



奥比中光 3D 视觉创新应用竞赛

轻量化、松耦合的 手持 RGB-D 室内环境实时重建系统

陈钧涛
朱安琪
郭超政

同济大学
TONGJI UNIVERSITY

摘要

近年来，消费级 RGB-D 传感器不断推出，让基于深度相机的三维扫描和重建技术得到了飞速发展。现有的基于 RGB-D 传感器的三维重建技术大多依赖于强大的算力如高性能显卡，其应用场景严重受限。轻量化的应用模式、与用户良好的实时交互体验，已成为学界和产业界的关注热点。

出于上述考虑，本项目以**轻量化、移动化的应用模式**以及**实时渲染交互**作为设计出发点，面向室内环境实时重建任务，具体设计了两套**松耦合的手持 RGB-D 三维重建系统**，包括：面向嵌入式应用的离线实时重建渲染系统与面向 PC 端的在线实时重建系统。

在**面向嵌入式应用的离线实时重建渲染系统**中，用户只需手持连接 Astra Pro 摄像头的 Zora P1 开发板离线端对室内环境进行重建，就可以通过开发板所连接的显示器实时观察到重建的三维模型，保障了良好的交互体验。该系统中开发板首先进行数据采集，通过网络将数据发送至服务器，并接收服务器的位姿估计结果进行三维模型的融合与渲染。通过网络连接至服务器端意味着对服务器的物理位置没有额外要求，解耦合的架构对于嵌入式应用开发具有重要意义，在**搭载开发板的机器人或其他嵌入式应用场景**中，开发板只需要获取数据，将消耗大量算力的三维重建交由服务器进行计算，充分利用开发板有限的算力进行模型融合与可视化交互。

在**面向 PC 端的在线实时重建系统**中，考虑到开发板算力有限，我们将其作为数据采集装置，用户可以使用开发板进行数据采集，实时在在线端 PC 看到重建效果并实现交互。开发板与在线端通过网络进行连接，普通消费级计算机即可满足本系统中三维重建对性能的要求，因此本系统可以应用在 PC 等具有显示交互功能的设备上。

在技术层面，我们针对系统各个模块的性能选择了较为合适的方案，如开发板离线端采用基于八叉树表示的 **CPU 实时 TSDF 融合与渲染**；而算力较强的服务器端选用**层次式全局联合位姿优化方法**；PC 端实现了一站式**三维重建框架**，其中包含了基于环境度量模型的位姿估计以及重建算法。系统各组件之间的连接通过 ROS 通信统一进行实现。

本文的结构如下：第一章首先介绍项目背景与亮点，以及广阔的市场前景；第二章对项目架构以及部署进行阐述；第三章对关键创新点以及核心算法作出了详细介绍；第四章展示了项目的重建效果以及量化测试结果；第五章记录了团队研发的过程；第六章则记录了研发过程中遇到的问题与解决方案，以及对开发板性能限制的阐述。

目录

1	背景介绍	4
1.1	项目背景.....	4
1.2	现有方案缺陷及本项目特色.....	4
1.3	产品化及未来市场潜力.....	6
2	项目概述	7
2.1	系统架构.....	7
	面向嵌入式应用的离线室内环境实时重建系统.....	7
	在线室内环境实时重建系统.....	8
2.2	部署环境.....	8
	硬件构成.....	8
	环境依赖.....	9
	代码说明.....	9
3	技术优势分析	10
3.1	关键技术创新点.....	10
	创新点 1: 基于八叉树表示的 CPU 实时 TSDF 融合与渲染.....	10
	创新点 2: 基于环境度量模型的位姿估计算法	11
	创新点 3: 层次式全局联合位姿优化	11
	创新点 4: 基于 ROS 通信的离线与在线离线室内环境实时重建系统...	12
3.2	核心算法设计.....	12
	核心算法 1: 离线端 CPU 实时 TSDF 融合与渲染.....	12
	核心算法 2: 在线端位姿估计及重建	14
	核心算法 3: 服务端全局位姿估计与重建	16
4	成果展示	20
4.1	系统重建效果展示.....	20
	离线室内环境实时重建效果.....	20

在线室内环境实时重建效果	22
4.2 系统性能量化测试	23
5 研发过程记录	24
5.1 内参标定	24
5.2 相机配置	27
5.3 ROS 网络通信	28
5.4 FastFusion 移植	31
5.5 BundleFusion 环境配置	32
6 问题记录	33
6.1 已知 Bug 记录	33
Zora P1 配置 OpenNI2 相关 Bug	33
FastFusion 移植相关问题记录	33
BundleFusion 移植相关问题记录	34
6.2 性能限制	34
KinectFusion 尝试	34
FastFusion 性能限制	35
参考文献与资料	36

1 背景介绍

1.1 项目背景

三维重建是计算机视觉领域的研究重点之一，利用视觉图像中的色彩、纹理、深度等信息进行三维空间中物体的形状和位置信息的恢复，对真实世界环境中的物体进行数字化。利用三维重建技术将目标物体构建为便于处理的数据模型，得到的三维模型能够被应用到后续的不同场景中。

近年来，消费级 RGB-D 传感器不断推出，如微软的 Kinect，华硕的 XTion 以及奥比中光自主研发的 3D 传感器 AstraPro、AstraS 等一系列的产品，这些传感器价格低廉，体积适当，操作方便，并且易于研究者和工程师进行开发，让基于深度相机的三维扫描和重建技术得到了飞速发展。

ORBEC Astra 系列 3D 传感摄像头采用单目结构光技术，具有高精度、低功耗、响应迅速、稳定可靠的优点。开发人员可以通过选择不同的版本自由地在短距离、长距离和高分辨率 RGB 摄像机之间切换，以满足个性化需求，其能够覆盖近距离和中远距离的多种室内场景，帮助我们更精确、快速地构建三维场景。

同时借助 ZORA P1 开发板，我们能够实现面向嵌入式应用的实时三维重建系统。通过将相机搭载在轻量级的嵌入式硬件设备上，用户能够在轻量化的平台上手持完成实时的三维重建，摆脱以往重建系统依赖大型硬件设备的物理限制，提供更好的用户体验。

1.2 现有方案缺陷及本项目特色

目前市面上现有的室内三维重建方案存在着以下的问题及缺陷：

首先现有的一些 VR 实景方案基于 360 全景相机，使用全景照片来还原室内场景，并提供 VR 浏览。但是全景相机得到的室内实景并非真实的空间模型，用户仅能在固定视角进行浏览，并不能提供空间化的真实实景重建。这样在 VR 体验的沉浸感上远不如基于三维视觉重建得到的真实模型。

其次现有三维重建方案往往需要独立运行在高性能计算机上，传感器设备需要直接与大型的硬件设备相连。而大型的硬件设备难以移动，同时有线连接的方式也会限制传感器扫描移动覆盖的范围，这将极大地限制现有的三维重建系统的使用场景。

而本方案利用网络通信实现了松耦合的模块化三维重建系统，传感器设备能够搭载在嵌入式平台上，摆脱对大型硬件设备的物理限制，同时也能实现高精度的三维重建。

具体而言，本项目的优势有以下几点：

- **轻量化的手持平台**

本项目中用户在进行三维重建时，只需要手持传感器以及与之相连的嵌入式开发板，开发板将通过网络连接实现完整的系统通信。传感器与嵌入式平台较小的体积以及轻便的重量，使得用户能够轻易地手持设备，完成室内的扫描与重建工作，提供更好的用户体验。

- **基于网络通信无线互联**

本项目基于 ROS 完成高效的网络通信，实现三维重建系统模块化的松耦合互联。这意味着用户的手持设备不再依赖于有线连接，手持设备能够通过无线网络通信与其他模块实现交互。无线的手持设备不会存在有线设备对活动范围的限制，用户能够自由地行动并完成重建工作。这样也意味着用户能够深入更加复杂的环境中进行重建，这大大拓展了系统的使用场景。

同时基于网络通信的架构，解除了整个系统对于大型硬件设备的物理限制。系统的服务器模块能够对用户实现完全透明，用户使用的只有传感器及相应的嵌入式平台。如此不再需要搬运大型的硬件设备至重建现场，只需要提供可靠的互联网接入，用户的手持设备就能够与系统的其他模块相连，完成重建工作。

- **高灵活性的松耦合架构**

网络通信实现了松耦合的系统架构，意味着系统模块能够轻易地实现更换或重构。这使得系统拥有极高的可拓展性，理论上系统的手持设备平台能够轻易地进行替换，根据不同的实际场景使用不同精度的传感器，为用户提供更多的选择，更好地平衡成本以及精度。

同时系统的服务器模块也不需要与用户的手持设备紧密绑定，服务器模块能够采取虚拟化或者更加复杂的架构，为不同的用户端提供服务。当服务器模块出现故障时，松耦合的架构方便系统对服务器模块进行及时的替换，使得系统的可维护性提高。

- **精确可靠的重建效果**

系统基于领域内的顶尖算法进行开发，能够实时地完成高精度的 RGB-D 三维重建。相较于单目以及双目视觉重建方案，在重建的实时性及精确度上都有着明显的优势。

- **低廉的设备成本**

市面上现有的手持激光扫描设备，价格在数万元至数十万元。高昂的价格导致手持三维重建设备难以得到普及。而本项目中使用的传感器设备及嵌入式平台，价格仅在千元水平。并且目前开发使用的仍是通用型的传感器与嵌入式开发板，若采用定制化的硬件设备，能够进一步降低设备的成本。

1.3 产品化及未来市场潜力

作为一种新颖的信息载体技术，三维重建模型以其高逼真度、强代入感、可修饰性等无可比拟的优势逐渐影响着人们的生活。借助三维重建技术，我们能够快速地得到室内一个区域，一个房间甚至是一栋房屋的数字三维模型。目前针对室内场景的三维重建在游戏、电影、测绘、定位、导航、自动驾驶、VR/AR、工业制造以及消费品领域等方面得到了广泛的应用。

在房屋销售租赁的应用场景中，借助三维重建，我们能够使用数码设备来浏览房屋室内场景，无需到达现场就能够观察到逼真的现场环境、感受沉浸的漫游体验；VR 看房的新型展示模式能够以一种对用户更友好的方式展示房屋的户型、面积等信息，改善消费者的用户体验。

针对室内房屋的三维重建也能够用于家居家装的三维实景指导，三维重建能够快速获取房屋的尺寸信息，让用户精确感知空间大小，方便购置家具；将房屋的三维模型与 AR 技术相结合，能够数字化模拟装修的效果，帮助用户观察家装效果。

在其他更多的场景中，三维重建都有着广泛的应用，如在工业制造中，利用三维重建技术，能够便捷快速地获取目标场景的数字模型，省去繁琐的人工建模过程；

2 项目概述

2.1 系统架构

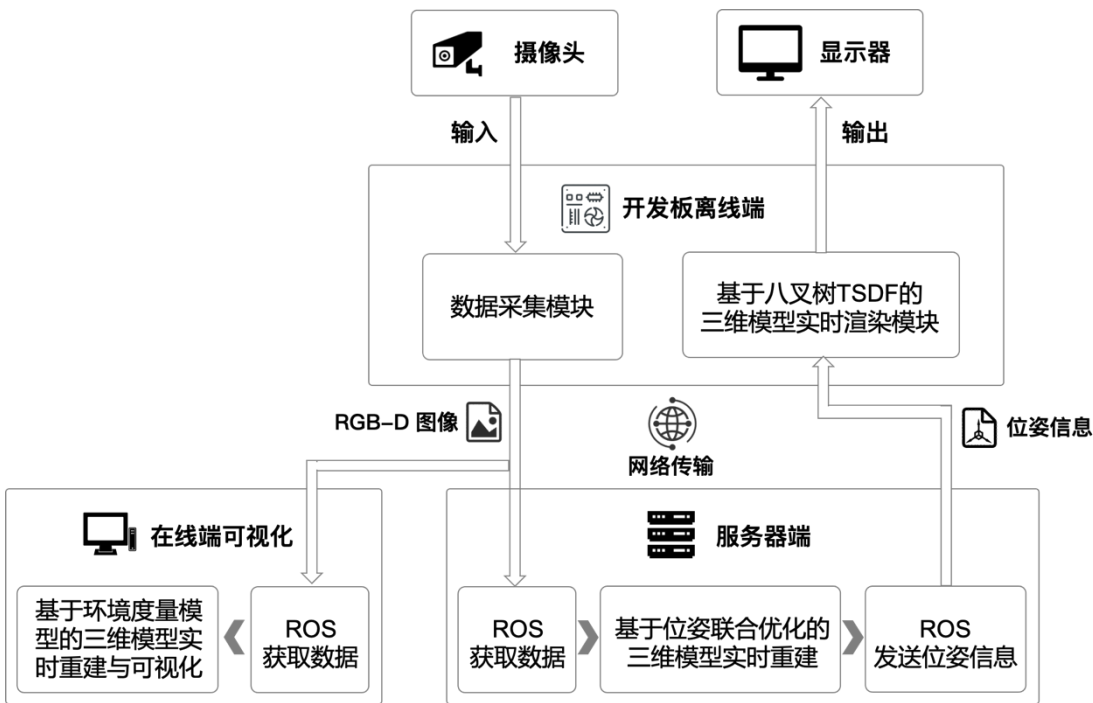


图 2-1 系统结构图

本项目的室内环境实时重建系统结构如图 2-1 所示，图中包含了两套实时重建系统，由离线端、在线端以及服务器端三部分组成。

上述三个组件中，离线端和服务器端可以单独构成一个离线室内环境实时重建系统，其中“离线”一词是指在离线端完成数据采集与可视化交互；离线端也可和在线端构成一个在线室内环境实时重建系统，其中“在线”一词是指由在线端提供可视化的交互。以下是两套实时重建系统的具体介绍：

面向嵌入式应用的离线室内环境实时重建系统

为了使得用户可以只在离线端进行操作就能进行室内环境的实时重建，并且充分利用开发板的算力，我们设计了离线室内环境实时重建系统。用户只需手持连接 RGB-D 摄像头的离线端对室内环境进行重建，就可以通过离线端所连接的显示器实时观察到重建的三维模型，通过网络连接至服务器端意味着对服务器的物理位置没有额外要求。这对于嵌入式应用开发具有重要意义，在搭载开发板的机器人或其他嵌入式应用场景中，开发板只需要获取数据，将消耗大量算力的三维重建交由服

务器进行计算，而开发板利用有限的算力进行模型的渲染和可视化，确保了良好的交互体验。以下是具体步骤：

首先由开发板离线端通过摄像头采集 RGB-D 数据，使用 ROS 通过网络传输发送至服务器端；服务器端获取 RGB-D 数据后，通过基于位姿联合优化的三维模型实时重建，得到优化后的位姿，并将其通过 ROS 发送至离线端；开发板离线端根据获取的位姿信息，对当前帧进行使用基于八叉树 TSDF 的三维模型实时融合渲染算法，对当前重建的模型进行可视化。

在线室内环境实时重建系统

考虑到开发板算力有限，我们将其作为数据采集装置，设计了在线室内环境实时重建系统。用户可以使用离线端进行数据采集，实时在在线端看到重建效果。以下是具体步骤：

首先由开发板离线端通过摄像头采集 RGB-D 数据，使用 ROS 通过网络传输发送至服在线端；在线端获取 RGB-D 数据后，通过基于环视度量模型的三维模型实时重建与可视化，实时展示重建的三维模型，用户可以在在线端进行交互。

2.2 部署环境

硬件构成

本项目的硬件构成如图 2-2 所示。左图为 Astra Pro 相机，可以同时采集彩色图像与深度图像；中图为 Zora P1 开发板，主要负责连接 Astra Pro，采集 RGB-D 图像以及在离线系统中对模型进行融合与可视化；右图为提供算力的工作站，主要用于为三维重建任务。



图 2-2 系统硬件构成

项目中的三个不同模块分别部署在不同的硬件设备上：

系统组件	配置	操作系统	外设
离线端	开发板 ZORA P1 Amlogic A311D 四核 A73+双核 A53 @2.8GHz 内存 2GB DDR4,	Armbian 18.04	Astra Pro RGB-D 相机，用于可视化交互的显示器、鼠标
服务器端	Nvidia GeForce GTX Titan Xp GPU	Ubuntu 16.04	无
在线端	2 * Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz	Ubuntu 16.04	用于可视化交互的显示器、鼠标

环境依赖

- 离线端：ROS Kinetic, OpenCV 3.4 + GTK2.0, Boost, Eigen3, Doxygen;
- 服务器端：ROS Melodic, CUDA 11, Boost, OpenCV, GLEW, Eigen3, OpenGL, X11;
- 在线端：ROS Melodic, PCL 1.8, octomap 1.6, G2O, X11, PkgConfig, Qt5, OpenMP, Eigen3, OpenCV。

代码说明

本代码共包含 4 份源码，皆为 ros package:

- bundlefusion_ros: bundlefusion 在 ros 上的移植，为本项目的服务器端
- fastfusion_ros: fastfusion 在 ros 上的移植，为本项目的在线端
- rgbdslam: rgbdslam-v2, 为本项目的离线重建与渲染端
- topic_demo: 自定义 ros 消息类型，包含相机位姿与 RGB 图像和深度图像

具体编译运行步骤见代码 README 文件。

3 技术优势分析

3.1 关键技术创新点

创新点 1：基于八叉树表示的 CPU 实时 TSDF 融合与渲染

本项目中，我们在客户端即开发板 ZORA P1，采用了一种八叉树的数据结构对体素化表示的三维模型 TSDF（Truncated Signed Distance Function）进行表示，并设计了加速的数据融合算法与网格模型渲染算法，最终能够在 CPU 上达到超过 45Hz 的实时模型融合和每秒一帧的三维网格模型渲染，有利于在嵌入式设备等无显卡加速的情景下实时展示三维模型。

将三维空间建模为许多小方块是一种常见的做法，八叉树地图的思想是将一个大方块的每个平面均匀切成两片，从而将其划分为八个同样大小的方块，这个步骤可以看成从一个节点展开成八个，那么整个从最大空间细分到最小空间的过程，就是一棵八叉树，如下图 3-1 所示。只有接近重建物体表面的节点会展开并记录 TSDF 值，而离表面较远的节点没有必要展开，因此节省了大量空间并提升了效率。在本项目中，每个节点包含 8 个 brick，每个 brick 如图 3-2 所示，其中包含 8^3 个体素，便于处理与后续渲染。

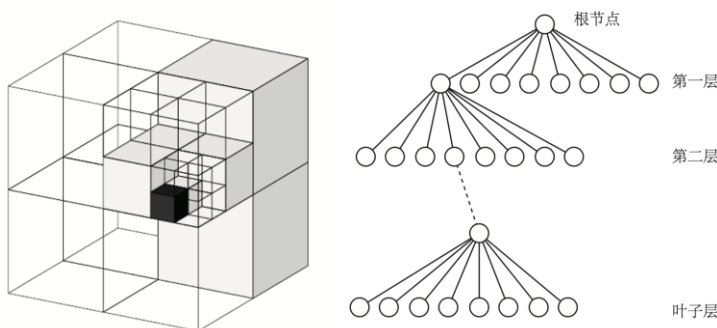


图 3-1 八叉树示意图

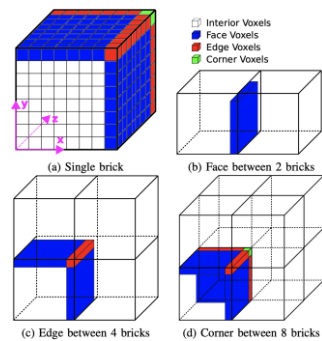


图 3-2 brick 示意图

基于上述的八叉树结构，我们采用的加速的数据融合算法步骤为：首先计算当前帧所有像素的 3D 点，找到相应的 brick，将所有点对应的 brick 放入队列中；对这个队列中每个 brick 中的 TSDF 值进行更新。在渲染步骤中我们采用了改进的 Marching Cubes 算法并进行增量式渲染的优化得到三维网格模型，具体细节见算法设计中。

创新点 2：基于环境度量模型的位姿估计算法

在传统的运动估计中，判断估计的位姿是否可靠通常使用判断内点的个数，当内点个数少于指定阈值时就认为估计的位姿不可靠，而进行剔除。但是，当出现运动模糊或者拍摄到纹理信息较少的区域时都容易出现内点较少的情况。且可能出现这样的情况，有一些点在一帧中可以看到，另一帧可能就被其他点挡住了，从而错误地认为相机位姿估计不可靠。本项目中使用的环境度量模型，实现了对估计的位姿可以实现更鲁棒的错误位姿剔除。可以证明给出两点 y_i 、 y_j ，它们之间的差满足高斯分布，方差为表示噪音的协方差矩阵[1]。

创新点 3：层次式全局联合位姿优化

本项目中使用了层次式的全局联合位姿优化，实现了实时的鲁棒位姿估计，避免了局部相机位姿估计误差的累积。我们将连续输入的 RGB-D 图像划分为单独的片段，首先对片段进行局部的位姿估计。在得到局部位姿的估计结果后，选取片段内的第一帧图像作为该片段的关键帧，再以片段为单位，对所有片段关键帧进行全局的位姿优化。层次式全局联合位姿优化的过程如图 3-3 所示。该方法弱化了连续输入图像之间的时序约束，将图像的位姿估计视为一个独立的全局轨迹的联合优化过程，而非仅仅针对相邻的图像进行局部的位姿估计。

如此我们能够在进行位姿估计的同时，完成全局的回环检测，有效地避免误差的累积，提升重建的稳定性。同时由于去除了部分时序跟踪的依赖，不会出现在传统连续相机位姿追踪时的跟踪丢失问题，新输入的图像能够随时匹配结合至现有的重建结果中。层次式的处理方案能够减少优化求解时的未知量数量，保证了算法的运行速度，让算法能够实时运行。

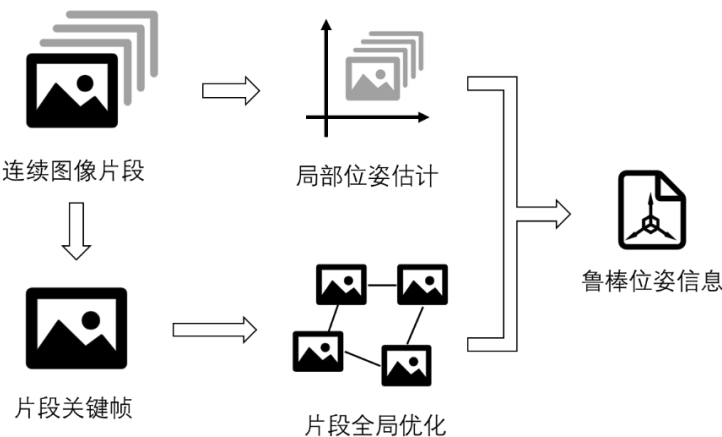


图 3-3 层次式位姿优化示意图

创新点 4：基于 ROS 通信的离线与在线离线室内环境实时重建系统

本项目考虑了两种具体的应用场景，设计了离线与在线的两套室内环境实时重建系统。其中离线系统主要考虑嵌入式的应用场景，用户可以只在离线端进行扫描操作就能进行室内环境的实时重建，并实时观察到重建效果，嵌入式设备有限的算力被充分利用以进行实时三维模型渲染；而在线系统则是在有好的算力的主机的情况下，在离线端进行扫描，在在线端完成与重建三维模型的实时交互。

以上系统在技术实现上依赖于 ROS 在多机系统上的通信，只保证不同的设备之间的网络连接，即可完成系统不同组件的数据传输。我们在获取 RGB-D 图像时尽可能保证 RGB 图像和深度图像的同步性，在离线系统的位姿传输环节设计了专门的位姿消息类型，保障了数据的精准高效传输。

3.2 核心算法设计

核心算法 1：离线端 CPU 实时 TSDF 融合与渲染

八叉树主要数据结构：本项目的八叉树数据结构如图 3-4 所示，几何结构信息被存储在 Brick 类中，每个 Brick 对象中包含 83 个体素，不同层级的节点对应的体素对应的真实尺寸是不同的；每个体素中存储 TSDF 值，权重以及体素的颜色，因此最后可视化的三维模型是带有颜色信息的；渲染主要用到的数据结构是 MeshCell 类与 Mesh 类，负责三维网格的渲染可视化。

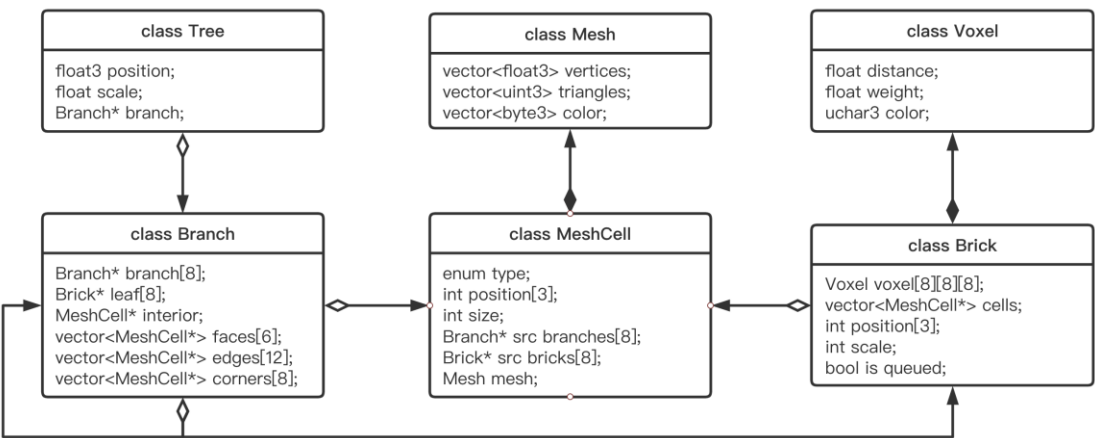


图 3-4 八叉树类图

基于八叉树的多分辨率数据融合：数据融合是指给定新的一帧深度图以及对应的位姿，将其融合进已有的模型。基于上述的八叉树数据结构，数据融合主要分为两个步骤：将当前帧所有像素的 3D 点对应的 brick 放入队列中；对这个队列中每个 brick 中的 TSDF 值进行更新。

首先，遍历深度图中的所有点，在八叉树中找到对应 brick。本项目采用小孔成像模型，假定内参 f_x, f_y, c_x, c_y 已知，当前帧的深度图为 z_t ，当前帧相对世界坐标的位姿为 R_t 和 T_t ，对于深度图中任意一个像素 $(x, y)^T$ ，其世界坐标为：

$$p_w = R_t \begin{pmatrix} (x - c_x) \frac{z_t(x, y)}{f_x} \\ (y - c_y) \frac{z_t(x, y)}{f_y} \\ z_t(x, y) \end{pmatrix} + T_t$$

根据 z_t 的值，我们为该点分配 brick 粒度 $s_1 := \exp_2[\log_2 \max\{z_t, 1\}]$ 。给定一个像素的 3D 世界坐标和粒度，我们在树中查找相应的砖块并将其放入队列中。我们要对它上层所有粒度更粗的 brick 更新。每个体素的更新权重取决于到表面的距离和 brick 粒度。此外，在我们的实现中，TSDF 截断带的大小随 brick 粒度而增长，因此更新的截断带始终包含相同数量的体素，而与尺度无关。对于包含 n 个 brick 的树，此查找步骤的复杂度为 $O(\log n)$ ，此步骤的成本很高，因此最小化树中的查找数会使得性能显著提高。因此，事先检查两个或两个以上的点位于同一块砖中可以防止我们执行不必要的树遍历。本项目采用一种快速的启发式方法，检查一个点是否与图像中其左邻点或上邻点位于同一块砖中，并且仅在没有的情况下遍历树。

在更新八叉树 brick 步骤，本项目在串行更新队列中的 brick 的基础上，做出两点优化：1) 利用迭代的串行顺序来重用预先计算的值；2) 减少代码中的 if 语句以启用单指令多数据流计算。具体更新步骤见[2]中。

实时三维网格渲染：本项目采用 Marching Cubes[3]算法进行 TSDF 中的网格提取。尽管该算法易于在规律网格上应用，但是由于要在边界上查找相邻体素，在 brick 上较难应用，因此 FastFusion[2]设计了 4 种嵌套的递归算法：

- 如果分支包含一个子分支，则计算分支内部网格的算法将递归到子分支中，并生成面，边和中角的网格；
- 如果与一个面相关的 2 个分支中的任何一个分支都包含一个子分支，则计算该面网格的算法将递归为 4 个子面，并生成 4 个边和它们之间的中间角的计算；
- 如果与边缘关联的 4 个分支中的任何一个包含一个子分支，则计算边缘网格的算法将递归到 2 个子边缘中，并生成它们之间的中间角的计算；

- 如果与角相关联的 8 个分支中的任何一个包含子分支，则算法将递归为较小的大小，并将现有的子分支传递给递归调用。

Marching Cubes 比数据融合需要更多的时间，因为它必须考虑体素的邻域关系。因此，本项目在与数据融合平行的第二个线程中运行网格渲染。每当数据融合线程更新 **brick** 时，它将受影响的网格单元加入队列。我们使用二进制标志来指示已将网格单元添加到队列中，以防止两次添加相同的单元。处理完网格单元后，将其“排队”标志复位并将其从队列中删除。通过这种方式，网格划分的运行时复杂度从线性时间 $O(n)$ 减少为线性时间在队列中的 **brick** 数。

核心算法 2：在线端位姿估计及重建

在本项目在线端可视化基于 RGBDSLAM-V2[4]实现，其基本流程图如图 3-5 所示。Zora P1 开发板通过 Astra Pro 同时采集彩色图像与对应的深度图像。并将图像发送到算力资源丰富的服务端，进行在线端的重建，并在在线端对重建的结果进行可视化展示。

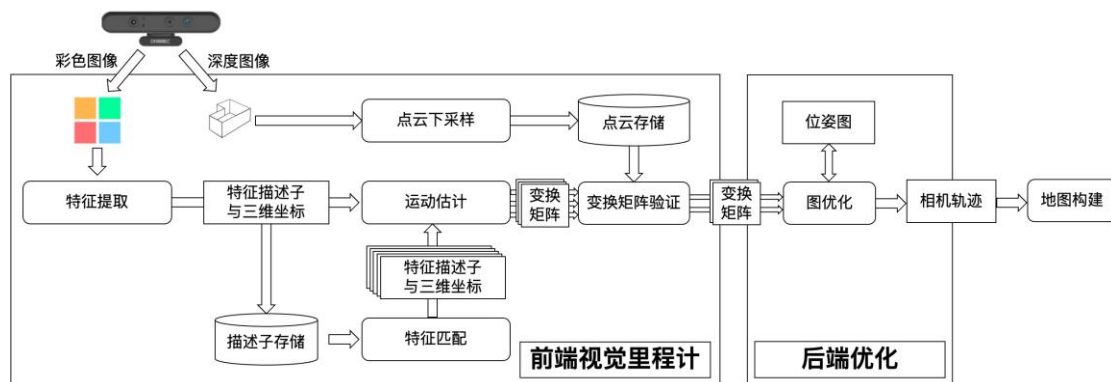


图 3-5 在线端流程图

特征提取：在本系统中可选的特征包含 SIFT,SURF 和 ORB 特征，为了保证计算的实时性，使用 SIFT 特征时需要 GPU 加速计算，SURF 特征和 ORB 特征都由 CPU 版本的 OpenCV 实现。

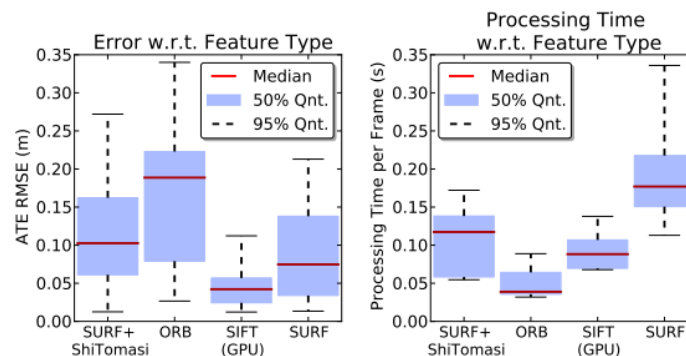


图 3-6 不同特征错误率与运行时间比较

如图 3-6 所示，有 GPU 时，SIFT 特征表现最好。综合考虑算力和准确率则 ORB 特征更具优势。

运动估计：本系统使用 RANSAC 方法计算相邻帧的映射矩阵。即使用三对特征点快速计算 RANSAC 的初值。接着迭代求解映射矩阵的最优值，每一轮迭代优化对应点之间的马氏距离。马氏距离相比欧氏距离考虑到了距离的各向异性，可以得到更好地优化结果。

环境度量模型对运动估计进行筛选：为了更可靠的对运动估计结果进行筛选，我们使用了环境度量模型(EMM)。如图 3-7 所示，Camera A 与 Camera B 是相邻两帧的相机，将 Camera A 观测到的点投影到 Camera B，在 Camera B 观测到的点中找到内点、外点与被遮挡的点。Camera A 中的 y_i 与 Camera B 中的 y_j 为同一个点，即内点。 y_k 不在 Camera A 的视场范围内，被忽视。投影后的 y_q 被 y_k 遮挡，即归为被遮挡的点。至于 y_p 没有与之对应的点，被归为外点。因此整个观测结果为两个内点，一个外点和一个被遮挡的点。通过将点进行分类考虑，模型对相机位姿估计的判断具有更强的鲁棒性。

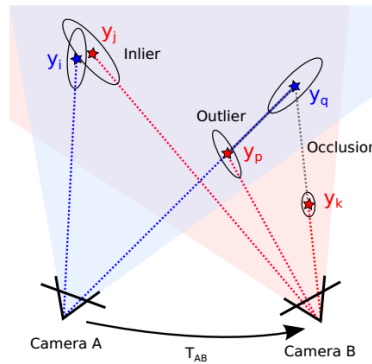


图 3-7 EMM 计算方法示意图

回环检测：本项目基于最小生成树和随机森林进行回环检测。通过计算特征描述子之间的距离，生成一颗以前一帧为根结点的有限深度的最小生成树。接着按照时间顺序，删除 n 张前置帧以避免重复。然后从最小生成树中随机抽取 k 帧（偏向于抽取较早的帧）来寻找闭环。

图优化：没有被 EMM 拒绝的 motion，将两帧相机位姿作为优化顶点，motion 作为约束，加入优化边。检测到的回环，也加入优化顶点和边。接着使用图优化库 G2O 来优化边，误差函数为：

$$F(X) = \sum_{\langle i,j \rangle \in E} e(X_i, X_j, Z_{ij})^T \Omega_{ij} e(X_i, X_j, Z_{ij})$$

这里的 x_i, x_j 为优化变量（顶点位姿的估计）， Z_{ij} 是约束，也就是 x_i 和 x_j 之间的变换。 E 是度量 x_i, x_j 对 Z_{ij} 约束下的误差。中间的 Ω 是约束（优化边）的信息矩阵（协方差矩阵的逆），表示对边的精度的估计。

三维重建：本项目使用八叉树进行三维重建。按照立方体的结构，将大立方体等分成8个小立方体，每个小立方体对应八叉树的一个节点，并对需要细分的小立方体进行进一步细分，以构建八叉树地图。每个小立方体存储被占据的概率的logist回归，被观测到占据次数越多，概率越大，而如果没有一次观测到被占据，则不用展开该节点。使用八叉树建图的优点是利于导航，易于更新且存储方式比较省空间。

核心算法 3：服务端全局位姿估计与重建

本项目中的服务器端基于 BundleFusion[5]实现实时的位姿估计与优化。其中的核心算法如下：

特征匹配与筛选：本算法中首先针对图像提取特征点，并对图像间的特征点进行匹配，以用于后续的位姿估计。由于特征点的提取与匹配是后续重建工作的基础，为了在特征点处理阶段得到鲁棒的匹配结果，算法对搜索匹配得到的特征点对进行了筛选。

当新的图像输入，首先提取 SIFT（Scale-Invariant-Feature-Transform）特征点，SIFT 特征点对手持 RGB-D 扫描时的图像旋转、缩放等情况较为鲁棒。提取得到的特征点将与其它所有的图像特征点进行搜索匹配，此时得到的匹配结果仅为候选匹配，剔除错误匹配的特征点对以及误差较大的点对，得到有效匹配。

对于一对输入图像 f_i 及 f_j ，对应提取得到的特征点为 P 和 Q ，需要从从中筛选出分布一致且稳定的匹配特征点对。对于当前的特征点对集合，使用 Kabsch 算法[6]计算其对应图像间的变换矩阵 T_{ij} ，根据变换矩阵计算匹配点对的重投影误差，若重投影误差较大（ $> 0.02m$ ），则将该匹配点对视为无效匹配，移出集合。对集合中的每一对候选匹配都进行如上筛选，直到所有无效点对被剔除，若当前图像的所有对应候选匹配均被剔除，则对应两帧图像中的特征点将不会参与后续优化。

另外需要考察匹配的特征点对所覆盖的图像区域是否足够大，即特征点对在图像内是否分布地较为均匀，若特征点均集中在图像的某一区域，容易导致位姿估计出现误差。将候选匹配集合 P 和 Q 中的特征点投影至一个平面中，计算所有特征点对应的最小外接矩形的面积，若该矩形的面积较小（ $< 0.032m^2$ ），则将当前候选匹配集合丢弃。

最后需要对匹配的特征点对进行几何及光度上的稠密验证。使用先前计算得到的变换矩阵 T_{ij} ，计算对应匹配特征点之间的深度值差异、法向量差异以及光度误差。为了减少匹配稠密验证时的运算量，这一步骤中首先将输入图像下采样至 80×60 的尺寸，再进行稠密验证。只有当对应特征点对满足以下条件时，才会被视为有效匹配：

$$\|T_{ij}(d_{i,x,y}) - d_{j,x,y}\|_2 < \tau_d$$

$$(T_{ij}(n_{i,x,y})) \cdot n_{j,x,y} < \tau_n$$

$$\|T_{ij}(c_{i,x,y}) - c_{j,x,y}\|_1 < \tau_c$$

其中第一个条件表示对应特征点对间的深度误差， $d_{i,x,y}$ 为图像 f_i 在坐标位置 (x,y) 的深度值；第二个条件为对应特征点法向量的余弦相似度， $n_{i,x,y}$ 为图像 f_i 在坐标位置 (x,y) 的法向量；第三个条件为对应特征点对的光度误差， $c_{i,x,y}$ 为图像 f_i 在坐标位置 (x,y) 的光度值。此处选取 $\tau_d = 0.15m$, $\tau_n = 0.9$, $\tau_c = 0.1$ 。

候选匹配中的特征点对需要通过以上所有的筛选条件才会被列为有效匹配，用于后续的位姿优化中。同时对于一对输入图像，其筛选后的有效匹配数量需要大于 N_{min} 才会用于后续的重建，虽然理论上存在3对有效匹配的特征点对就能够完成位姿估计，但此处选择 $N_{min} = 5$ ，以保证算法的准确度。

层次式位姿优化：为了保证算法能够以实时速度完成大量图像的位姿优化，算法使用了层次式的全局位姿优化。输入的图像序列被划分为一个个独立的片段，片段内包含少数的连续图像，首先针对这些连续图像进行局部的位姿估计。随后将从片段中选择相应的关键帧，并以片段为单位，对所有历史片段进行全局的位姿优化。

此处选取连续的11帧图像作为一个片段，片段之间则保留1帧图像的重叠。对于片段内的连续图像，首先对所有图像进行特征提取及匹配，并经过前述特征筛选得到的有效匹配特征点集。利用有限匹配的特征点集，进行局部的位姿估计，即计算每一帧图像相对于当前片段第一帧图像的变换矩阵。由于片段中仅包含了少量的图像，相机位姿的变化程度较小，图像间的变换能够直接以单位矩阵作为初始化进行优化。在完成局部位姿估计后，使用前述的稠密验证方法检验位姿估计的一致性，若当前重投影误差较大（ $> 0.05m$ ），则将该片段将不会参与后续的全局位姿优化。

完成片段内的局部位姿优化后，将片段内的第一帧图像作为该片段的关键帧，并根据片段内的局部位姿估计结果，将片段内的所有特征点整合得到该片段对应的特征点集。片段内所有图像的有效特征匹配可能包含对世界坐标下同一特征点的重复匹配，因此将世界坐标下较为接近的（ $< 0.03m$ ）的不同特征点进行融合。所有

特征点将会被投影至片段的关键帧中，后续的全局位姿优化将针对片段关键帧及对应的关键特征点集进行。

对于所有的关键帧及对应关键特征点集，首先会重复特征匹配及特征筛选的过程，让关键帧在所有历史关键帧中进行特征匹配。再针对关键帧之间的特征匹配结果，进行全局的位姿优化。位姿优化的能量函数与片段内局部优化时相同。在得到关键帧的全局位姿后，将关键帧的全局位姿与片段内图像对应的局部位姿进行叠加，即可得到每一帧图像的全局位姿。

位姿优化与能量函数：层次式的位姿优化需要在局部和全局两个层次进行位姿优化，而两次优化均采用相同的优化策略。输入图像集合为 S ，根据进行位姿优化的层次不同， S 可能是单个图像片段内的连续图像，或者是全局所有的片段关键帧。虽然不同层次时的输入图像集合不同，但是优化的目标都是得到集合内的每一帧图像对应的位姿变换矩阵 T_i ，每个矩阵中包含了 3 个旋转变量及 3 个平移变量。将所有的目标变量记为：

$$X = (R_0, t_0, \dots, R_{|S|}, t_{|S|})^T = (x_0, \dots, x_N)^T$$

其中 N 为未知量总数。位姿的求解则是求解关于 X 的非线性优化问题，而最小化目标则是基于稀疏特征匹配以及稠密光度几何误差的能量函数：

$$E_{align}(X) = w_{sparse}E_{sparse}(X) + w_{dense}E_{dense}(X)$$

其中 w_{sparse} 以及 w_{dense} 分别为稀疏优化项及稠密优化项的权重，并且 w_{sparse} 会随着迭代次数逐渐降低，目的是在迭代初期通过稀疏特征匹配得到优化目标的初值，后续迭代中通过稠密匹配来逼近最优值。

稀疏匹配项 $E_{sparse}(X)$ 使用的是集合内所有匹配特征点对之间的距离之和：

$$E_{sparse}(X) = \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} \sum_{(k,l) \in C(i,j)} \|T_i p_{i,k} - T_j p_{j,l}\|_2^2$$

$p_{i,k}$ 表示第 i 帧图像中的第 k 个特征点， $C(i,j)$ 则是第 i 帧图像与第 j 帧图像之间有效匹配的特征点对的集合。稀疏匹配项是通过最小化匹配特征点对的几何距离来优化目标位姿。

稠密匹配项 $E_{dense}(X)$ 则是基于针对图像像素的几何及光度进行约束：

$$E_{dense} = w_{photo}E_{photot}(X) + w_{geo}E_{geo}(X)$$

$E_{photot}(X)$ 与 $E_{geo}(X)$ 分别代表光度误差及几何误差。光度误差项的目的在于最小化彩色图像之间对应点的光度误差：

$$E_{photo}(X) = \sum_{(i,j) \in E} \sum_{k=0}^{|I_i|} \left\| I_i \left(\pi(d_{i,k}) \right) - I_j \left(\pi(T_j^{-1} T_i d_{i,k}) \right) \right\|_2^2$$

I_i 表示第 i 帧图像的像素梯度， π 为透视投影， $d_{i,k}$ 则是第 i 帧图像中的第 k 个像素所对应的三维坐标。几何误差则是：

$$E_{geo}(X) = \sum_{(i,j) \in E} \sum_{k=0}^{|D_i|} \left[n_{i,k}^T \left(d_{i,k} - T_i^{-1} T_j \pi^{-1} \left(D_j \left(\pi(T_j^{-1} T_i d_{i,k}) \right) \right) \right) \right]^2$$

稀疏匹配项针对特征点的有效匹配进行优化，而稠密匹配优化项是针对图像中的所有像素。为了减少计算量，与特征匹配筛选阶段的稠密验证类似，稠密匹配优化是在尺寸为 80×60 的下采样图像上完成。

为了求解以上的优化问题，使用了基于高斯牛顿法的 **GPU** 并行求解方案。通过利用优化问题中的稀疏特性，并借助现有的 **GPU** 并行求解策略[7]，对求解过程进行了优化加速，实现了全局位姿优化的实时求解。

4 成果展示

4.1 系统重建效果展示

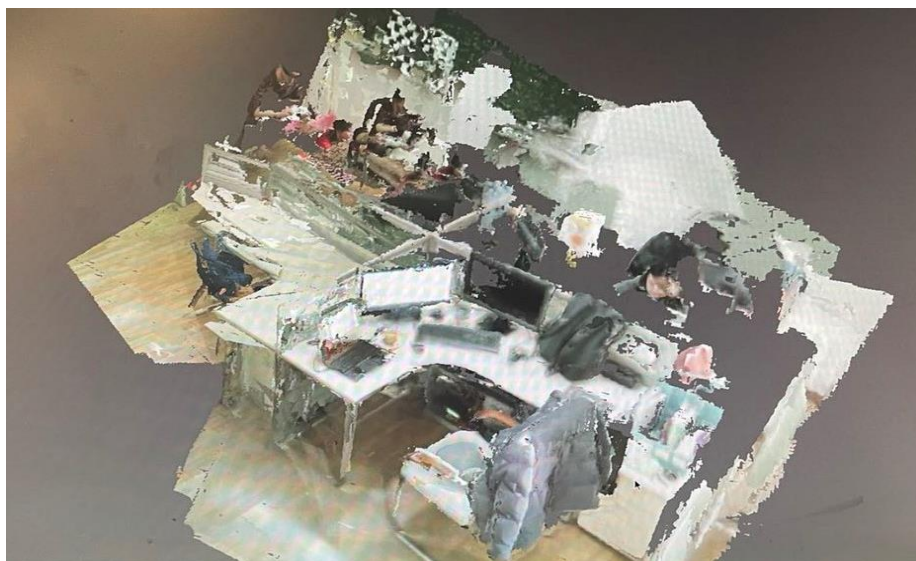
我们尝试使用本系统对面积约为 10 平方米的室内区域进行重建。重建的目标场景中包含了较为复杂且包含更多细节内容的桌面，以及较为空旷的桌面，如图 4-1 所示。



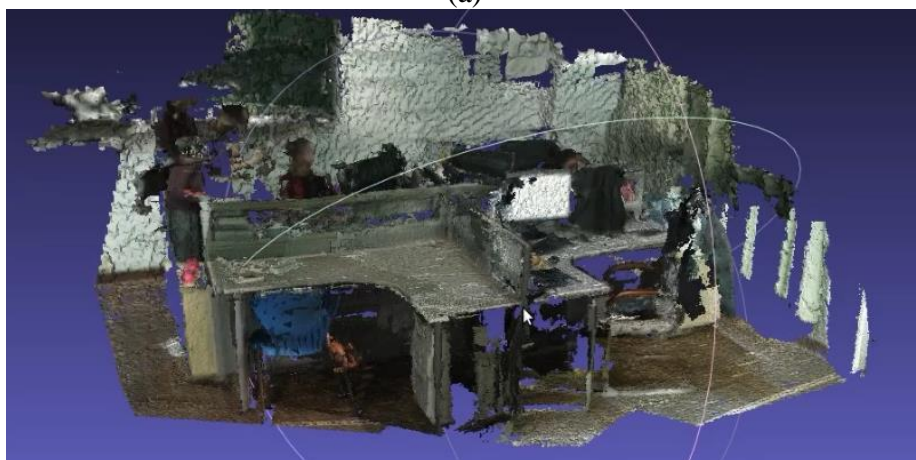
图 4-1 实验环境

离线室内环境实时重建效果

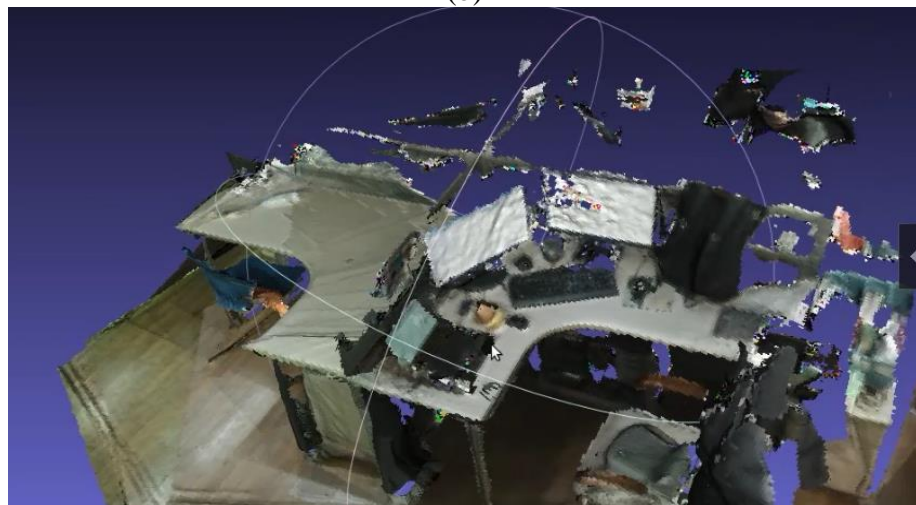
图 4-2 为本项目中的离线系统重建的效果，图 4-2（a）为开发板上实时渲染的模型截图，在整个手持重建过程中，可以流畅地渲染出当前重建的进度以及效果；图 4-2（b）为开发板融合形成的模型效果展示；图 4-2（c）为服务器优化整合后的最终模型，可以看出两张桌子的结构以及细节得到很好的重现。



(a)



(b)



(c)

图 4-2 离线重建结果

在线室内环境实时重建效果

图 4-3 为本项目中的在线系统重建的效果，上图和下图分别为两次重建实验的结果，上图重建了 3 个工位，下图重建了 4 个工位，在整个手持重建过程中，在线端可以流畅地渲染出当前重建的进度以及效果，并且显示相机拍摄的轨迹。可以看出相机的位姿估计较为准确地还原了拍摄时的运动轨迹，桌子相对位置与结构以及桌面物体细节得到很好的重现。

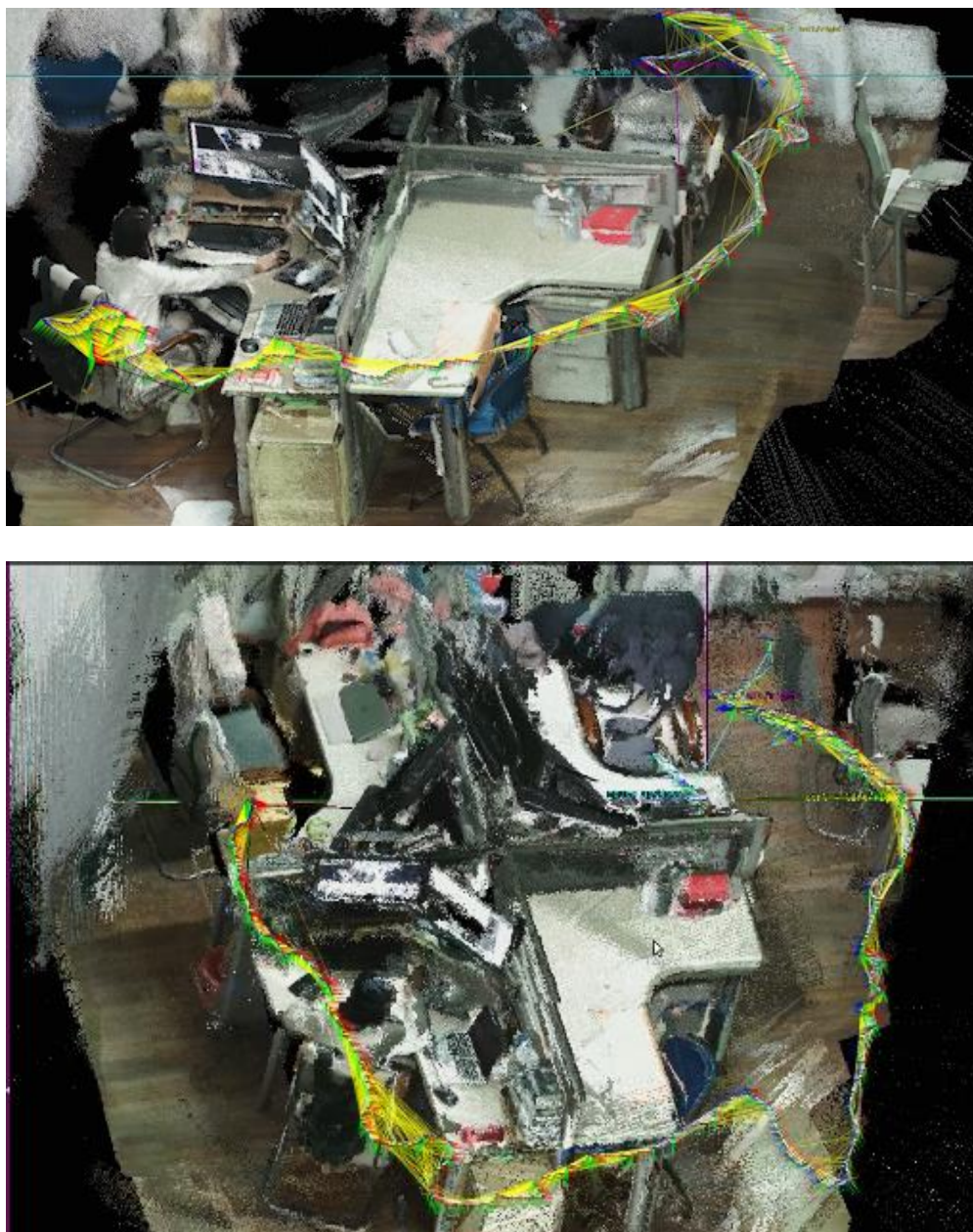


图 4-3 在线重建结果

4.2 系统性能量化测试

为了量化测试整个系统的性能，我们进行了量化的运行速率测试。为了确定具体的性能影响因素便于确定系统瓶颈，我们对比了每个环节独自在现有数据集上运行与在系统中实时重建的速率，结果如下表所示：

系统组件	任务	速率(fps)
开发板离线端	在本机数据集上进行融合	11.6
	在本机数据集上进行渲染	2.3
	离线实时重建读取 ROS 数据融合	8.6
	离线实时重建读取 ROS 数据渲染	2.8
GPU 服务器端	在本机数据集上进行重建	2.6
	离线实时重建读取 ROS 数据重建	8.6
CPU 在线端	在本机数据集上进行重建渲染	7.6
	在线实时重建读取 ROS 数据重建渲染	8.4

开发板离线端在数据集上进行融合的速率达 11.6fps，然而在本项目的离线系统中融合速率与 GPU 服务器端重建速率保持一致，说明开发板离线端融合速率主要受限于 GPU 服务器端的重建速率。

由上表可见，本项目的离线系统与在线系统均能达到实时重建并可视化的效果，保障了良好的交互体验。

5 研发过程记录

5.1 内参标定

我们使用针孔相机模型来对相机拍摄到的图像进行建模，因此为了确定空间物体中的三维几何位置与图像上对应像素的关系，需要获取相机的内参。

而本项目使用的 Astra Pro 相机自身未包含内参的数据，因此需要自行对内参进行标定。由于 RGB-D 传感器包含用于测量深度的红外相机以及拍摄彩色图像的 RGB 相机，因此需要对两个相机分别进行标定。本项目中使用的 Matlab 中的相机标定工具进行标定。

首先打印制作用于标定的标定板，本项目中使用的棋盘格标定板包含 8×6 的矩形方格，方格边长为 30mm。



图 5-1 标定板

分别使用彩色相机以及红外相机从不同的角度拍摄标定板。拍摄过程使用 Orbbec Viewer 完成。



图 5-2 彩色相机及红外相机拍摄标定板

使用 Matlab 中的标定工具进行标定。可以使用 Matlab App 中的 Camera Calibrator，此工具提供了相应的图形界面进行操作；或使用 Matlab 提供的函数通过代码实现。

Camera Calibrator 的使用：

- Add Images 导入拍摄的照片，本项目中分别使用了 15 张彩色图像及 15 张红外图像。图像导入的过程中会完成图像中标定板关键点的检测。
- Calibrate 进行标定。标定时计算得到的相机位姿以及标定的重投影误差会展示在窗口右侧。
- Export Camera Parameters 导出标定的内参。默认保存至变量 cameraParams 中，查看相应的变量即可得到相机的内参。

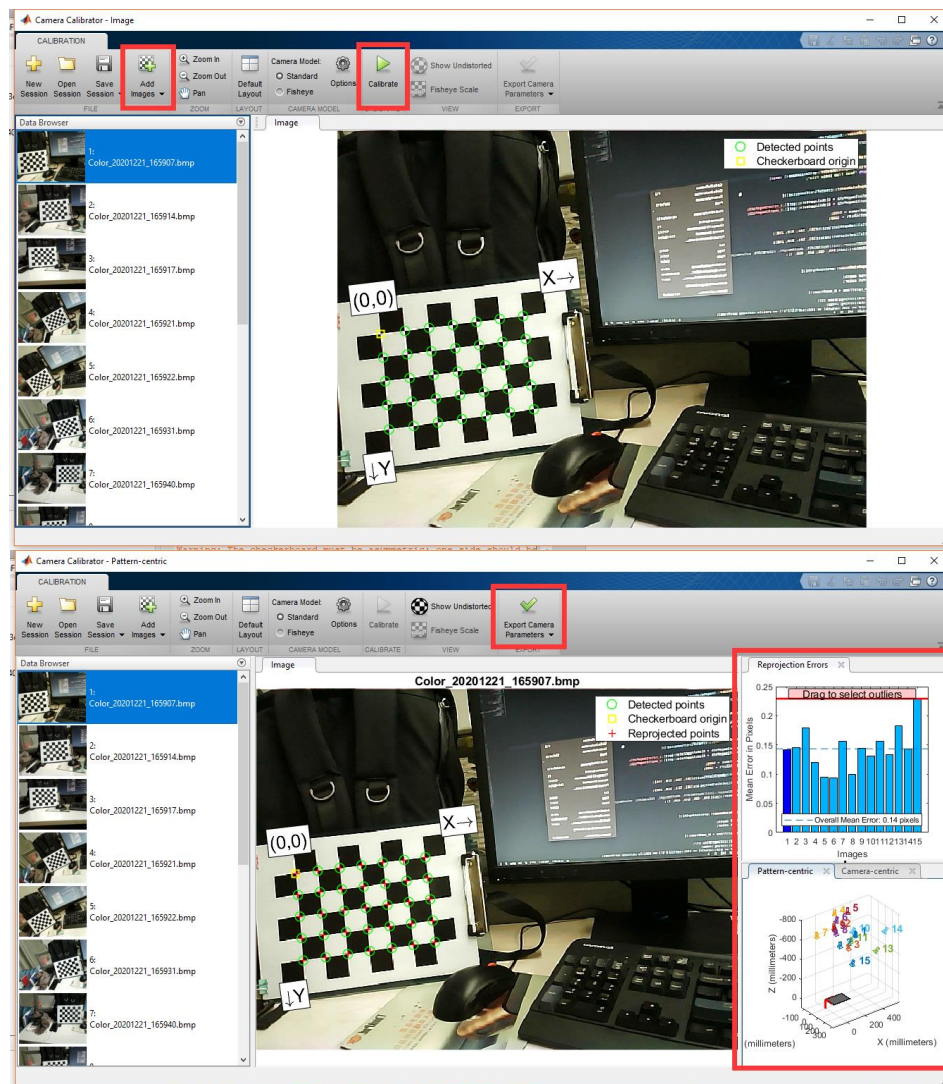
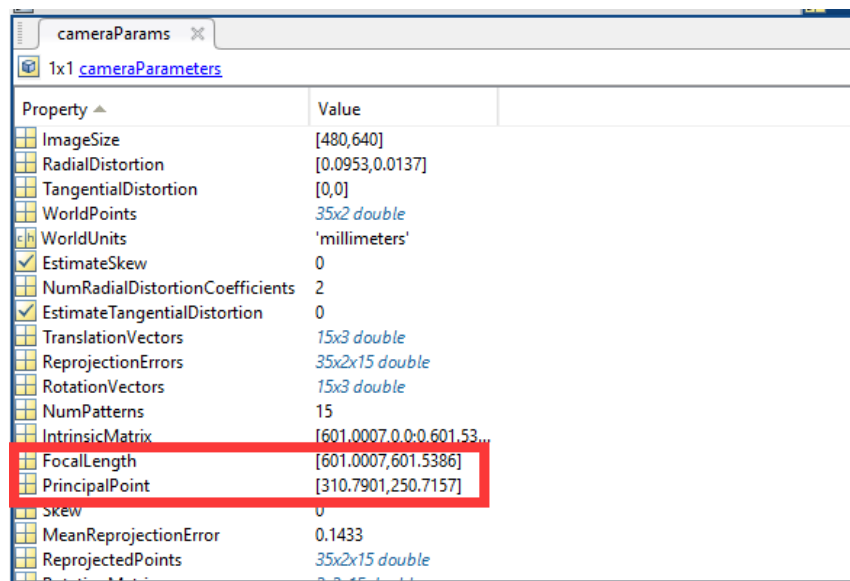


图 5-3 Camera Calibrator 操作界面



Property	Value
ImageSize	[480,640]
RadialDistortion	[0.0953,0.0137]
TangentialDistortion	[0,0]
WorldPoints	35x2 double
WorldUnits	'millimeters'
EstimateSkew	0
NumRadialDistortionCoefficients	2
EstimateTangentialDistortion	0
TranslationVectors	15x3 double
ReprojectionErrors	35x2x15 double
RotationVectors	15x3 double
NumPatterns	15
IntrinsicMatrix	[601.0007,0.0,0.601.53...
FocalLength	[601.0007,601.5386]
PrincipalPoint	[310.7901,250.7157]
Skew	0
MeanReprojectionError	0.1433
ReprojectedPoints	35x2x15 double

图 5-4 导出内参结果

通过调用 Matlab 相关函数完成相机标定：

```

1. % 读取待标定的图像序列
2. images = imageDatastore('color/');
3. imageFileNames = images.Files;
4.
5. % 检测标定板中的特征带
6. [imagePoints, boardSize] = detectCheckerboardPoints(imageFileNames);
7.
8. % 计算特征点的世界坐标
9. squareSize = 30; % 棋盘格尺寸，毫米
10. worldPoints = generateCheckerboardPoints(boardSize, squareSize);
11.
12. % 标定相机
13. I = readimage(images, 1);
14. imageSize = [size(I, 1), size(I, 2)];
15. [params, ~, estimationErrors] = estimateCameraParameters(imagePoints, worldPoints, 'ImageSize', imageSize);
16.
17. % 输出标定结果
18. displayErrors(estimationErrors, params);
19.
20. % 展示相机位姿
21. figure;
22. showExtrinsics(params, 'PatternCentric');
23.
24. % 展示重投影误差
25. figure;
26. showReprojectionErrors(params);

```

运行程序即可输出得到相机标定的内参，及相机位姿和重投影误差的示意图：

1. Intrinsics
2. -----
3. Focal length (pixels): [599.2689 +/- 1.8768 600.0419 +/- 1.8956]
4. Principal point (pixels): [309.0445 +/- 0.7184 249.9152 +/- 0.6680]
5. Radial distortion: [0.0833 +/- 0.0059 0.0466 +/- 0.0211]

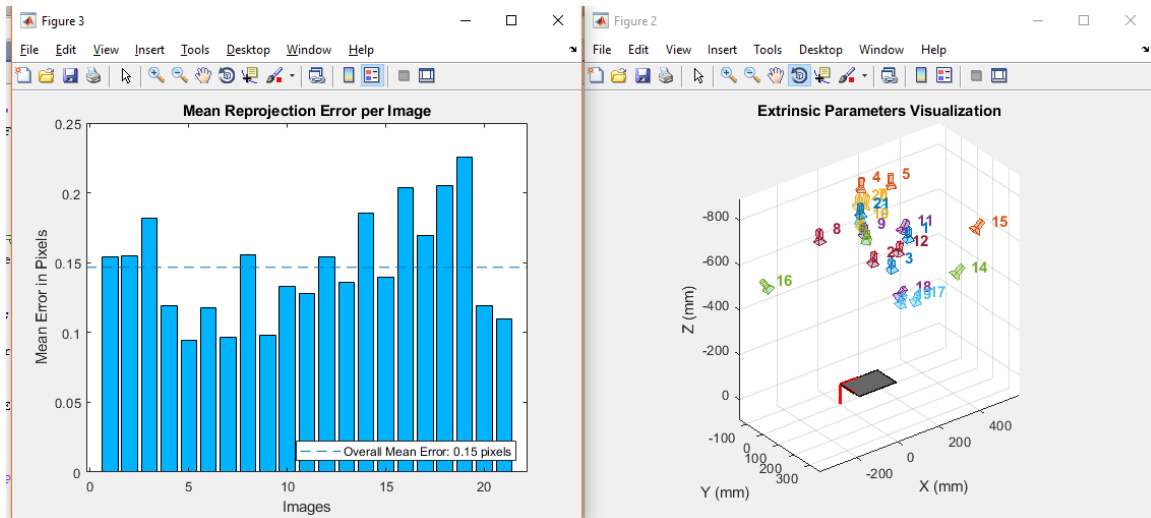


图 5-5 标定的相机位姿及重投影误差

5.2 相机配置

出于降低系统各模块之间耦合度的目的，本项目选择了使用以 `ros` 为平台搭建三维重建系统。相应的我们按照官方文档，使用 `ros_astra_camera` 来调用相机，并将 `Astra Pro` 拍摄到的图像发布到对应的 `ros topic` 下供各模块订阅调用。

其配置步骤如下：

1. 安装 `ros` 系统，具体可参考官方链接，注意需安装自己设备对应的版本：
<http://wiki.ros.org/ROS/Installation>
2. 安装相关依赖：

```
1. sudo apt install ros-$ROS_DISTRO-rgb-d-launch ros-$ROS_DISTRO-libuvc ros-$ROS_DISTRO-libuvc-camera ros-$ROS_DISTRO-libuvc-ros
```

3. 创建 `ROS Workspace`（如果已经有了则跳过这一步）：

```
1. mkdir -p ~/catkin_ws/src
2. cd ~/catkin_ws/
3. catkin_make
```

```
4. source devel/setup.bash
```

4. Clone ros_astra_camera 仓库到本地:

```
1. cd ~/catkin_ws/src
2. git clone https://github.com/orbbec/ros_astra_camera
```

5. 创建 astra udev rule:

```
1. roscd astra_camera
2. ./scripts/create_udev_rules
```

6. 编译 astra_camera:

```
1. cd ~/catkin_ws
2. catkin_make --pkg astra_camera
```

7. 运行 astra_camera (注意需要选择自己相机对应的 launch file):

```
1. roslaunch astra_camera astra.launch
```

运行后, 通过 `rostopic list` 可以看到很多信息, 如图像、相机位姿、点云已经被成功发布到对应的 topic 以供订阅。可通过 `rviz` 或 `image_viewer` 尝试读取相应图片。

为了进一步提升三维重建的可靠性, 需要使用相机内参。为此, 需要将标定好的内参按照指定格式放置在 `~/ros/camera_info` 文件夹中以供 `astra_camera` 读取并使用。彩色相机与深度相机的内参分别需要命名为 `camera.yaml` 和 `depth_Astra_Orbbec.yaml`。

5.3 ROS 网络通信

为了使各模块功能解耦合化, 我们使用 ROS 在各模块之间通信。本项目共用到了两台计算设备: 配备高性能显卡的主机与 Zora P1 开发板。为了使两台设备通过 ROS 通信, 需要通过无线局域网或以太网连接两台设备。以以太网连接为例, 具体步骤如下:

1. 使用网线连接两台设备的以太网口。
2. 设置主机 ip 地址
 - 2.1. 使用 `ifconfig` 查看当前网络连接状况

```
1. $ ifconfig
2. enp3s0f0 Link encap:Ethernet HWaddr **:**:**:**:**:**
3. ...
```

可以看到这里的名称是 `enp3s0f0`,

2.2. 设置静态 ip 地址

```
1. sudo ifconfig enp3s0f0 192.168.2.51
```

3. 设置开发板 ip 地址,与设置主机 IP 地址类似

3.1. 使用 `ifconfig` 查看当前网络连接状况

```
1. $ ifconfig
2. eth0 ...
```

可以看到这里的名称是 `eth0`,

3.2. 设置静态 ip 地址

```
1. sudo ifconfig eth0 192.168.2.53
```

需注意, 两台设备的 ip 地址需设置在同一个子网内

4. 两台设备之间尝试 ping 操作, 看看能不能 ping 通

```
1. $ ping 192.168.2.53
2. PING 192.168.2.53 (192.168.2.53) 56(84) bytes of data.
3. 64 bytes from 192.168.2.53: icmp_seq=1 ttl=64 time=0.908 ms
4. ...
```

5. 配置两台设备的 `~/.bashrc`

5.1. 在主机的 `~/.bashrc` 加入以下两行

```
1. export ROS_HOSTNAME=192.168.2.51
2. export ROS_MASTER_URI=http://192.168.2.53:11311
```

5.2. 在开发板的 `~/.bashrc` 加入以下两行

```
1. export ROS_HOSTNAME=192.168.2.53
2. export ROS_MASTER_URI=http://192.168.2.53:11311
```

经过上述步骤两台设备对 `ros` 节点就具有透明性了, 每一个节点无须关心自己在哪台机器上运行了(驱动节点必须运行在跟硬件设备有物理连接的机器上)。也就是说, 每个节点可以自由发布 `ros topic`, 每个节点也可以自由订阅另一台设备上发布的 `ros topic`。

在本项目中, 开发板直接通过 `Astra Pro` 采集彩色图和深度图, 并将图像和对应的相机参数发布到指定节点。通过建立局域网连接后主机上运行的服务器端可以直接订阅采集到的图像数据。服务器端采集到对应的数据后, 经过一系列计算, 得到

相机的位姿，并将相机位姿和对应的彩色图像和深度图像一并发布到指定 topic，开发板上运行的节点订阅该 topic 并进行实时的融合与渲染。该 topic 数据格式下：

```
1. //pose.msg
2. float32 translation1
3. float32 translation2 // 平移量
4. float32 translation3
5. float32 q1
6. float32 q2 // 旋转量（四元数表示）
7. float32 q3
8. float32 q4
9. sensor_msgs/Image colorImage // 彩色图像
10. sensor_msgs/Image depthImage // 深度图像
```

在~/catkin_ws/src 建立 topic_demo 包，并添加相关依赖，并编译以供需要该数据类型的节点使用。

服务器端对相机位姿完成估计后，发布到 pose_info 节点，相关代码摘录如下：

```
1. ros::NodeHandle nh;
2. ros::Publisher pub = nh.advertise<topic_demo::pose>("pose_info", 100);
3. topic_demo::pose msg;
4.
5. ...
6.
7. {
8.   msg.translation1 = translation_vector[0];
9.   msg.translation2 = translation_vector[1];
10.  msg.translation3 = translation_vector[2];
11.  msg.q1 = -q.coeffs().transpose()[0];
12.  msg.q2 = -q.coeffs().transpose()[1];
13.  msg.q3 = -q.coeffs().transpose()[2];
14.  msg.q4 = q.coeffs().transpose()[3];
15.  cv_bridge::CvImage(std_msgs::Header(), imageColor->encoding, pColorImage->image).toImageMsg(msg.colorImage);
16.  cv_bridge::CvImage(std_msgs::Header(), imageDepth->encoding, pDepthImage->image).toImageMsg(msg.depthImage);
17.  pub.publish(msg);
18. }
19.
20. ...
```

在开发板端，订阅 pose_info 节点，并根据计算得到的位姿和对应的彩色图和深度图，进行融合与渲染。相关代码如下：

```
1. ros::NodeHandle n;
2. ros::Subscriber sub = n.subscribe<topic_demo::pose>("pose_info", 1, boost::bind(&addMapCallback, _1, imageDepthScale, maxCamDistance, fusion, newMesh));
3. ros::spin();
4.
5. ...
6.
7. Void addMapCallback(const topic_demo::pose::ConstrPtr &msg,
```

```
8. float imageDepthScale,  
9. float maxCamDistance,  
10. FusionMipMapCPU *fusion,  
11. volatile bool *newMesh)  
12. {  
13. // 融合与渲染模型  
14. }
```

5.4 FastFusion 移植

我们主要采用 FastFusion 在开发板上进行实时的模型融合与渲染，主要根据[8]进行修改移植。

首先考虑到上述仓库是在普通笔记本电脑上进行开发运行，我们希望先在 PC 上运行调试，在给定数据集上运行成功后再移植到开发板按需求进行修改。由于代码版本较为久远，在其依赖库 OpenCV、OpenGL、QT 等都遇到一些问题，最终选择手动编译安装 OpenCV3.4（使用 GTK2.0 进行编译），以及 QGLViewer-qt4 的库版本，并对涉及 OpenCV 的问题代码进行修改。图 5-6 为在给定数据集上成功运行的截图。

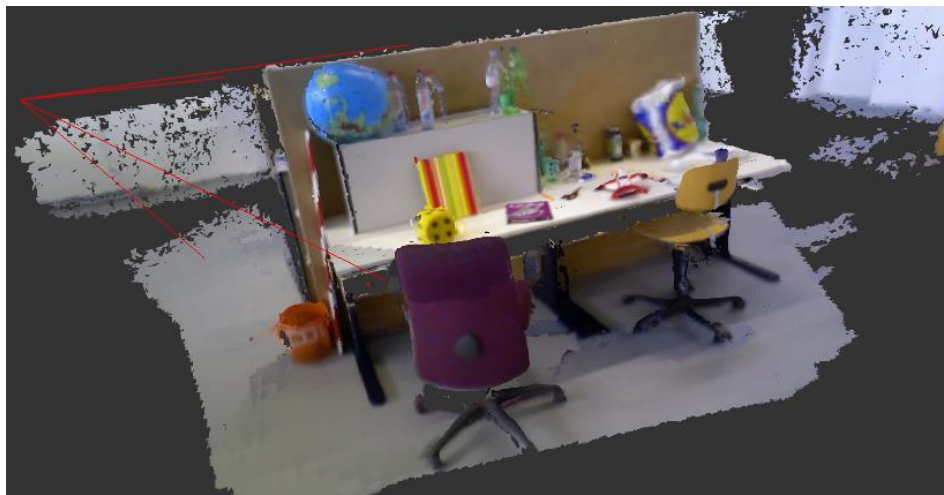


图 5-6 FastFusion 运行截图

在 PC 上成功运行后，我们将其移植到开发板上，由于开发板为 aarch64 架构，和 PC 的 x86-64 架构不同，因此在涉及系统调用部分的接口需要进行一些修改。具体问题记录见“已知 Bug 及问题记录”章节。

移植后需要将 FastFusion 移入开发板的 ROS 工作环境中，并且修改读取数据的方式，将顺序读取数据集改为回调函数式读取 ROS 传入的图像与位姿信息。

5.5 BundleFusion 环境配置

由于官方公布的[9]是基于 Windows 平台开发的，在 Ubuntu 系统上运行会涉及系统调用及渲染等方面的问题，因此我们主要参考[10]进行修改，其间环境配置遇到一些问题，具体问题记录见“问题记录”章节。在数据集上成功运行的截图如图 5-7 所示。

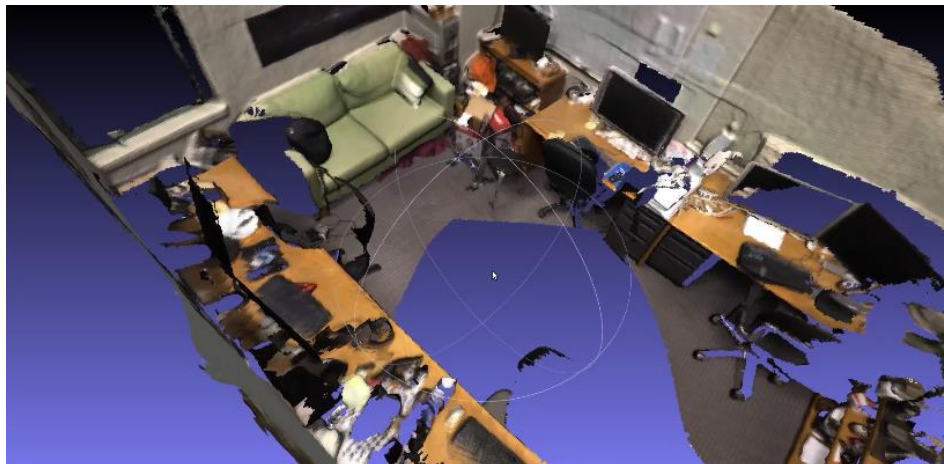


图 5-7 BundleFusion 运行截图

6 问题记录

6.1 已知 Bug 记录

Zora P1 配置 OpenNI2 相关 Bug

刚拿到相机和板子的时候，曾配置过 OpenNI2 SDK，遇到了一点问题。当时将问题记录下来并发到了 Orbbec 的开发者社区论坛上，并获得了 3 个点赞和 4 个回复。帖子链接如下：

https://developer.orbbec.com.cn/forum_plate_module_details.html?id=677

主要配置过程如下：

1. 下载 openni, <https://developer.orbbec.com.cn/download.html?id=64> 需要注意，须下载自己设备对应架构的版本，如果不知道自己设备架构，可以在终端运行 `uname -a` 查看。
2. 安装相关依赖项

```
1. sudo apt install build-essential freeglut3 freeglut3-dev
```

3. 进入 Sample/SimpleViewer 尝试编译

```
1. cd Sample/SimpleViewer
2. make
```

这边可能会碰到一个问题是，部分版本的 Openni 包里，Include 文件夹，部分是大写 I，部分是小写的，而 Makefile 里面似乎都是大写的，需要统一，否则会找不到头文件，之后编译成功后就可以通过 `simpleviewer` 查看深度图了。

此外 Tools 目录下的 NiViewer 也可以直接运行，但需要修改文件的权限：

```
1. chmod 777 ./NiViewer
```

FastFusion 移植相关问题记录

在配置移植 FastFusion 过程中主要遇到的问题与解决方案有：

1. OpenCV 类型问题：由于该库使用的 OpenCV 版本较老，新版本不再支持 `cv::DataType`，因此将所有诸如 `cv::DataType<uchar>::type` 的地方改为 `cv::traits::Type<cv::Vec<uchar, 3>>::value`，其他类型也是类似修改方式。
2. BOOST_JOIN 错误：编译代码时出现 `/usr/include/boost/type_traits/detail/has_binary_operator.hpp:50: Parse error at "BOOST_JOIN"错误`，修改

/usr/include/boost/type_traits/detail/has_binary_operator.hpp 文件，将

```
1. namespace BOOST_JOIN(BOOST_TT_TRAIT_NAME, _impl) {  
2. ...  
3. }
```

改为

```
4. #ifndef Q_MOC_RUN  
5. namespace BOOST_JOIN(BOOST_TT_TRAIT_NAME, _impl) {  
6. #endif  
7. ...  
8. #ifndef Q_MOC_RUN  
9. }  
10. #endif
```

3. 运行时读取图像出错：在 `cmakelist` 中链接 `opencv_imgcodecs` 库。
4. 运行时报错 `GTK2.0` 和 `GTK3.0` 不能同时使用：原因是因为使用的 `OpenCV` 库中用到的是 `GTK3.0`，而项目依赖于 `GTK2.0`，我们使用的解决方案是重新编译 `OpenCV3.4`，在编译时修改 `cmakelist` 中 `GTK` 的版本为 `2.0`。
5. 开发板头文件 `<xmmintrin.h>` 找不到的问题：该系列头文件为 `x86-64` 架构下的头文件，我们参考 <https://github.com/DLTCollab/sse2neo> 将其替换为 `aarch64` 架构下支持的 `#include "sse2neon.h"`。

BundleFusion 移植相关问题记录

1. 需要指定 `g++5` 编译:

```
1. sudo apt-get install g++-5  
2. export CC=/usr/bin/gcc-5  
3. export CXX=/usr/bin/g++-5  
4. cmake /path/to/your/project  
5. make
```

2. 关于类型 `BOOL`、`WORD` 等不识别的问题，是因为不正确的宏定义导致这些类型的声明被跳过了，在其声明处注释掉 `#ifdefined` 之类的条件即可。
3. 需要安装 `openni2-dev libglew3 libglew3-dev`。
4. `invalid device symbol` 问题：`CMakeLists.txt` 改成 `-arch=sm_30`（不同架构机器可能需要对应更改）

6.2 性能限制

KinectFusion 尝试

我们尝试在开发板上配置并运行 `KinectFusion` 的开源实现版本 `OpenCV KinFu[11]`，此为 `OpenCV` 提供的 `KinectFusion` 开源实现。我们成果在开发板上编译

并运行 OpenCV KinFu，但是由于 CPU 性能及内存大小的限制，运行时无法有效地完成重建任务。因此我们放弃了在开发板上搭载完整的 RGB-D 重建系统的方案，而是尝试将三维重建的位姿估计任务与重建渲染任务分离,再利用网络传输进行连接，将计算密集的任务由服务器完成，开发板仅需承担有限的计算任务。

FastFusion 性能限制

我们采用 FastFusion 在开发板上进行实时的模型融合与渲染，论文中描述采用双核 CPU 的架构运行（Intel Core i7-2720QM with 2.2 GHz and 8 GB RAM），模型融合线程的融合速率可以达到 45Hz，模型渲染线程的渲染速率约为 1Hz。在开发板上的融合速率有所下降，且总内存（2G）占比高达 70%，且内存交换高达 1G，可能会影响其他应用正常使用，也出现过卡顿、死机等现象，在与渲染的模型交互时也有一定程度卡顿。

参考文献与资料

- [1] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.
- [2] Steinbrücker, Frank, Jürgen Sturm, and Daniel Cremers. "Volumetric 3D mapping in real-time on a CPU." 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.
- [3] Lorensen W E, Cline H E. Marching cubes: A high resolution 3D surface construction algorithm[J]. *ACM siggraph computer graphics*, 1987, 21(4): 163-169.
- [4] Endres, Felix, et al. "3-D mapping with an RGB-D camera." *IEEE transactions on robotics* 30.1 (2013): 177-187.
- [5] Dai, Angela, et al. "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration." *ACM Transactions on Graphics (ToG)* 36.4 (2017): 1.
- [6] Wolfgang Kabsch. 1976. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32, 5 (1976), 922–923.
- [7] Michael Zollhofer, Matthias Nießner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, and Marc Stamminger. 2014. Real-time Non-rigid Reconstruction using an RGB-D Camera. *ACM TOG* 33, 4 (2014).
- [8] <https://github.com/tum-vision/fastfusion>
- [9] <https://github.com/niessner/BundleFusion>
- [10] https://github.com/nonlinear1/BundleFusion_Ubuntu_V0
- [11] https://docs.opencv.org/master/d8/d1f/classcv_1_1kinfu_1_1KinFu.html