

# 基于全景视频的大范围室内场景点云融合与重建

何紫嫣

## 摘要

全向感知在三维重建中变得越来越重要和普及，因为其宽广的视野大大增强了感知能力。然而，缺乏密集和连续的 360° 深度数据集，给传统方法带来了挑战。本文提出了一个点云融合和重建系统来解决大规模室内环境中的这个挑战。首先，我们设计了一种基于 TSDF 的全景点云融合方法，其中引入了全景 TSDF 表示来处理深度估计的有限精度和投影失真问题。此外，我们提出了一个三维重建系统，将深度估计算法整合到以全景视频为输入的 SLAM 中，并使用提出的点云融合方法对其进行处理。最后，我们在一个基于 Potree 的 Web 平台上渐进式渲染大规模点云。本文在 360D 大规模室内数据集和一个具有挑战性的室内场景的真实世界全景视频上评估了提出的方法。大量的实验表明，所提出的方法在大规模室内环境的三维重建方面优于传统方法。

**关键词：**三维重建；全景视觉 SLAM；全景深度估计；全景 TSDF

## 1 引言

三维重建 (3D Reconstruction) 是指利用从真实场景中采集的二维单视图或多视图的图像数据信息，在相关理论的指导下对三维物体或场景的表面信息进行重建的过程。相比于激光三维重建，视觉三维重建成本低廉且采集的信息更为丰富，常作为包括虚拟现实、增强现实、无人驾驶、文物数字化展览等在内的各种新型的数字化应用场景所需要的模型数据的重建生成的方法。

根据应用场景的需求，三维重建结果的数据存储格式主要分为体素 (Voxel)、点云 (Point Cloud) 和网格 (Mesh) 三种。三类格式在一定条件下也可以互相转换。体素可以被视作二维的像素在三维空间中的扩展，表示三维空间中具有一定体积的点；点云则是包括了三维坐标和颜色等信息的无体积的空间点的集合，可用于相对精确地表示三维模型的表面信息，通常在逆向工程中作为中间状态存在，可以对其进行再编辑和再处理；而网格另外记录了边和面的相关信息，可以更好地附加光照等后期渲染效果，通常在商业应用场景里会被选作最终的呈现模式。本文因采用基于深度图像的三维场景重建方法，故将以点云模型作为存储格式用于三维重建效果的呈现。

随着近年来全景影像设备的技术发展，能够提供高分辨率的 360 全景照片，并内置了包括水平矫正、运动防抖等功能在内的很多对采集的原始图像进行预处理的方法的全景相机进入了消费级市场。全景视频具有单帧图片信息包含多、覆盖视角广等特点，作为三维重建系统的输入，将会显著加快采集和处理的速度，并降低采集难度。因此，本文搭建了一个通过全景视频重建室内场景点云模型的系统，且该系统支持在 Web 端完成渐进式渲染，为用户提供了便捷地使用全景视频生成室内场景点云模型的服务。

## 2 相关工作

本文的研究内容涉及到视觉 SLAM、稠密三维重建和大规模点云渲染关键技术，以下对相关背景知识具体展开介绍：

## 2.1 视觉 SLAM

视觉 SLAM 是指只利用摄像头作为外部感知传感器，在没有任何先验知识的情况下，以一组图像序列为输入，输出图像对应的相机位姿，再根据传感器数据实时构建周围环境地图，同时根据构建的环境地图推测自身的定位的系统，目前经典的视觉 SLAM 系统一般包含前端视觉里程计、后端优化、闭环检测和构图四个主要部分。

ORB-SLAM<sup>[1]</sup>是一种基于 ORB 特征，使用了提取关键帧机制的三维定位和稀疏建图的算法。ORB 特征是一种具有旋转不变性的特征，并可以利用特征金字塔构建出尺度不变性，能够在很短的时间内有效提取出图像特征点。同时使用统一的 ORB 特征有助于 SLAM 算法在特征提取与追踪、关键帧选取、三维重建、闭环检测等步骤里具有内生的一致性。ORB-SLAM 利用 Tracking、Local mapping 和 Loop closing 三大线程，并行化地完成了实时提取、追踪特征点，更新局部地图，以及闭环检测修正尺度漂移的工作。

OpenVSLAM<sup>[2]</sup>在 ORB-SLAM 的基础上添加了全景图像接口，兼容更多的相机类型。本文在 OpenVSLAM 的基础上进行修改扩充，配合全景密集深度估计算法，实现了增量式点云生成模块。

## 2.2 稠密三维重建

稠密三维重建通常以彩色图像和对应相机位姿为输入信息，预测彩色图像对应的深度图，并对每帧图像的重建结果进行拼接和叠加处理，从而生成整个场景的稠密模型。

对于全景图像深度估计，近来的研究通过用不同的方式对 ResNet 提取的图像特征进行解释处理，来恢复全景图像的密集深度信息。Zeng 等人<sup>[3]</sup>将深度估计和布局预测相结合，利用初步的深度预测和语义分割结果预测布局深度图，再将布局深度图用于优化初步的深度预测；BiFuse<sup>[4]</sup>将 ERP 和 cubemap 两种投影方式利用 Bi-Projection Fusion 模块结合起来，通过学习 mask，对两种特征进行选择性融合；HoHoNet<sup>[5]</sup>则在提取出特征金字塔的基础上，还使用了卷积和下采样将特征金字塔每一层进行进一步的压缩，并通过 multi-head self-attention (MHSA) 将压缩后的各层特征数据合并优化后再进行解码，通过上采样、卷积和逆离散余弦变换预测密集深度信息。

对于单帧重建结果的拼接与融合，在 Kinect Fusion<sup>[6]</sup>中，深度图被增量式地融合进了一个全局的 TSDF Fusion 中，后者是稠密三维重建的结果的最终表征。在 Elastic Fusion<sup>[7]</sup>中，基于表面的融合和非刚性表面的变形被提出，得以实现全局一致的重建。而这些单一且大小固定的 Fusion 在面对较大的环境时表现将会变差，Matthias 等人<sup>[8]</sup>提出的体素哈希 (Voxel Hashing) 则可以很好的应对这个问题，它使得全局的 TSDF Fusion 不受初始包围盒大小的限制，可以自由地扩展空间，同时在不使用规则的网格数据结构的情况下，依旧保证了对 TSDF 数据的实时访问和更新。

本文将全景图像深度估计算法 HoHoNet 与全景 VSLAM 相结合，实现了一个单目全景 RGB 实时重建的系统，同时基于原始的 TSDF 算法，提出了适用于全景图像的点云数据生成与融合的全景 TSDF 算法。

## 2.3 大规模点云渲染关键技术

大规模点云在渲染时通常无法一次性放入内存进行处理，因此需要分成很多个小部分进行处理和传输。为了在短时间内高质量地渲染海量的点云数据，大部分方法会采用分层空间分区结构，根据表示分辨率的不同来完成对点云数据的层次结构组织与索引编码存储。在此过程中，主要使用的数据结

构包括 R 树、KD 树、四叉树和八叉树等，其中八叉树作为在三维空间中对点云数据进行递归分割的数据结构，无需过多空间存储点云数据分割的结构信息，更适用于本文重建真实场景这种紧凑型点云数据的存储与管理。

Scheiblauer 等人<sup>[9]</sup>提出了可修改的嵌套八叉树（Modifiable Nested Octree, MNO），它使用三维网格来创建子样本，根据原始点云的细节等级对其进行下采样，使得离根近的节点可以占据更大的体素以及更稀疏的下采样样本点云，随着节点层级的升高，其点云的稠密程度增加。Potree<sup>[10]</sup>的八叉树结构在 MNO 结构的基础上，提出了对每个层级对应的点之间的最小间距的要求，使用了泊松蝶采样的方法生成点之间具有最小距离的均匀间隔的子采样，同时为了减少点之间的距离检查量，将每个节点划分为网格，仅计算到相同和相邻单元内的点的距离。此外，为了应对大规模点云数据导致的多分辨率八叉树的层次结构需要大量数据表示的问题，Potree 还构建了用于存储多分辨率八叉树的层次结构的层次结构八叉树。

本文基于 Potree 修改后的八叉树结构进行二次开发，实现了在 Web 端渐进式渲染室内场景的大规模点云数据的功能。

### 3 本文方法

#### 3.1 本文方法概述

标准的三维重建步骤为首先通过输入图像序列进行位姿估计获得关键帧的位姿信息，再通过深度估计模块以获得关键帧的深度图并生成世界坐标系下的单帧点云，最后点云融合模块通过叠加式方式完成关键帧点云数据的融合，输出重建后的 3D 模型。然而，以全景图像序列作为输入，存在以下问题：1) 拍摄时的设备遮挡导致全景图像序列所生成的点云存在重影和上下空洞等失真；2) 低精度全景图像深度估计和坐标映射转换会引入误差，进而使得重建的平面出现波浪化的问题；3) 全景图像采集一般用于空间相对较大的室内场景，每帧图像中深度估计误差导致的错误点很难被已有的点云滤波算法过滤清除。因此，本文基于 TSDF 的思想，提出了能针对性解决或缓解全景点云融合问题的全景 TSDF 表示方法，完成对重建的全局点云数据的滤波操作。

图 1 所示为本算法的全流程示意图，当接收到单个关键帧的彩色图、深度图和相机位姿信息后，会先根据彩色图和深度图，在相机坐标系下生成该帧图像对应的点云数据，之后通过相机位姿信息计算该帧点云数据在世界坐标系中所占据的体素块，得到相应体素块坐标后，对其中每个体素进行并行处理，将空间体素反投影回彩色图和深度图中以更新体素的 SDF 函数值、权重以及颜色信息。示意图中用不同颜色代表了各体素 SDF 函数绝对值的大小，红色为小于预设定阈值的最贴近真实场景表面的体素，橙色次之，黄色为在截断距离以内的相对更远离表面的体素。当所有关键帧接受完毕，处理得到全局 TSDF 表示后，将从红色部分对应的空间体素中提取出最终的表面点云数据。对于整个算法中较为重要的全景 TSDF 和表面点云提取部分，本章将在 3.1 和 3.2 小节中分别进行详细阐述。

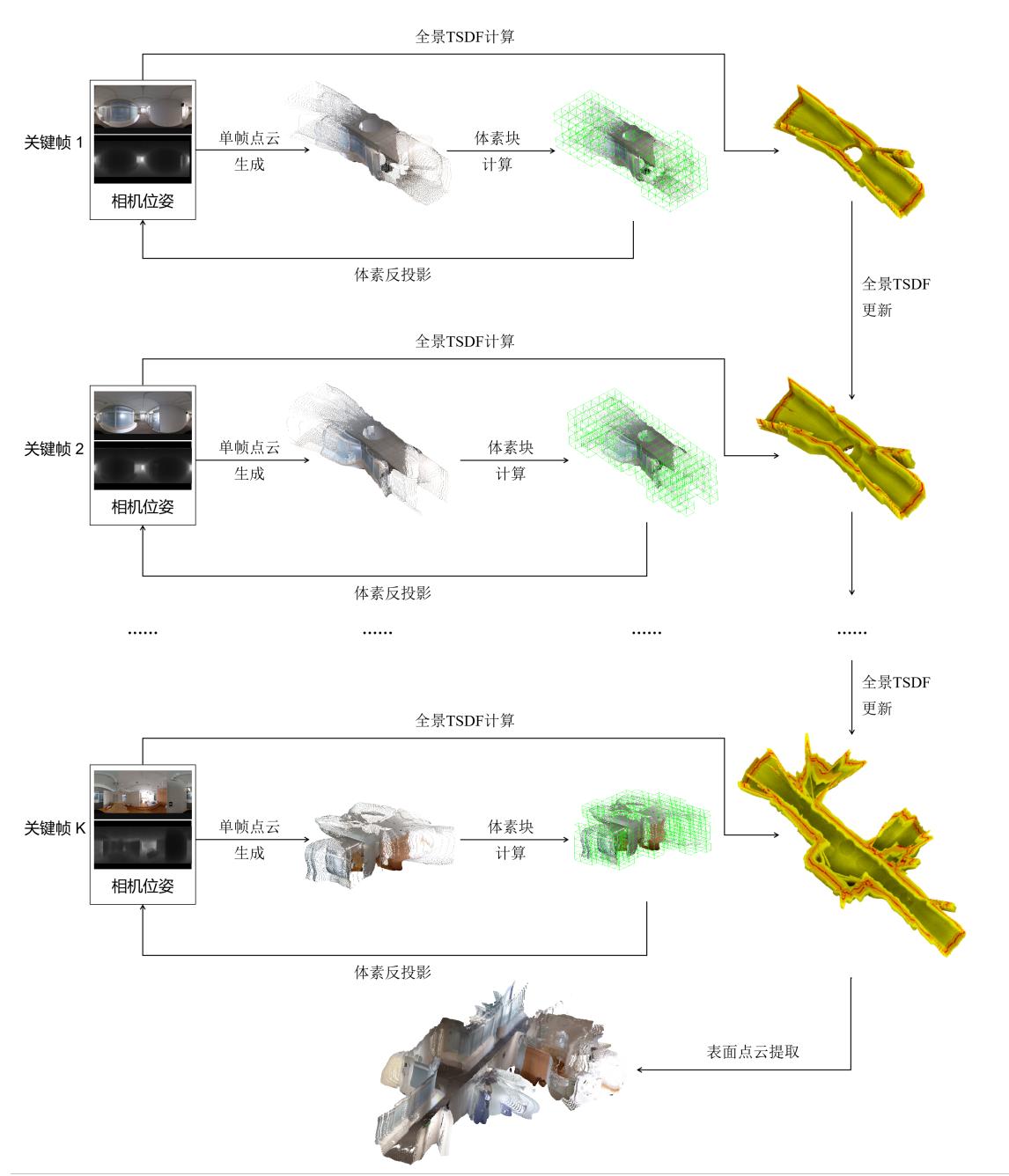


图 1: 基于 TSDF 全局表面重建的点云滤波算法示意图

### 3.2 全景 TSDF 表示

SDF 是用于表示某个空间体素到被观测表面的距离的函数，正值代表该体素位于表面的前方，负值代表该体素位于表面后方，在当前观测点下观测时被表面遮挡，物体的表面由体素的零值面隐式表达，即符号距离为 0 的体素就是观测表面。为了减少计算代价，TSDF 被引入用于替代 SDF，在计算 TSDF 时只考虑在截断距离  $t$  以内的体素，同时用符号距离与截断距离的商计算该体素的 tsdf 值，限制其大小在  $[-1, 1]$  之内。空间中每个体素都存储了截断符号距离 TSDF 和权重  $W$ ，对于输入的每帧深度图，都将有限空间中的每个体素映射到深度图的像素平面上，带权更新该体素的 TSDF 值。

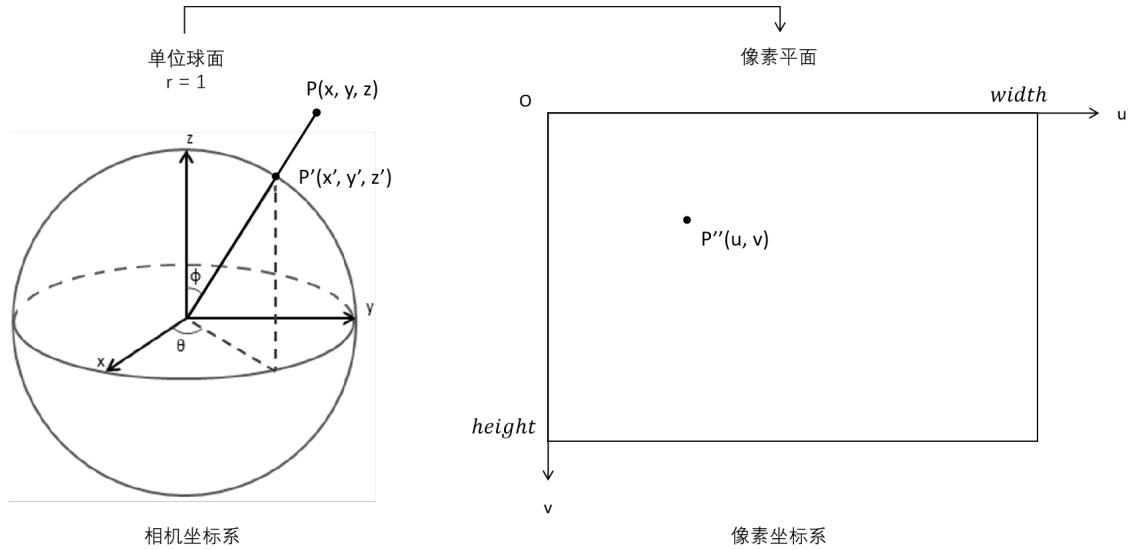


图 2: 全景图三维空间、单位球面与像素平面映射关系示意图

全景 TSDF 在将相机坐标系下的空间体素点投影到像素坐标系下的像素点的映射过程如图 2 所示。首先计算相机坐标系下的空间体素点  $P$  与原点的距离  $r$ ，然后作归一化处理，即对每一维的坐标除以  $r$ ，得到该点在单位球面上的投影，接下来根据球面坐标和直角坐标转换的公式求得和，以在展开的等距柱状投影 (Equi-Rectangular Projection, ERP) 图像中标记出空间体素点投影的位置，最后同样对其进行放缩和平移变换，得到全景深度图像中像素坐标系下的像素点。整个过程中相关公式如下所示：

$$\begin{cases} r = \sqrt{x^2 + y^2 + z^2} \\ \phi = \arctan\left(\frac{z}{\sqrt{x^2 + y^2}}\right) \\ \theta = \begin{cases} \arctan\left(\frac{y}{x}\right) + \pi, & x < 0 \\ \arctan\left(\frac{y}{x}\right) + 2\pi, & x > 0, y < 0 \\ \arctan\left(\frac{y}{x}\right), & x > 0, y > 0 \end{cases} \end{cases} \quad (1)$$

$$\begin{cases} u = \frac{\theta}{2\pi} \times width \\ v = \left(\frac{1}{2} - \frac{\phi}{\pi}\right) \times height \end{cases} \quad (2)$$

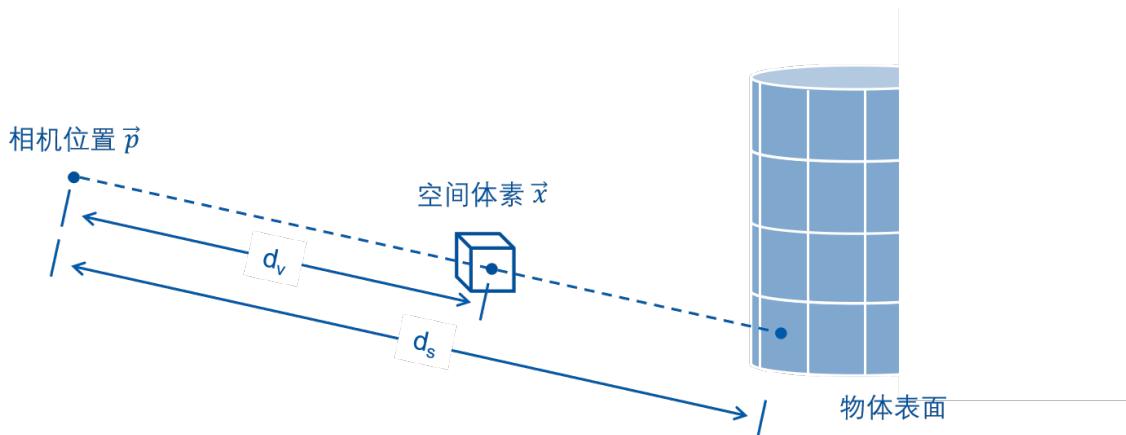


图 3: 截断符号距离函数计算示意图

基于上述映射方式，根据输入的相机位姿信息即可将世界坐标系下的空间体素映射到深度图中，

更新全景 TSDF。如图 3 所示，基本步骤如下：1) 获取空间体素在世界坐标系下的位置向量  $\vec{x}$  和当前帧相机位置向量  $\vec{p}$ ；2) 将  $\vec{x}$  通过相机位姿矩阵的逆矩阵转换到相机坐标系下，再根据公式 (2) 和 (3) 计算得到像素坐标系下的坐标  $(u, v)$ ；3) 记  $d_v = |\vec{x} - \vec{p}|$  为空间体素与相机的距离，记  $d_s$  为直接从该帧深度图中  $(u, v)$  处取出的深度值，则该帧（假设为第 i 帧）的  $sdf_i(\vec{x}) = d_x - d_v$ ；3) 通过  $sdf_i(\vec{x})$  计算  $tsdf_i(\vec{x})$ ，并使用  $tsdf_i(\vec{x})$  和第 i 帧权重  $w_i(\vec{x})$  更新该体素的 TSFD 和 W，更新计算所用公式如下所示：

$$tsdf_i(\vec{x}) = \max(-1, \min(1, \frac{sdf_i(\vec{x})}{t})) \quad (3)$$

$$TSDF_i(\vec{x}) = \frac{W_{i-1}(\vec{x}) \times TSDF_{i-1}(\vec{x}) + w_i(\vec{x}) \times tsdf_i(\vec{x})}{W_{i-1}(\vec{x}) + w_i(\vec{x})} \quad (4)$$

$$W_i(\vec{x}) = W_{i-1}(\vec{x}) + w_i(\vec{x}) \quad (5)$$

较之对每帧的点云数据直接进行叠加和体素滤波、统计滤波处理，通过全景 TSDF 的融合更新有以下四点优势，可以针对性解决或缓解全景点云融合问题：1) 将融合时由每帧图像独立决定点信息的方式，转变为了由所有能观测到该体素的帧共同决定一个空间体素的信息的方式，可以更加精确地确定空间体素与真实场景表面的距离和实际颜色，缓解重影问题；2) 对于某一帧中因为设备遮挡无法观测到空间点或因为 ERP 图像在高纬度处深度估计误差过大需舍弃对应点信息而产生的空洞，可以由其他帧的可信观测结果对其进行填补，有效解决了上下空洞问题，且使得空洞周围的过度更为自然；3) 由于同一个空间点映射到每帧图像中进行深度估计时的误差不是恒定不变的，而呈正态分布，因此综合考虑连续几帧图像的深度预测结果可以减少误差，有效缓解平面波浪化的问题；4) 通过控制权重计算空间体素 TSDF 值的方式，相比于点云统计滤波，能够有效且精确地剔除因全景图像采集数据范围较大而落在模型已重建部分的错误点。

### 3.3 表面点云提取

三维空间中的等值面是指所有满足  $F(x, y, z)=C$ ，其中 C 为某一定值常数的点在空间中构成的曲面。TSDF 本质上可以被看作是等值面的集合，其中值为 0 的等值面对应了物体的表面。在三维重建领域，目前主流的对 TSDF 模型进行表面提取的算法是移动立方体算法 (Marching Cubes, MC) 和光线投射算法 (Ray Casting, RC)。由于本系统的设计目的是完成室内场景的整体点云重建，而非单个虚拟视点的图像生成，因此本文选择了移动立方体算法作为表面点云提取的基础算法。

移动立方体算法是基于线性插值的基本思想提出的算法，通过空间体素八个顶点处函数值，可以确定该体素中是否有等值面经过以及其具体的分布情况。当所有顶点的函数值均大于或均小于等值面对应值时，该体素中不存在等值面；当其中一个顶点函数值大于等值面的值而另一个小于该值时，则可以近似地认为等值面经过了该体素中这两个顶点对应的棱的中间位置。因为空间体素有八个顶点，故一个体素内等值面的分布情况共有  $2^8=256$  种可能，图 4 对其中一些情况进行了列举，深色顶点为函数值小于等值面值的点，即内部点，白色顶点为函数值大于等值面值的点，即外部点。

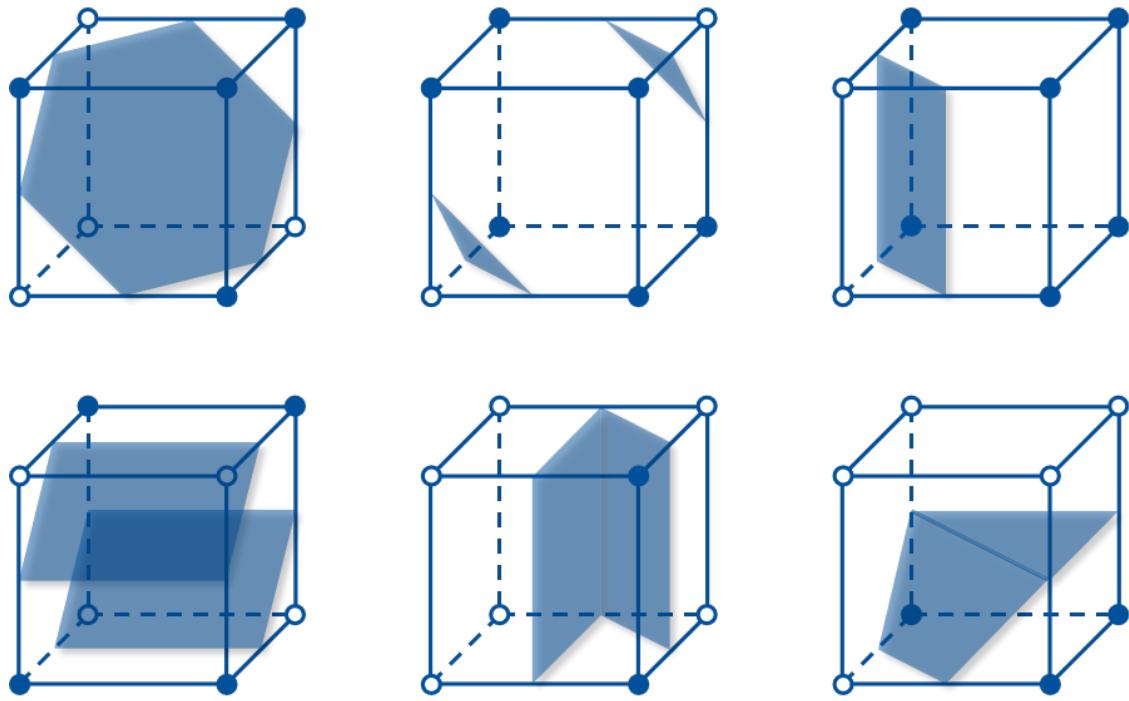


图 4: 等值面分布情况部分示意

在本算法中，TSDF 值大于或等于 0 的点为外部点，小于 0 的点为内部点，将外部点状态标记为 1，内部点状态标记为 0，则可以通过八位二进制数唯一定位一种等值面分布情况。同时，为提高运行效率，本系统事先建立了一个等值面分布情况查询表，以八位二进制数为索引，可以查找到在等值面上的棱中点的相对坐标。最后通过体素块描述信息中记录的体素块位置和体素在体素块内的相对位置，计算出上述等值面上点的世界坐标，放入全局点云中导出。

通过移动立方体算法提取表面点云，相比于以全部 TSDF 值在预设置的阈限内的所有体素中心点为实际点云数据的方法，能够有效减少点云大小，同时提高场景表面的清晰度。

## 4 系统实现细节

### 4.1 与已有开源代码对比

本系统参考的原始论文为 TANDEM<sup>[11]</sup>，这是一个利用单目透视视频进行三维重建的工作，使用 DSO 作为前端视觉里程计估计相机位姿并挑选关键帧，将关键帧传入 CVA-MVSNet 模块估计对应的深度值，最后将 RGBD 图像传入 TSDF Fusion 模块完成三维重建。本系统仿照 TANDEM 系统框架，同样设置了 SLAM、深度估计和点云融合模块，但由于输入变为了全景图，为应对其特殊的投影格式和畸变，每个模块都需要另外寻找或编写适用于全景图的算法。

具体来说，本系统以 OpenVSLAM<sup>[2]</sup>、HoHoNet<sup>[5]</sup>和 Potree<sup>[10]</sup>的源码为基础，实现一个真实场景点云在线生成与渲染的系统。其中 OpenVSLAM<sup>[2]</sup>用于追踪全景视频得到每帧图像对应的相机位姿，HoHoNet<sup>[5]</sup>用于根据全景 RGB 图像预测对应深度图，而 Potree<sup>[10]</sup>则用于将生成的点云转换为八叉树的格式存储，并在 Web 端渐进式渲染，具体使用情况和为系统实现进行的二次开发及功能拓展详见本章后续部分。

## 4.2 系统框架总述

本文基于 Node.js 框架、Bootstrap 框架、ORB-SLAM2 框架、PCL 库和 Potree 库五个基础框架所提供的技术支持，设计了由后台服务器进行数据生成、Web 客户端完成用户交互的点云在线生成与渲染系统。其中，服务器端基于 ORB-SLAM2 框架进行相机跟踪与位姿估计，使用 HoHoNet 网络模型预测全景图的密集深度信息，基于 PCL 库完成了点云重建和初步处理，并基于 Potree 库将点云数据转换为了八叉树的结构存储表示。而客户端则基于 Bootstrap 框架为用户提供了上传并预览视频的功能，同时基于 WebGL 完成了点云模型的渐进渲染和展示。

上述整个系统框架如图 5 所示，图中白色节点为中间数据的内容及结构，深蓝色节点为构成本系统主要功能的五大模块，其细节实现和具体流程详见 4.3 至 4.7 节，而浅蓝色节点为实现用户交互的前端页面内容，其详细信息将在本文的 5.2 节中展示。

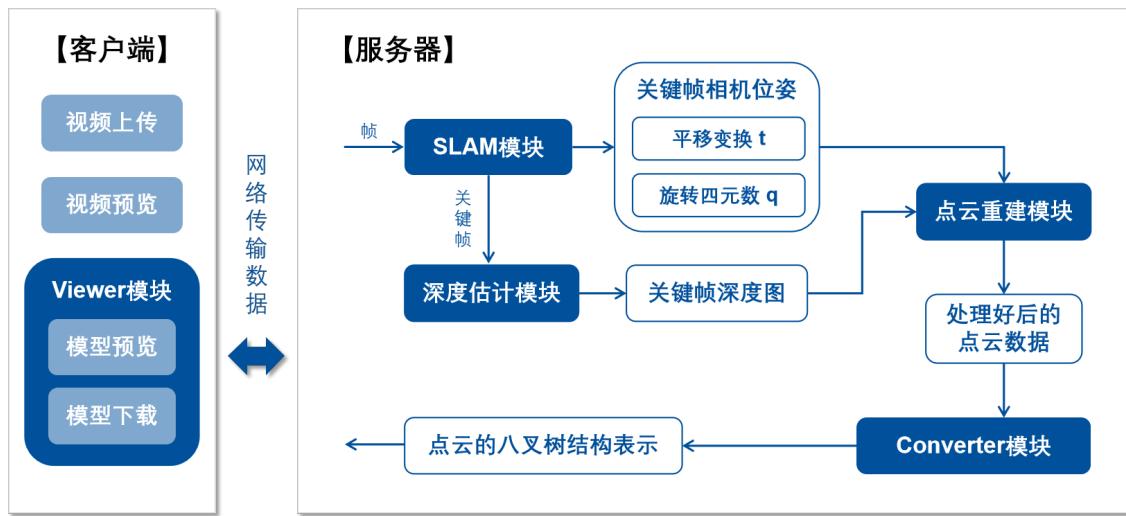


图 5: 室内场景在线点云生成与渲染系统整体框架

## 4.3 SLAM 模块

该模块在本系统中主要负责估计输入帧图像对应的相机位姿，以及进行关键帧的选择。在客户端通过网页上传视频文件后，SLAM 模块从视频中依次取出帧图像进行处理，将挑选出的关键帧对应的图像传给深度估计模块，并将对应相机位姿呈递给点云重建模块。由于本文在 OpenVSLAM 的代码之上进行修改扩充的部分主要集中在 Tracking 线程，且只使用到了单目全景相机作为输入，故该小节中仅对 Tracking 线程中针对单目全景图的相关类、方法进行汇总与阐述。

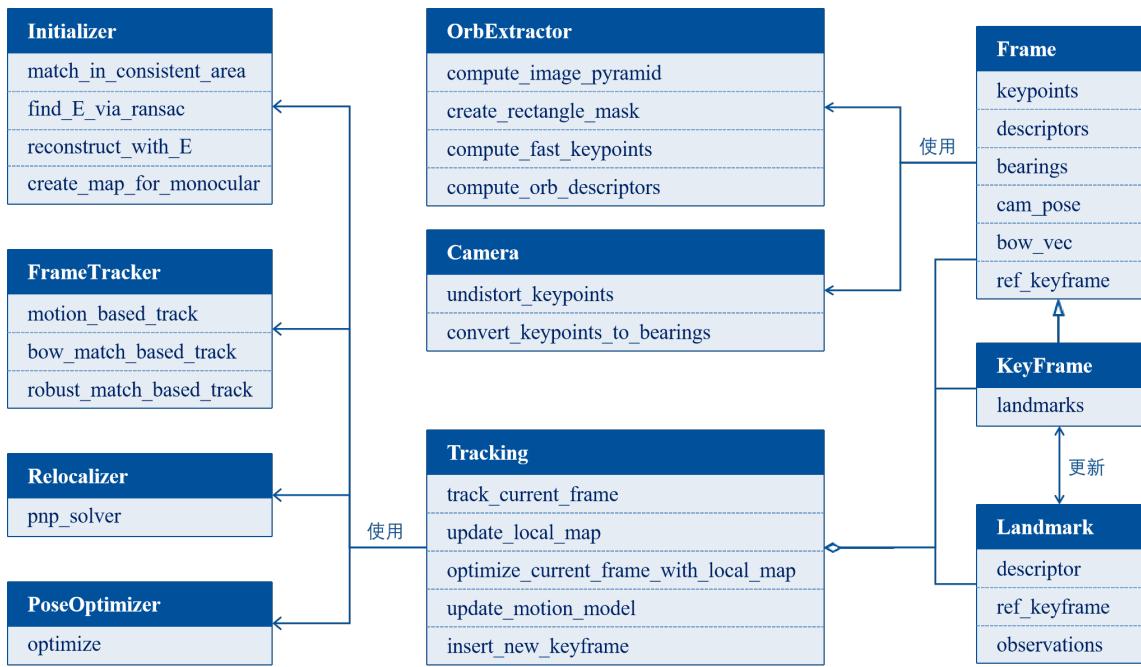


图 6: SLAM 模块核心类图

图 6 为 SLAM 模块的 Tracking 线程的核心类图和主要变量及方法，因为本文在 OpenVSLAM 的代码之上进行修改扩充的部分主要集中在 Tracking 线程，且只使用到了单目全景相机作为输入，故类图中不包括 Local mapping 线程和 Loop closing 线程以及其他非核心函数和辅助工具类。

Tracking 类负责完成线程的整个流程，是聚合了 Frame 类、KeyFrame 类和 Landmark 类的整合类，包括了当前处理的帧及其前一帧、当前帧的参考关键帧和局部地图中全部关键帧，以及局部地图中所有的地图点。Frame 类保存了帧的相关信息，其中较为重要的如特征点、描述子、特征点在单位球面上的位置、相机位姿、词袋向量以及参考关键帧。因为关键帧是从普通帧中选出用于局部建图和闭环检测线程的特殊帧，需要能够维护共视关系以尽可能高效、准确地完成相机跟踪与位姿估计，所以 KeyFrame 类相比于 Frame 类，多保存了该帧所观测到的所有地图点 landmarks，同时提供了对地图点和共视图的添加、删除与修改等维护方法。Landmark 类保存了地图点的相关信息，重要的成员包括描述子、参考关键帧、能被哪些帧观测到以及具体观测关系。

其余类则实现了线程完成位姿估计任务时需要使用到的相关方法。OrbExtractor 类负责提取特征点和计算描述子，Camera 类负责对图像中的特征点按相机类型进行去畸变处理与归一化坐标转换，Initializer 类负责初始化局部地图，FrameTracker 类和 Relocalizer 类提供了三种帧初始位姿估计方法和一种重定位方法，PoseOptimizer 类则负责进行相机位姿优化。

下面将具体描述 SLAM 模块的相机跟踪和位姿估计流程：1) 对于接收到的新输入的帧，将其实例化为 Frame 类的对象，调用 OrbExtractor 类和 Camera 类所提供的方法计算去畸变归一化的特征点与其对应的描述子，将其记录为对象属性后交给 Tracking 类；2) Tracking 类在线程刚启动时调用 Initializer 类提供的方法进行初始化，后者会根据连续两个 Frame 类对象中特征点的对应关系，通过 RANSAC 计算本质矩阵 E，并由此恢复相机的旋转矩阵 R 和平移向量 t，最后使用三角化创建初始地图；3) 初始化完成后，Tracking 类中正常的相机跟踪主循环开始运行，对于接收到的每帧新图像，首先都会依次尝试 FrameTracker 类中提供的运动跟踪、词袋匹配、暴力匹配的方法计算新帧的相机位姿，直到得到的初始位姿误差较小为止，如果三种方法计算的位姿误差都较大，则利用 Relocalizer 类

中的 pnp\_solver 完成重定位；4) 确定初始位姿后，Tracking 类将通过三角化生成地图点的方式对局部地图进行更新，之后调用 PoseOptimizer 类根据局部地图点重投影到当前帧上的投影点，使用 g2o 优化器配合卡方校验对当前帧相机位姿进行优化；5) Tracking 类根据需要新建关键帧，并将新的关键帧交给深度估计模块、点云重建模块以及 Local mapping 线程。

#### 4.4 深度估计模块

该模块在系统中负责接受从 SLAM 模块传来的关键帧的彩色图，预测其对应的全景深度图，交给点云重建模块，以完成该帧的点云生成。所使用的 HoHoNet 网络模型在 PyTorch 中被训练，其模型结构在 2.1.2 部分有详细阐述，表 1 为模型训练中所有超参数及其对应的取值。因为 HoHoNet 模型架构对高分辨率图像损失较大，且实验证明直接预测  $1024 \times 2048$  的高分辨率深度图的质量不如对预测的  $512 \times 1024$  的深度图进行双线性插值的结果，故将该模块处理的图像大小设置为  $512 \times 1024$ ，并根据使用时需要，选择性对其进行双线性插值的上采样。

在现实世界的应用中，通常由搭载摄像头的移动机器人来完成较大的室内场景的环境探索和三维重建，此时如果仍由服务器完成推理计算，将会产生较大的网络传输开销。虽然这种工程导向的开发和优化并不属于本研究的工作范畴，但设计一个可以轻松部署到嵌入式平台的系统依旧有很大的意义和价值。因此本研究考虑使用 PyTorch 提供的 TorchScript 脚本和 LibTorch 接口库，将在 PyTorch 中训练的 HoHoNet 网络模型转换为可以在 C++ 工程中使用的 Torch 脚本，作为深度估计模块接入整个重建系统。

为完成上述转换，在模型训练完成后，通过修改网络模型中各个参数的 `requires_grad` 属性，将其设置为禁止更新的状态，并通过 `model.eval()` 禁用 Batch Normalization 和 Dropout，以避免对单帧图片进行预测时取平均导致的较大失真与神经元的闲置。之后，将输入的图片转换成形状为  $(C, H, W)$  的 Tensor 张量，并通过双线性插值将图片尺寸修改到  $512 \times 1024$ ，转为可以放置到 GPU 上运行的格式。此外，网络模型的输出需要修改为仅输出用 Tensor 张量表示的尺寸为  $512 \times 1024$  的深度图。完成相关准备后，使用 `torch.jit.trace` 函数完成对模型的追踪，记录示例输入在模型中的流动和模型对各个张量上的操作，从而捕获模型结构，并将其转换为脚本模型，导出保存成 `model.pt` 文件。

为在 C++ 环境中成功调用上述保存的模型文件，需要在 PyTorch 官网下载 LibTorch 接口库，并在 `CMakeLists.txt` 文件中配置好路径，添加 `<torch/script.h>` 头文件后即可使用 `torch::jit::load` 函数加载 `torch::jit::script::Module` 类型的模型文件。要成功使用 HoHoNet 网络模型完成关键帧的密集深度预测，需要通过 `torch::from_blob` 函数将归一化后的 cv 图像转换成形状为  $(B, C, H, W)$  的一个可以在 GPU 上运行的输入张量，将其放入存放了 `torch::jit::IValue` 类型变量的 `vector` 中。以该 `vector` 为模型输入，即可得到其中每个关键帧彩色图张量对应的深度图张量，对其进行降维、类型转换等操作后，得到最终需要的深度图。此时，对于深度图中  $(u, v)$  位置的像素点，其深度值可以通过 `depth[u][v].item().toFloat()` 方法获得。

#### 4.5 点云重建模块

该模块负责根据接收到的关键帧彩色图、深度图生成点云数据，并按照各帧的相机位姿对生成的点云进行全局融合。由于全景点云的独特性，普通的点云滤波算法性能较差，本文提出了基于 TSDF

表 1: HoHoNet 网络模型训练参数

	值	补充说明
D	256	LHFeat 的潜在大小, 即第一维的大小
n_components	64	一张图像中一列像素所共享的像素信息的数目
basis	DCT	密集深度预测任务中高度恢复的方法
loss	L1	
batch size	4	
backbone	ResNet-50	
image size	$512 \times 1024$	输入彩色图像预处理放缩后大小及输出深度图的大小 $H \times W$
optimizer	Adam	使用默认参数 $(\beta_1, \beta_2) = (0.9, 0.999)$
epoch	60	
learning rate scheduler	poly decay	学习率调整策略
learning rate	0.0001	初始学习率
power	0.9	多项式的幂/指数

全局表面重建算法的点云滤波, 所涉及算法的详细内容见第三章, 本小节主要阐述点云生成与融合处理的流程, 以及具体使用时的权重设计。

每当新的关键帧被插入序列时, 该模块所在的线程被唤醒, 在预测得到深度图后, 遍历图中每个像素生成点云, 将生成的单帧点云数据转换至世界坐标系下后, 计算得到相关的所有体素块的坐标, 遍历其中每个体素更新 TSDF。在处理完成所有的关键帧后, 使用移动立方体算法进行表面点云的提取, 所提取出的全局表面点云即为本模块最终输出的点云数据。

此外, 在具体使用该模块时, 针对全景图的特殊性设计权重进行重建, 会得到比使用完全相同的默认权重更好的结果。由于全景图像在拍摄时产生的设备遮挡, 与 HoHoNet 在恢复高度信息时高纬度处准确度较低的原因, 输入的每一帧全景图在计算空间区域体素 TSDF 时, 对于相机所在的与地面垂直的一定范围圆柱区域内的体素应不做贡献。同时, 对于体素在当前帧观测表面后方的情况, 因为当前帧所提供的观测信息无法确定后方是否存在其他表面, 为了避免对其他表面的确定产生较大影响, 令该帧对在已知表面后方一个截断距离以外的区域中的体素不做贡献。但是考虑到使用大型专业全景相机在室内环境中拍摄全景视频时, 相机等相关拍摄设备不会距离物体表面太近, 故以相机为中心的一定半径的球体内部不可能存在表面, 需要挖空。以上三种特殊情况下和权重的设计如表 2 所示。

表 2: 全景 TSDF 特殊情况权重设计

r	$\sqrt{x^2 + y^2}$	$sdf_i$	$tsdf_i$	$w_i$
$r < 0.3$	任意	任意	1	5
$r \geq 0.3$	$\leq 0.3$	任意	0	0
$r \geq 0.3$	任意	$\leq -t$	0	0

除以上三种情况外, 因为深度估计时对于远处物体估计结果的可信度没有近处物体高, 故可以倾

向于认为对距离较近的体素的观测结果要比对远处体素的观测结果更可信，从而可以根据体素与相机位置的距离对该帧权重做递减处理，具体权重设计如表 3 所示。

表 3: 全景 TSDF 距离递减权重设计

$r$	$tsdf_i$	$w_i$
$0.3 < r < 0.8$	$\min(1, \frac{tsdf_i(\vec{x})}{t})$	3
$0.8 < r < 1.6$	$\min(1, \frac{tsdf_i(\vec{x})}{t})$	2
$r > 1.6$	$\min(1, \frac{tsdf_i(\vec{x})}{t})$	1

#### 4.6 Converter 模块

该模块负责对经过初步处理的点云数据进行转换，将其组织为多分辨率八叉树的结构，同时构建一个浅层的八叉树记录对应的层次结构信息。经过该模块的转换处理，客户端在运行时，就可以快速加载八叉树点云的层次结构信息，并按照分辨率层次渐进渲染点云数据。

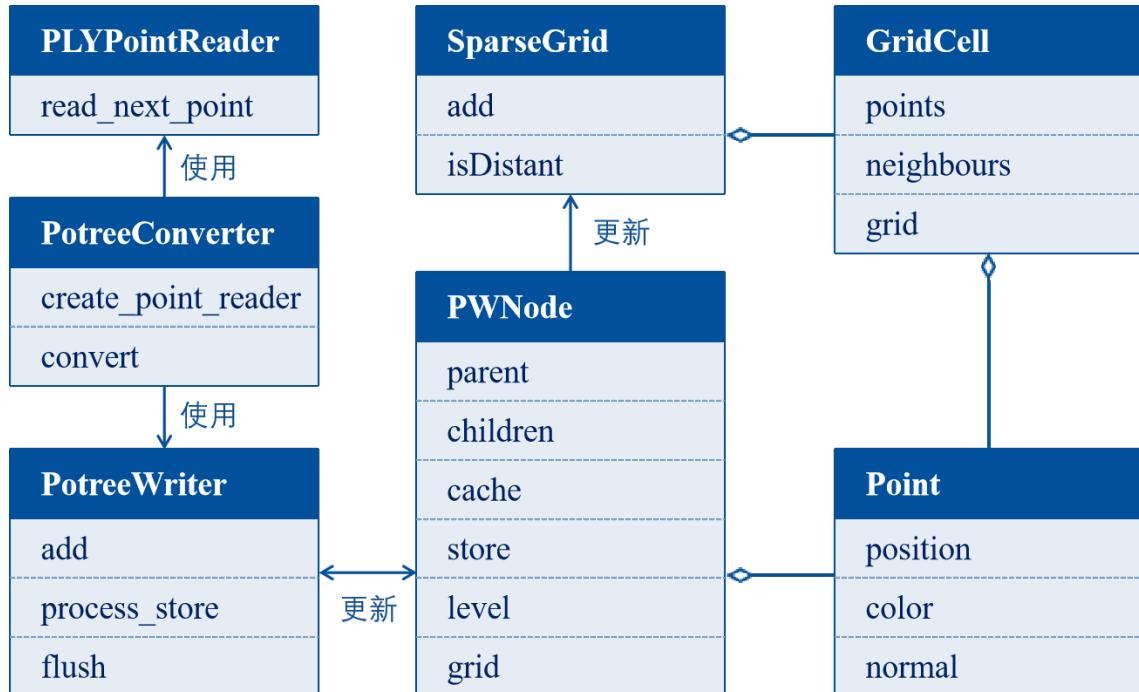


图 7: Converter 模块核心类图

图 7 为 Converter 模块的核心类图和主要变量及方法。Point 类记录了点云数据中每个点的空间位置和颜色、法向等信息；GridCell 类是将点云数据所在包围盒划分为网格后的一个网格单元，其中记录了该网格单元内包含的点数据、相邻的其他网格单元和整个网格的指针；SparseGrid 作为记录全局网格信息的类，可以根据需要设置不同的最小间隔阈限，它是以 STL 库中 `unordered_map` 为基类的派生类，在记录索引与 GridCell 的映射关系之外，还提供了 `add` 方法将符合最小距离要求的点放入网格单元中；PWNode 类是八叉树的节点类，记录了父节点和子节点的指针、当前节点在八叉树中的层级和设置了对应最小间隔阈限的全局网格指针，同时也提供了 `store` 和 `cache` 两个存储区来存放点信息，前者作为暂存缓冲区，当存放的点数量到达阈限后，会将一部分点放入 `cache` 中存储，另一部分则放入子节点中；PotreeWriter 则对八叉树进行了封装，向外提供了将点信息写入 PWNode 的方

法和将点信息、层次结构八叉树写入硬盘的方法；PotreeConverter 类向其他模块提供了接口，让外部可以通过 convert 函数使用上述所有方法完成八叉树的构建。

下面将具体描述 Converter 模块中构建八叉树点云的流程：1) PotreeConverter 类使用 create\_point\_reader 方法构造点云数据的读取器，在 convert 函数中不断从文件中读取点数据；2) 每当读取一个点数据后，都会调用 writer 的 add 方法将其放入自己的暂存区 store 中，当暂存区中的点数目达到预先设定的阈值后，writer 会调用 process\_store 方法，将暂存的点插入八叉树里；3) 八叉树初始时根节点即为叶节点，每次插入时对八叉树的遍历都从根节点开始，如果当前节点是叶节点，则直接放入暂存区 store 中；如果是分支节点，则会调用对应 grid 的 add 方法，通过 isDistant 判断待插入的点与其所在的网格单元及相邻网格单元中其他已有点的距离是否大于该层的最小距离阈限，如果有足够的距离则插入成功，将其放入该节点的 cache 中最终保存，否则根据其空间位置选择当前节点对应的孩子节点进行访问，再次尝试插入；4) 在插入过程中，当一个节点的暂存点的数量超过阈值后，会为该节点创建新的孩子节点，并将暂存区中所有点数据取出重新进行一次插入操作，而在这一次遍历时，之前暂存的节点已经由叶节点变为了分支节点，按照 3) 中遍历到分支节点时的处理方式处理；5) 插入一定量的点后，writer 会调用 flush 将当前的八叉树点云和层次结构八叉树写入一次硬盘，flush 函数中将从根节点开始对多分辨率八叉树点云进行一次遍历，按照超参数 hierarchyStepSize 的值，将 level 在一个 step 内的节点作为浅层层次结构八叉树的一个节点，并将相关信息写入文件中；6) 当原始点云文件中所有点都插入八叉树后，程序会处理完所有暂存区中的点数据，并将最终结果写入硬盘。

## 4.7 Viewer 模块

本模块提供了点云数据的在线渐进式显示，使用户可以通过网页便捷地浏览重建的点云模型，支持多种键鼠操作模式。基于 Node.js 环境和 NPM 包管理工具，使用 Three.js 和 Potree.js 作为主要的第三方库，本系统设计了该模块作为客户端的主要模块，负责模型渲染与用户交互。该模块对第三方库和框架的使用如图 8 所示。

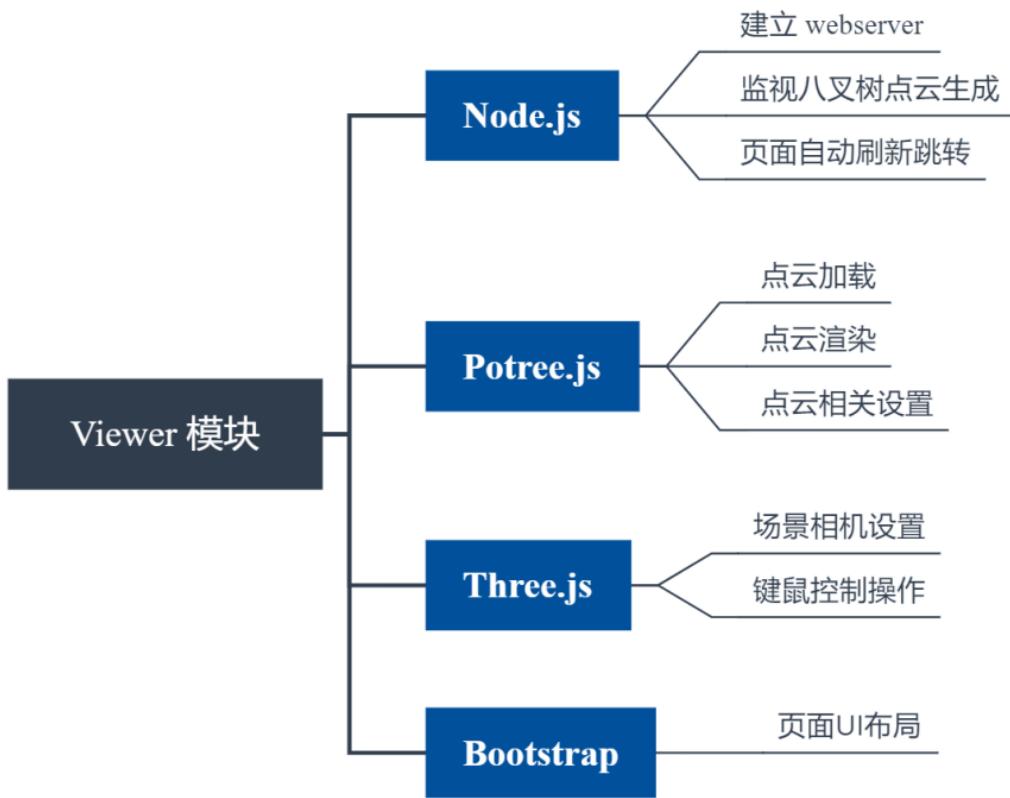


图 8: Viewer 模块与第三方库

Gulp 及其插件是本模块开发和运行时的主要依赖，可以通过 NPM 对其进行安装和管理，在 gulpfile.js 中用 require() 引入并使用。其中，gulp-connect 插件用于建立 webserver，让用户可以使用浏览器远程访问系统页面；gulp-livereload 插件配合 gulp 本身的 watch 方法，实现了对文件的监视和页面内容的自动刷新，使得客户端在检测到八叉树点云构建完成后可以自动跳转到模型渲染页面。

在静态 HTML 主页面中，调用 Potree.js 中的 loadPointCloud 函数可以加载以多分辨率八叉树结构存储的点云数据，配合在页面的显示区域实例化出的一个 Potree.js 提供的 Viewer 类的对象，可以完成点云模型的渲染。同时，Viewer 类对 Potree 中的方法进行了封装，可以使用其对外暴露出的接口，进行点云的相关设置，如点大小、数量、形状等。

Three.js 提供了简单的接口用于初始化摄像机，同时也提供了多种控制器对场景中的相机进行控制，如轨道控制器 OrbitControls、第一视角控制器 FirstPersonControls 等，不同控制器对应不同的键鼠控制操作，由此用户可以根据需要选择不同的浏览模式。最后，该模块使用 Bootstrap 提供的模板样式和 Web 组件，完成了页面 UI 的设计。

## 5 系统展示与实验结果分析

### 5.1 平台简介

#### 5.1.1 客户端设备和服务器设备

本系统旨在支持不同设备通过 Web 网页上传视频并在线浏览生成的室内场景点云模型，因此在性能测试部分，选择了不同性能的笔记本和移动设备作为客户端，测试其展示效果和运行时的性能指标。表 4 所示为客户端设备和服务器设备的相关信息。

表 4: 测试客户端与服务器设备信息

移动设备		PC1	PC2	服务器
CPU	高通骁龙 855	Intel i7-8550U	Intel i7-8750H	Intel i7-7820X
核心数	8	8	12	16
主频	2.84GHz	1.8GHz	2.2GHz	3.6GHz
GPU	高通 Adreno 640	Intel UHD Graphics 620	NVIDIA GeForce GTX 1060	NVIDIA GeForce GTX 1080Ti
显存	/	/	6GB	11GB
RAM	12GB	16GB	16GB	64GB
浏览器	Chrome	Firefox	Firefox	/
操作系统	Android11	Windows10	Ubuntu18.04	Ubuntu18.04

### 5.1.2 数据集

针对从单目全景视频重建真实室内场景模型任务，需要有多个专业采集的室内场景中无人像干扰、高分辨率且帧率在 30 帧以上的全景 RGBD 图片序列，才能对各个模块以及系统整体进行全面的评估测试。但目前并没有一个公开数据集能够同时满足上述各个要求。因此，本文在对所设计系统进行测试时，根据不同部分的算法特性选择了不同的数据集以进行测试与横向对比。

对于深度估计算法测试，本文选择了 Stanford2D3D 作为模型的训练和测试用数据集。该数据集提供了 1413 张分辨率为  $4096 \times 2048$  的全景图像的彩色图、深度图和相机参数信息，这些图像捕获自 6 个大型室内区域，通常选择其中五个区域的图像数据作为训练数据集，而剩下 1 个作为测试数据集使用。

对点云模型生成部分的测试，根据现有公开数据集各自的特点，本文将其划分为了单帧图像处理时间测试和室内场景重建模型整体效果对比两个方面，分别进行测试。前者使用了 Insta360 Titan 11K 的照片样片作为测试数据，而后者选择了 OpenVSLAM 全景视频数据集作为测试数据，该数据集由多个室内场景的全景单目 RGB 图像序列构成，分辨率为  $1920 \times 960$ ，帧率为 30FPS。由于本系统中所使用的全景 TSDF 算法的时间复杂度是与帧分辨率无关的，使用高分辨率的全景图像进行测试才能够对比出该算法相比于点云生成滤波在时间上的性能优势。而重建模型的整体效果则更多取决于多帧点云融合的性能，且 TSDF 算法本身的特点也在于通过连续多帧图像更准确地提取物体表面，因此使用连贯的全景视频进行测试，才能体现出全景 TSDF 算法在减少错误点影响、减轻平面波浪化程度、进行空洞填补以及解决重影问题方面的性能优势。

对渲染系统性能的测试，本文选择了使用 Stanford2D3D 数据集中全景 RGBD 图生成并融合后的单个场景的点云模型作为渲染的测试数据。因为 Stanford2D3D 提供的原始全景图分辨率较高，且相对来说重建后的场景较为完整，更贴近本系统设计时的预期使用场景。

### 5.2 界面分析与使用说明

本系统中客户端用于用户交互的页面主要包括视频上传页面、视频预览页面和模型预览页面，其中模型预览页面提供了多种浏览模式和模型下载功能。

### 5.2.1 视频上传页面

进入系统后会自动跳转到如图 9所示的视频上传页面，用户可以点击按钮选择 mp4 格式的视频文件，或直接拖拽至上传框内，后点击视频下方的上传按钮，等待进度条加载完毕即上传成功。

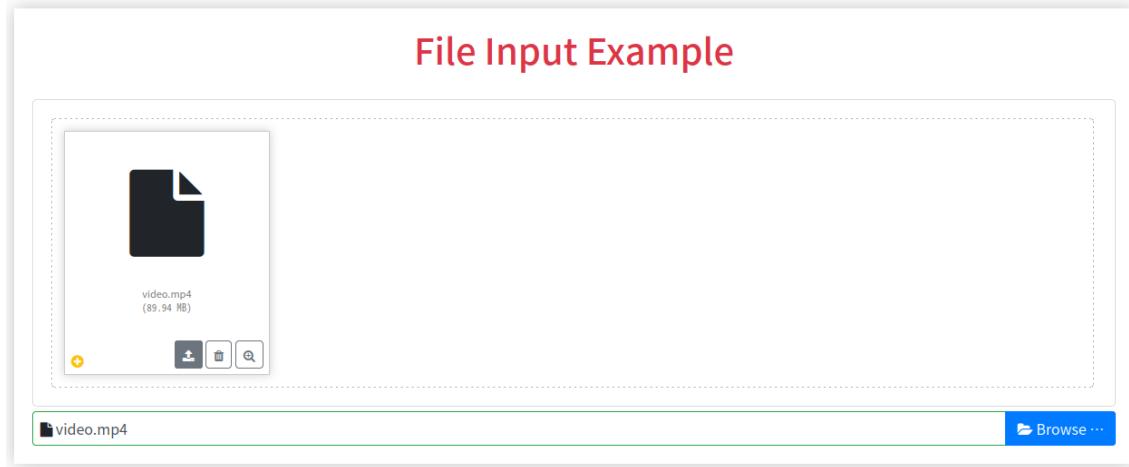


图 9: 客户端视频上传页面

### 5.2.2 视频预览页面

视频上传成功后，系统会自动跳转到如图 10所示的视频预览页面，用户可以播放上传的视频内容，等待服务器完成点云模型的生成重建。

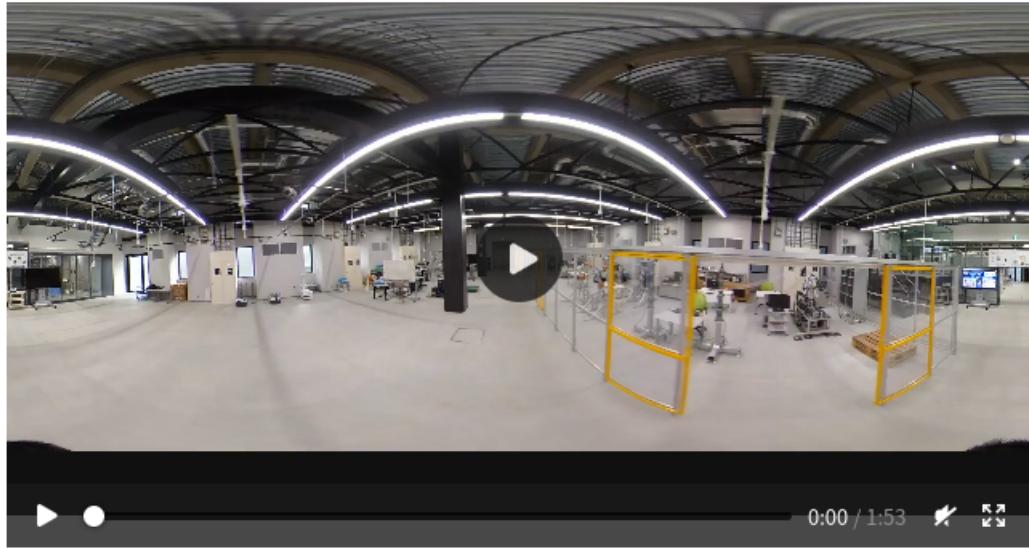


图 10: 客户端视频预览页面

### 5.2.3 模型预览页面

当服务器完成八叉树点云的构建后，系统将跳转到如图 11所示的模型预览页面，在该页面中，用户可以通过 WSAD 键和鼠标浏览生成的室内场景点云模型，同时在 Toolbox 中可以切换不同浏览模

式、调整当前模型中点的数量和大小，以及将点云模型下载到本地保存。



图 11: 客户端模型预览页面

### 5.3 性能测试

#### 5.3.1 深度估计算法测试

由于当前的 SLAM 算法对相机位姿估计的准确程度已经达到了一个比较高的水平，深度估计算法的性能直接决定了本系统中重建点云模型的效果。为选择表现最好的算法，本文对前人研究中较为先进的算法在 Stanford2D3D 数据集上进行了测试。因为全景视频的分辨率接近 2K，故测试时将数据集中的 ERP 图像都缩小为了  $1024 \times 2048$  的大小。相关的性能指标选取了深度估计中常用的标准——MRE、MAE、RMSE、RMSE(log) 和  $\delta$ ，结果如表 5 所示。

表 5: 不同深度估计算法性能测试

Method	MRE	MAE	RMSE	RMSE(log)	$\delta_1$	$\delta_2$	$\delta_3$
Cube 2k	0.9736	1.3581	1.5017	0.8362	0.1171	0.3176	0.5854
ERP 2k	0.4445	0.7906	1.0805	0.8060	0.3343	0.6266	0.7641
Bifuse 2k	0.3883	0.6876	0.9294	0.7999	0.3934	0.6797	0.8121
HoHoNet 2k	<b>0.1295</b>	<b>0.2584</b>	<b>0.4939</b>	<b>0.0859</b>	<b>0.8468</b>	<b>0.9403</b>	<b>0.9773</b>

测试数据表明，使用 HoHoNet 网络模型预测  $1024 \times 2048$  的深度图的性能是最好的，因此本系统中也选择了此模型作为深度估计模块的算法。

#### 5.3.2 点云模型生成测试

PCL 和 Open3D 是目前两个最常用的进行点云处理的第三方库，提供了包括点云滤波算法在内的很多封装好的处理函数，其中 Open3D 还提供了后端高度并行优化的通过 RGBD 图生成点云数据的方法。本文测试了 Open3D 库函数生成点云并对其进行滤波处理与本文提出的全景 TSDF 算法在单帧生

成处理时间上的差异，以及不同分辨率图像下的耗时变化，结果如图 12 所示。数据显示二者在单帧处理上的耗时差异显著，然而，目前对点云进行处理运算的库中，点云数据一般都在 CPU 上并行生成，而 TSDF 算法则一般都在 GPU 上实现，因此显著的时间差异主要是由 CPU 和 GPU 算力不同导致的。

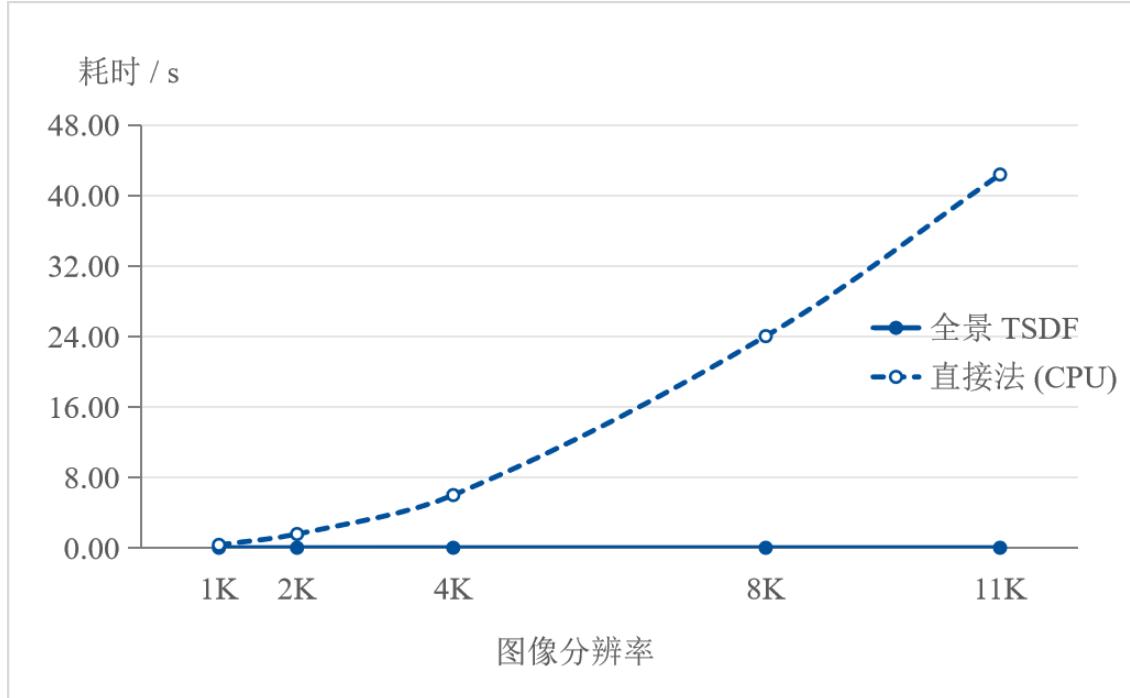


图 12: 全景 TSDF 与直接法 (CPU) 单帧处理时间对比

为了平衡掉算力本身的影响，本文又测试了在 GPU 上根据深度图中每个像素点生成对应空间体素点的耗时，以此代表单帧点云生成的耗时，与本文提出的全景 TSDF 单帧处理的耗时相比较，在二者使用的 GPU 线程数量相同的情况下，耗时结果如图 13 所示。可以发现，直接生成点云的耗时与图像中像素点个数呈线性正相关，而全景 TSDF 算法每次更新处理的都是以相机为中心 5m 范围内的体素，在体素大小一定时处理时长并不会随着图像分辨率的增高而增加。因此虽然在使用低分辨率图像时，后者的耗时显著高于前者，但随着分辨率的增高，差异逐渐减小，全景 TSDF 算法的优势逐渐显现。此外，在 CPU 上将空间点云信息按照第三方库中定义好的数据结构进行存放，才能更好地使用其提供的方法和接口进行后续处理，而这部分的耗时在测试时并未计入，因此直接生成点云的实际耗时会更长。

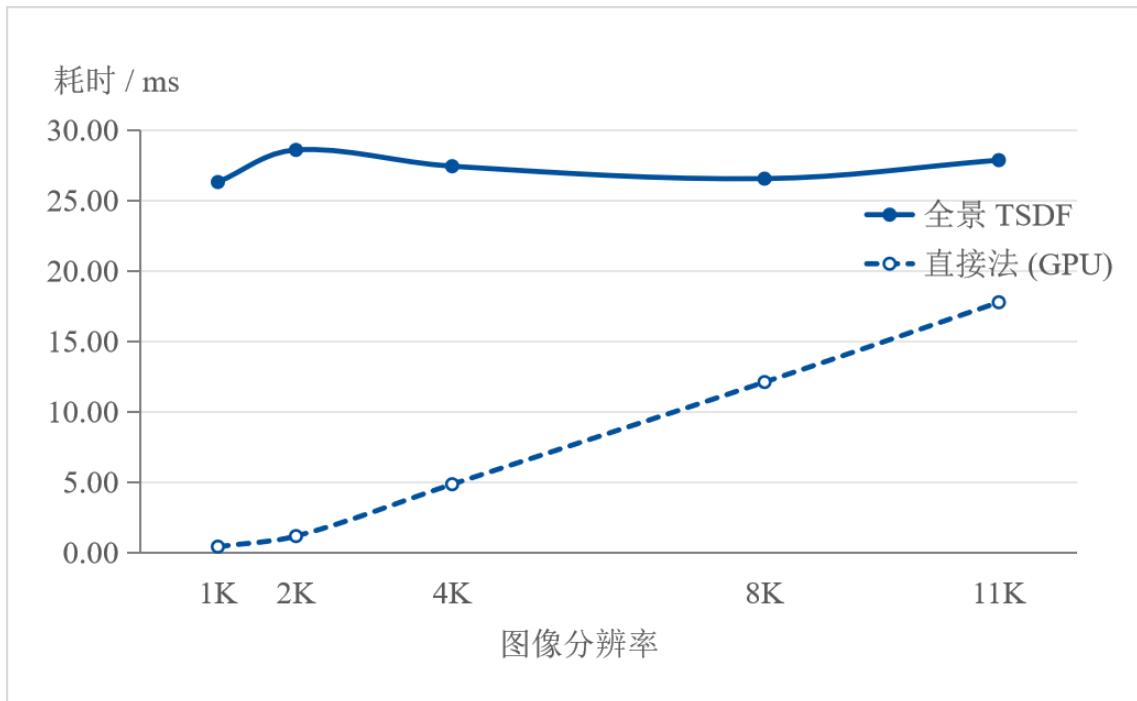


图 13: 全景 TSDF 与直接法 (GPU) 单帧处理时间对比

两种方法生成的整体模型效果对比如图 14 所示，可以发现，在整体场景点云模型的生成重建中，传统的点云滤波算法并不能够很好的过滤掉落在主模型中的错误点，而全景 TSDF 算法在确定每个体素时都综合考虑了多帧的观测结果，并根据距离不同设置了不同的权重，因此可以有效地过滤错误点并填补孔洞，甚至平滑掉一部分深度估计的误差，虽然这样的平滑也一定程度上牺牲了模型清晰度，但相对来说比前者更能被接受，且后续基于此模型进行修改或二次创作，相比之下会更加方便。



图 14: 全景 TSDF 与直接法生成全局模型对比

### 5.3.3 渲染系统性能测试

帧率是指页面在单位时间内被刷新的次数，与分辨率和显卡处理能力有关，一般当帧率达到 24fps 时即可基本满足人眼识别的要求。本文使用了 Stanford2D3D 数据集的 5a 区域中的 hallway\_1 场景作为本系统在各设备上渲染性能的测试模型，该模型共包括 41979036 个点，原始文件大小为 1.4G，其中包括细节信息较多的走廊交汇处场景和规则性较强的室内弱特征场景。图 15 所示为不同设备在两个场景中的渲染效果，对应帧率如表 6 所示。



图 15: 不同设备点云模型渲染效果对比

表 6: 不同设备点云模型在不同特征强度下的渲染帧率对比

设备	复杂场景	简单场景
移动平台	25-30fps	35-41fps
PC1	22-29fps	33-39fps
PC2	27-36fps	46-54fps

#### 5.4 创新点与主要工作内容

本文系统设计的目标是实现一个真实场景点云在线生成与渲染的系统，该系统可以接收用户上传的全景视频文件，结合全景 SLAM 框架和目前性能表现最好的全景图深度估计神经网络模型，以预测生成 3D 场景的全局点云模型。并根据全景图像的特性，完成滤波处理。生成的点云数据将被转换为易快速渲染的双八叉树结构进行存储，通过 WebGL 在网页页面中完成渐进式渲染，为用户提供便捷友好的交互功能。

基于上述系统设计目标，本文在 OpenVSLAM<sup>[2]</sup>、HoHoNet<sup>[5]</sup>和 Potree<sup>[10]</sup>的源码的基础上进行的创新和主要工作包括：1) 针对全景图像序列输入，设计了全景视觉 SLAM 框架。通过全景图像深度估计，实现全局点云模型的增量式生成；2) 提出了全景截断符号距离函数 (Truncated signed distance field, TSDF) 表示方法，在此基础上，改善了基于 TSDF 全局表面重建的点云滤波算法，以改善由于低精度全景图像深度信息和投影失真导致的点云质量差等问题；3) 基于 WebGL 实现了网页端室内场景点云模型的渐进式渲染展示。

## 6 总结与展望

近年来，随着全景影像设备中创新技术的研发和产业化，非专业人士也可以通过 360 全景相机较为容易地采集到高质量的全景照片。而因其包含信息多、覆盖范围广的特点，在配合视觉 SLAM 系统完成室内大型真实场景点云的实时生成重建任务时，采集处理速度会显著快于普通的透视相机。同时基于 Web 端的交互，使得用户可以不受开发环境约束，便捷地通过全景视频生成场景点云模型，浏览其渲染效果，并按需下载原始点云模型进行存档或二次开发创作。

本文设计的真实场景点云在线生成与渲染系统可以有效地利用当前 360 全景相机的硬件优势，使用全景 SLAM 系统估计相机位姿，通过神经网络预测图像的密集深度信息完成全局重建，并在 Web 端进行渲染。下面是本文主要工作的总结：1) 对比测试当前较为先进的全景密集深度预测的神经网络模型的性能，将其模块化后导出，与全景 SLAM 框架结合，形成通过全景图进行实时点云生成重建系统的雏形；2) 提出了能针对全景点云融合问题的全景 TSDF 表示方法，利用 TSDF 算法的思想，完成对重建的全局点云数据的过滤操作，有效缓解了空洞、错误点和波浪平面的问题；3) 设计并实现了基于 WebGL 的室内大场景点云模型的渲染系统，并完善了 Web 端的用户交互功能，使用户可以通过网页页面完成室内场景点云模型的在线生成和渲染。

此外，本系统尚存在很大可以改进提升的空间与可扩展的功能：首先，深度估计网络模型预测的准确率极大程度上决定了最后整体模型的质量，本文对以 RGB 彩色全景图为输入、16 位深度全景图为输出的神经网络模型进行了模块化导出，并设计了解耦合的接口，从而在后续的开发优化中可以便捷地更换其他性能表现更好的网络模型，以进一步提升重建点云模型的质量。其次，目前成熟的视觉 SLAM 框架中还包含了闭环检测的模块，能够提升大场景下的定位精度，但本系统在进行关键帧点云生成时并没有使用到这个部分，如果能够在全局融合的同时根据帧编号分离出对应的点云，并进行位置的修改，将会支持更大场景的重建。最后，虽然本系统的设计目的之一是更好地利用高性能的 360 全景相机设备，且服务器端进行点云生成时也支持增量式全局重建，但因为条件所限并未能够将全景相机设备作为直接输入，而是选择了以视频的方式间接上传，后续的工作中可以配合 ROS 机器人系统完成实时的采集与重建。

## 参考文献

- [1] MUR-ARTAL R, TARDÓS J D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras[J]. IEEE Transactions on Robotics, 2017, 33(5): 1255-1262. DOI: 10.1109/TRO.2017.2705103.
- [2] SUMIKURA S, SHIBUYA M, SAKURADA K. OpenVSLAM: A Versatile Visual SLAM Framework [J/OL]. SIGMultimedia Rec., 2022, 11(4). <https://doi.org/10.1145/3530839.3530849>. DOI: 10.1145/3530839.3530849.
- [3] ZENG W, KARAOGLU S, GEVERS T. Joint 3d layout and depth prediction from a single indoor panorama image[C]//European Conference on Computer Vision. 2020: 666-682.
- [4] WANG F E, YEH Y H, SUN M, et al. Bifuse: Monocular 360 depth estimation via bi-projection fusion

[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 462-471.

- [5] SUN C, SUN M, CHEN H T. Hohonet: 360 indoor holistic understanding with latent horizontal features [C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 2573-2582.
- [6] NEWCOMBE R A, IZADI S, HILLIGES O, et al. KinectFusion: Real-time dense surface mapping and tracking[C]//2011 10th IEEE International Symposium on Mixed and Augmented Reality. 2011: 127-136. DOI: 10.1109/ISMAR.2011.6092378.
- [7] WHELAN T, LEUTENECKER S, SALAS-MORENO R, et al. ElasticFusion: Dense SLAM without a pose graph[C]//. 2015.
- [8] NIESSNER M, ZOLLMÖFER M, IZADI S, et al. Real-time 3D reconstruction at scale using voxel hashing[J]. ACM Transactions on Graphics (ToG), 2013, 32(6): 1-11.
- [9] SCHEIBLAUER C, WIMMER M. Out-of-core selection and editing of huge point clouds[J]. Computers & Graphics, 2011, 35(2): 342-351.
- [10] SCHÜTZ M, et al. Potree: Rendering large point clouds in web browsers[J]. Technische Universität Wien, Wieden, 2016.
- [11] KOESTLER L, YANG N, ZELLER N, et al. TANDEM: Tracking and Dense Mapping in Real-time using Deep Multi-view Stereo[EB/OL]. arXiv. 2021. <https://arxiv.org/abs/2111.07418>.