

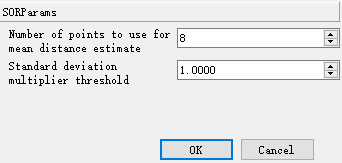
1. **使用说明**
2. **系统功能**
3. 实时渲染大规模点云和mesh
4. 可以使用SorFilter和MLS(移动最小二乘法)对点云进行预处理（滤波和平滑）
5. 对点云提取点云平面
6. 将点云转化为mesh
7. 对mesh进行简化
8. 对mesh贴上纹理
9. **界面介绍**



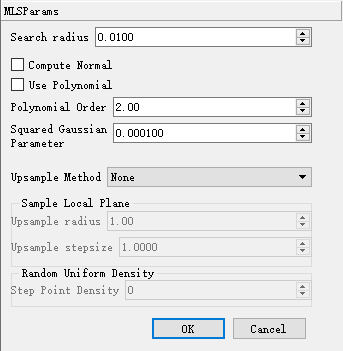
1. **文件管理**
2. Open：选择打开的文件
3. Save：在DB Tree中选择要保存的文件，点击Save，保存文件。文件目录在命令行中显示。
4. Clear：清除所有文件



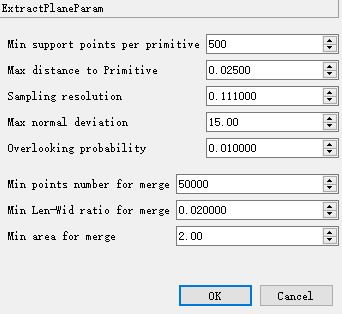
1. **SorFilter**
2. 在filter栏点击SOR或直接点击SOR图标，在Parameters栏设置参数后，点击OK按钮运行SorFilter。
3. Number of points to use for mean distance estimation：计算平均距离时的邻近点个数
4. nSigma：方差的系数项
5. 最大距离=平均距离+nSigma\*方差
6. **MLS**
7. 在filter栏点击MLS或直接点击MLS图标，在Parameters栏设置参数后，点击OK按钮运行MLS。
8. Search radius：搜索半径
9. Compute Normal：计算法向量
10. Use Polynomial：使用多项式
11. Polynomial Order：多项式次数，只有在使用多项式是生效
12. Squared Gaussian Parameter：平方高斯参数
13. Upsample Method：上采样方式，可选None，Sample Local Plane或Random Uniform Density。选择Sample Local Plane需设置上采样半径和步长；选择Random Uniform Density需设置阶跃点密度。
14. **Construction**
15. 在Mesh栏点击Construct或直接点击Construct图标，在Parameters栏设置参数后，点击OK按钮运行Construction。
16. Voxel length：体素长度
17. Volume trunc length：TSDF的截断值
18. **Simplification**
19. 在Mesh栏点击Simplify或直接点击Simplify图标，在Parameters栏设置参数后，点击OK按钮运行Simplification。
20. Target number of faces：简化后面的数量
21. Target percentage(0-1)：简化后面的数量/简化前面的数量（优先级高）
22. Quality threshold：简化质量
23. Preserve Bundary of the mesh：简化过程中尽量不影响mesh的边界
24. Boundary Preserving Weight：0到正无穷。大于1时，会减少移除边界上的点
25. Presere Normal：尽量维持原表面的方向
26. Preserve Topology：避免造成mesh拓扑结构变化的简化
27. Optimal position of simplified vertices：每个顶点被置于使得平方误差最小的位置
28. Planar Simplification：使平面区域的减化结果更优
29. **Texture**
30. 选中需要贴上网格对象，然后点击，让对象进行问题贴图，现在只是用了默认的参数没有用其它的参数，后续可以将参数贴到界面上。
31. **Extract Plane**
32. Min support points per primitive：它指的是提取平面的时候每个需要的最少的顶点数量。数量越少会多很多小平面，特别在复杂的场景中。
33. Max distance to Primitive：离平面的最近的距离，它是对越是大，出来的平面越是粗糙，在边缘区域可能会多一些点云。
34. Sampling resolution：它是在点云投向平面的时候，连通性的判断，并且获取点云的数量最多的平面。
35. Max normal deviation：投向平面的时候，对点云进行判断，如果它的法向量和平面法向量大于我们设定的值，表示这个点云不属于这个平面。
36. Overlooking probability：它是提取一个平面的终止条件，迭代获取平面的时候获取更好的平面的概率不能大于设定的值。
37. Min points number for merge：提取的平面点云，是否需要作为轮廓平面加入轮廓点云。
38. Min Len-Wid ratio for merge：表示狭长点云，可以作为轮廓。
39. Min area for merge：平面点云面积，面积最小的可以保留，这样防止获取很多细小的平面。
40. **操作流程**
41. 点击File,得到目录，选取目录中ply或者e57格式文件，导入模型。
42. SOR去噪，这个步骤，是可选的，只是对噪声比较多的点云操作，一般Navvis扫描得到的点云噪声不多，所以这个步骤可以不用做。



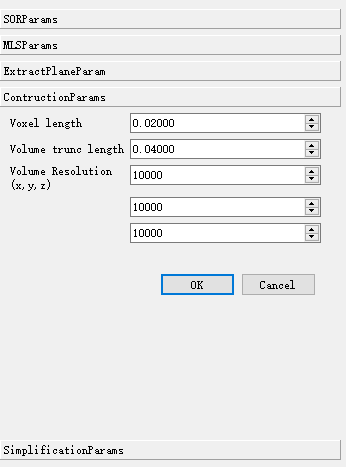
1. 如果需要去噪，它的参数按照默认参数设置，点击OK按钮就可以了。
2. MLS点云平面，这个也是可选的，因为Naviss，扫描的精度不错，平面比较平滑，所以这个步骤一般不用。



1. 如果需要去噪，它们参数设置，一般需要把Compute Normal，和Use Polynomial这个复选框选中，然后其它设置默认，点击OK即可。
2. 平面提取这个步骤对于后期优化需要，对于中间物体去除有用。暂且，这步骤还是在优化中，所以现在是可选项。可以把大平面点云提取出来，即轮廓。



1. 参数一般Min support points per primitive 1500,其它为默认参数。 Min area for merge 看场景的点云的数量来设置。然后点击OK即可。
2. 网格重建



1. 这个是必须项，主要是用这一步来重建网格。参数默认，从对象选取窗口选取点云，然后点击OK即可。
2. Simplification，这是可选项，现在还不需要对网格简化，现阶段操作没有网格简化。
3. 纹理贴图
4. 这是必须项，在网格上贴上纹理，从对象选取窗口选取网格，点击，即可。