

多 KINECT 实时室内动态场景重建技术

摘要

场景重建技术属于计算机视觉领域中的基础技术之一，在虚实结合和交互应用、人工智能机器人控制等方面有着重要作用。然而，受到硬件和软件的制约，人们一直无法实现实时重建动态场景。本课题旨在研究如何实现基于 Kinect 的实时室内动态场景重建技术。本课题采用多个 Kinect 从不同角度同时捕获场景，将它们的深度图和彩色图结合在一起，通过数据预处理、顶点构建、点云注册和表面重建等步骤后得到场景三维模型。整个流程均在 GPU 上实现以加速运算，实现了基于 GPU 的迭代最近点算法、基于 GPU 的八叉树构建和查找、基于有向距离函数的表面重建等关键算法。实验结果表明，整个算法运行效率达到 **TODO** fps；重建的模型较为完整，重建分辨率达到 **TODO** 毫米。从实验结果中可以看出，本课题实现的算法基本满足实时动态场景重建的要求，重建模型的精度满足非精确计算类应用的需求，但是从实时的标准而言运行效率仍有待提升。本课题所取得的成果表明，基于 Kinect 和 GPU 并行计算技术能够实现实时态场景重建。

关键词：实时动态场景重建，GPU，计算机视觉

REAL-TIME DYNAMIC INDOOR SCENE RECONSTRUCTION BASED ON MULTIPLE KINECT

ABSTRACT

Scene reconstruction is one of the underlying technologies in computer vision. It plays an important role in the application of augment reality and robot control. However, restricted by the technologies of hardware and software, real-time dynamic scene reconstruction has never been implemented. This subject is to study how to implement real-time dynamic scene reconstruction based on Kinect.

Multiple Kinect devices are used to capture a scene from different views at the same time. Depth and RGB images obtained are combined to modelling the scene. The reconstruction process mainly includes data preprocessing, vertex construction, point clouds registration and surface reconstruction. The entire process are GPU implemented to speed up the calculating. Key algorithms implemented in this subject includes Iterative Closest Point based on GPU, Octree construction and searching based on GPU, surface reconstruction based on the signed distance function.

Experimental results show that the algorithm efficiency reached **TODO** fps and models reconstructed are relatively complete, with the resolution of about **TODO** mm.

the experimental results prove that, the reconstruction process implemented meet the requirements of the real-time dynamic scene reconstruction and the precision of models reconstructed meet the needs of non-exact calculation application. But the efficiency still need to be improved. The outcomes of the subject showed it possible to reconstruct real-time dynamic scene based on the Kinect and GPU parallel computing.

Key words: real-time dynamic scene reconstruction, GPU, computer vision

目录

第一章 绪论.....	1
1.1 研究内容和意义.....	1
1.2 相关工作和文献综述.....	2
1.3 论文组织结构.....	2
1.4 本章小结.....	3
第二章 总体系统设计.....	4
2.1 硬件系统设计.....	4
2.2 软件平台和核心架构设计.....	4
2.3 算法流程概览.....	5
2.4 本章小结.....	7
第三章 基于彩色图和深度图构建三维点云.....	8
3.1 彩色图和深度图的预处理.....	8
3.1.1 彩色图的预处理算法.....	8
3.1.2 深度图的预处理算法.....	9
3.2 基于棋盘格的 Kinect 标定.....	10
3.2.1 内参标定.....	10
3.2.2 外参标定.....	11
3.3 基本三维点云的构建.....	13
3.4 点的权重和三维点云中的散点去除.....	15
3.5 本章小结.....	16
第四章 基于改进的 ICP 算法注册多组三维点云.....	17
4.1 ICP 算法的基本原理.....	17
4.2 基于位置-色彩距离的 ICP 算法改进.....	18
4.3 基于投影关系的快速最近邻搜索算法.....	18
4.4 ICP 算法的实现.....	19
4.5 本章小结.....	20
第五章 从三维点云提取表面.....	21
5.1 从三维点云提取表面的基本步骤.....	21
5.2 基于 GPU 的八叉树构建算法.....	22
5.2.1 八叉树的数据结构定义.....	22
5.2.2 邻居查找表和邻居查找算法.....	23
5.2.3 八叉树节点、顶点和边的构建算法.....	24
5.3 基于有向距离函数的等值面计算.....	25
5.4 Marching Cubes 表面提取算法.....	25
5.4.1 拓扑结构表的构建.....	26
5.4.2 基于八叉树的 Marching Cubes 算法.....	27
5.5 本章小结.....	29
第六章 实验结果和性能.....	30
6.1 对人物雕像的重建实验结果.....	30

6.2 对办公环境的重建实验结果.....	30
6.3 性能分析和调优.....	30
6.4 本章小结.....	30
第七章 总结.....	31
7.1 研究成果和问题.....	31
7.2 下一步工作.....	31
参考文献.....	33
谢辞	35

第一章 绪论

近些年来增强现实一词逐渐走进了普通大众的视野。正如[1]中所述,有别于虚拟现实,增强现实(Augmented Reality)试图将(一般是由计算机所合成的)虚拟的物体、场景等信息嵌入到真实的环境中,以增强真实场景的信息。由于真实环境的复杂性和难以控制性^[1],增强现实技术的发展较虚拟现实技术要困难许多。即便在最近基于增强现实技术的游戏和应用已经逐渐丰富起来,增强现实技术本身仍然存在诸多有待克服的困难和限制。

本课题所研究的主题是场景重建技术(更一般地,可以称为三维重建技术(3D Reconstruction)),属于计算机视觉领域中的基础技术之一,在虚实结合和交互应用、人工智能机器人控制等方面有着重要作用。场景重建技术是利用传感器和计算机,通过对场景的纹理、深度等元数据进行软件分析和处理自动(或半自动)地构建场景的三维模型。

然而由于真实场景中普遍存在各类表面形状复杂的物件,再加上受限于传感器技术,场景重建一直存在诸多困难和取舍,例如效率、硬件成本和重建精度之间的相互制约。本课题以满足实时应用的效率约束为基础,以 Kinect 这一大众化商业传感器捕获场景数据,研究如何重建精度满足一般用户视觉要求的场景模型。

1.1 研究内容和意义

过去,场景重建技术在工业模拟、智能机器人等诸多方面都承担着十分重要的作用。然而,极长的计算时间和昂贵的设备极大地阻碍了这一技术的推广应用。因此,基于廉价设备的实时场景重建技术具有巨大的应用价值。

人们都想相信未来的办公室会是全数字化的,倘若依托数字设备,人们能够将异地的工作环境无缝融合,这将对人们的工作方式产生革命性的影响。这样一个基于远程协作的增强现实应用不但出现在各种科幻片中,也出现在过去数十年人们的研究主题里。在早些年就有[2]提出了一个对于未来办公室的设想,但是由于技术限制并没有予以实现。后来人们在显示、交互等方面的研究上取得进展,使得一些基本的设想得以实现,例如[3]中通过投影在现有模型表面来模拟远程用户的头部,[4]中展现了他们自己设计的一种三维显示屏幕,在远程会议中可以使得坐在不同位置的用户看到不同角度的画面。

而本课题所研究的实时动态场景重建技术可以在上述应用中发挥巨大作用。倘若将实时动态场景重建技术应用与我们的办公环境,通过对场景的建模并在异地三维展现,异地的工作环境的无缝融合将得以实现。另外,它将整个真实场景数字化,使得在这个场景中进行的一切交互都能得到计算机的反馈,从而实现一个全智能的虚实结合办公空间。

Kinect 设备于 2010 年 6 月正式发布,最初作为 Xbox360 游戏主机配套设备发售,用以提供对体感游戏的支持。Kinect 由于其成本低廉拥有广泛的应用潜力而在近两年备受人们关注。Kinect 包含一个深度传感器、一个彩色摄像头、另外还内置了麦克风阵列和用于追踪人体的步进电机。其深度传感器采用红外结构光技术,能够获得传感器前方 1.2 米至 3.5 米的有效深度数据¹。Kinect 所具有的实时深度捕获能力使得实时场景重建技术的实现具备了硬件基础。

本课题最主要的研究重点在于实现基于 Kinect 满足实时交互要求的动态场景重建算法。在场景重建算法方面涉及图像去噪、可信点云构建以及基于点云的表面重建等多个问题;而

¹ Kinect 最大深度范围是 0.5 米至 9 米,在 1.2 米至 3.5 米之间的数据较为准确。

在实时计算方面涉及场景重建算法在 GPU 上的实现方案和快速近似算法的设计。为了得到较为完整的表面模型,本课题涉及同时使用多个 Kinect 设备,因此也涉及到对多组点云进行注册的研究。实时的动态场景重建使得当用户移动场景中的物体,做各种动作时,系统重建的模型能实时反应出这一变化,实现一些更加自然的交互手段。

1.2 相关工作和文献综述

由于三维重建技术的重要性,人们已经在这一方面研究了数十年,也提出了许多方法。这些方法各有优势和缺陷,适用于不同的应用场合。一般可以从原始场景数据的类型和重建结果的抽象等级两个方面对这些方法予以分类:

根据原始场景数据的类型主要分为两大类:基于深度数据的三维重建和基于纹理色彩数据的三维重建。大多数方法是基于深度数据的,例如[5-9]提出的重建方法都使用从特殊传感器(例如激光扫描仪等)获得的场景深度图重建三维模型的。另一类基于纹理色彩的重建有[10, 11]等,这些方法通过从图像中寻找特征点或特征区域来得到物体特征从而构建模型的。虽然基于纹理色彩的方法具有硬件成本低廉的优势(往往仅需要一个普通的摄像头),但物体表面的纹理色彩本身受环境光影响较大,很难设计一个鲁棒性高的算法,而且从本身就缺失第三维度信息的二维图像中反推三维信息的过程包含大量经验和假设,在一般情况下的重建精度往往不能满足要求,故基于深度数据的三维重建方法更为简单、稳定、易于推广应用。另外,还有部分工作是研究基于 RGB-D¹的三维重建技术,如[12]提出了基于深度数据和纹理色彩数据分别重建三维模型然后相互结合的方法改进鲁棒性和精度。

根据[8]中所述,由于重建结果的应用不同,三维重建技术从重建模型的抽象等级角度可以分3个层次:高层次关注的是一种或者一类物体,结果可能并不会给出一个具体可见的模型,这一层次的建模主要用于物体识别,例如[13];中层次使用诸如球、长方体等基本几何图元构建模型,这样构建出来的模型表面较为规整,有助于进行物体分割,例如[5]中所述就属于这一类;低层次则完全构建模型网格表面,不受任何假设的限制,能够重建出任意复杂表面的物体模型,用于对物体或场景精确建模以便重现,例如[6, 8, 9]中所提出的方法就属于这一类,他们通过从多个角度拍摄物体,获得大量深度图并结合在一起得到一个精细的三维模型。

近几年由于 GPGPU 的推广应用以及类似 Kinect 这样的具有实时深度获取的设备出现,实时三维重建技术开始成为研究热点。[9, 14]中所使用的重建算法虽然都是几年前就提出来的,但他们利用 Kinect 设备和 GPU 运算达到了实时重建的效率,这在三维重建领域是一个重要突破。美中不足的是,它们仍旧不能对动态场景进行实时重建,当场景中的一个物体位置发生变化时,它需要用数秒的时间才能将这个物体在原来位置上的模型彻底从重建出的场景中消除,并在新的位置重建。本课题根据实际需求改进了上述算法,实现了对动态场景实时重建。

1.3 论文组织结构

从论文的第二章开始将详细介绍本课题所进行的工作和取得的成果,将按照先总体后局部的顺序详细阐述设计思想和算法原理,然后通过实验和测试给出本课题所取得的结果以及尚存的不足之处,最后总结全篇并对未来的工作进行思考。

第二章主要阐述了本课题所实现的实时动态场景重建系统采用的硬件设备及布局设计、使用的软件平台和框架、核心重建算法的主要步骤等。

第三章至第五章详细阐述了重建算法的每个步骤,包括思想和原理以及主要的公式推导

¹ 即 RGB & Depth, 纹理色彩数据和深度数据的结合

等。

第六章给出了基于实时动态场景重建技术远程协作平台原型的设计和实现,并展现该平台的运行效果。

第七章给出了对系统进行的一系列实验以及结果,包括重建结果的精度、重建方法的鲁棒性以及性能等。

第八章总结。首先根据对第七章的实验结果的分析,总结本课题所取得的成果以及仍旧存在的问题。然后针对仍旧存在的问题提出对下一步工作内容的设想。

1.4 本章小结

本章主要介绍了本课题所进行的研究的基本情况。首先对研究领域进行了概要介绍,其次从场景重建技术目前存在的问题和实时动态场景重建技术所具有的意义引出本课题所研究的内容和主要目的。紧接着分类列举了过去二十年内人们在三维重建技术上进行的工作,它们的特点,并介绍了最新的研究进展和趋势,阐明选择 Kinect 设备和基于 GPU 的算法实现的原因。最后简要说明了本论文的组织结构和每章的主要内容。

第二章 总体系统设计

2.1 硬件系统设计

本系统采用 Kinect 设备作为输入设备，经由普通的个人计算机进行计算并得到场景的三维网格模型。本系统不需要其他额外的传感器，因此硬件平台的部署相对比较简单。

与[9, 14]中所采用的手持 Kinect 方式不同的是，本系统使用 3-4 个固定摆放在场景周围不同方位的 Kinect 设备（图 1），这是由应用环境所决定的。在远程协作的工作平台中，用户不可能手持 Kinect 进行工作，也不可能手持 Kinect 对着自己¹。另外，固定 Kinect 设备能简化重建动态场景的算法，也使重建结果更加稳定。

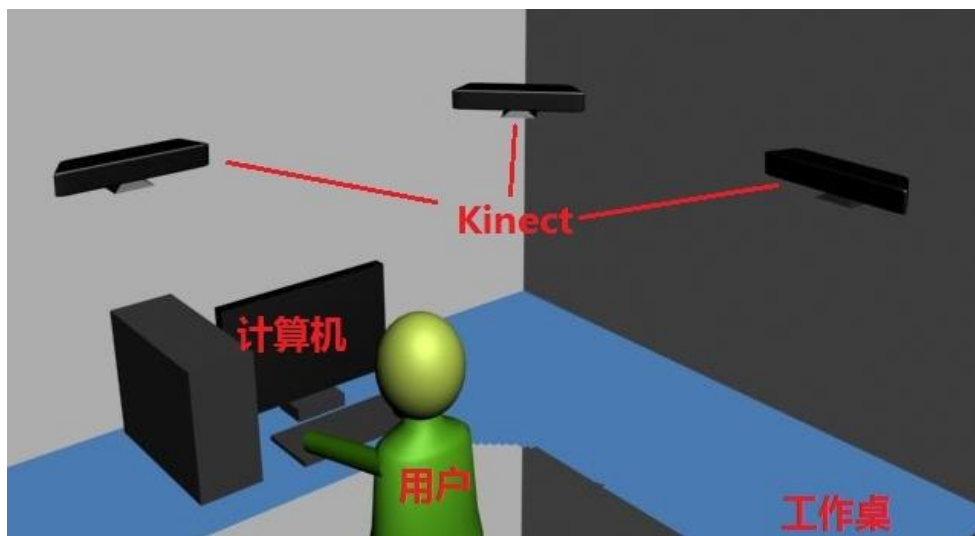


图 1 硬件系统布局示意图

本系统在部署完成后需要进行一次标定操作，包括对所有 Kinect 设备分别进行内参标定，对所有 Kinect 设备统一进行外参标定。采用棋盘格标定，具体阐述见下文 3.2 节。

2.2 软件平台和核心架构设计

本系统的设计运行平台为 Windows 7 操作系统，采用 Nokia QT 作为窗口显示框架、OpenNI 驱动 Kinect 设备并获取深度和纹理色彩数据，使用 DirectX 11 进行 GPU 运算和渲染。

虽然微软为 Kinect 设备发布了 Kinect SDK 开发库以及驱动程序，但该开发库更多地注重人体识别、追踪和语音识别等体感游戏需要用到的技术，在提供原始数据的接口和深度数据与纹理色彩数据的校准的方面不如 OpenNI，且在实际测试中较之 OpenNI 需要占用更多资源²，故本系统更适合使用 OpenNI 框架作为驱动 Kinect 的底层接口。

由于本系统的功能并不多，重点是在于核心功能——实时动态场景重建算法的实现，故本系统的架构设计也比较简单。图 2 描述了本系统的核心架构图，包含关键类和它们之间

¹ 由于 Kinect 有最近距离限制（约为 0.5 米），用户手持 Kinect 对着自己的距离过近以至于 Kinect 无法捕获那么近距离的表面深度信息。

² 在作者使用的计算机上（具体配置见下文 0），使用 Kinect SDK 获取数据并做深度图—纹理色彩图校准的执行性能约为 15fps，而使用 OpenNI 执行相同操作则能够达到 25fps 左右。

的关系。从该图中也可以很容易地看出本系统的主要执行流程。

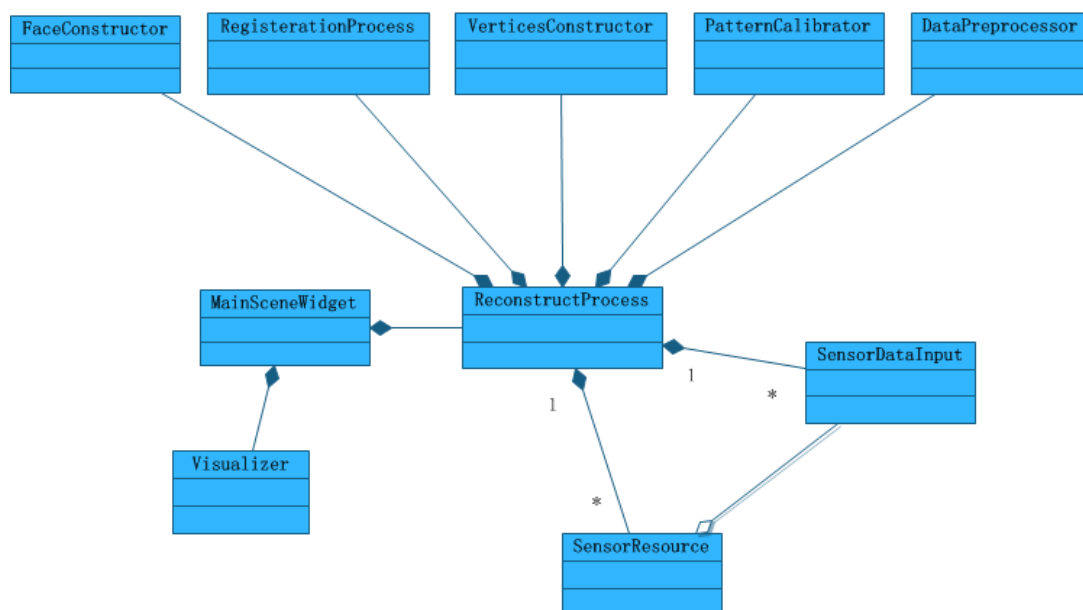


图 2 系统核心架构概览

2.3 算法流程概览

本节将概要性地介绍实时动态场景重建算法的执行步骤。从下一章开始，将对每个步骤进行详细阐述。上文第 1.2 节中提到了许许多多和三维重建算法有关的工作。虽然各论文中都提到了不同的方法，但凡是基于深度数据的三维重建算法，它们的主要流程都是一致的。一般基于深度数据的三维重建算法按先后顺序可以分为获取数据（深度数据和纹理色彩数据等）、对数据进行预处理、根据深度数据构建三维点云、对多组点云进行注册、从点云构建网格表面（图 3）。

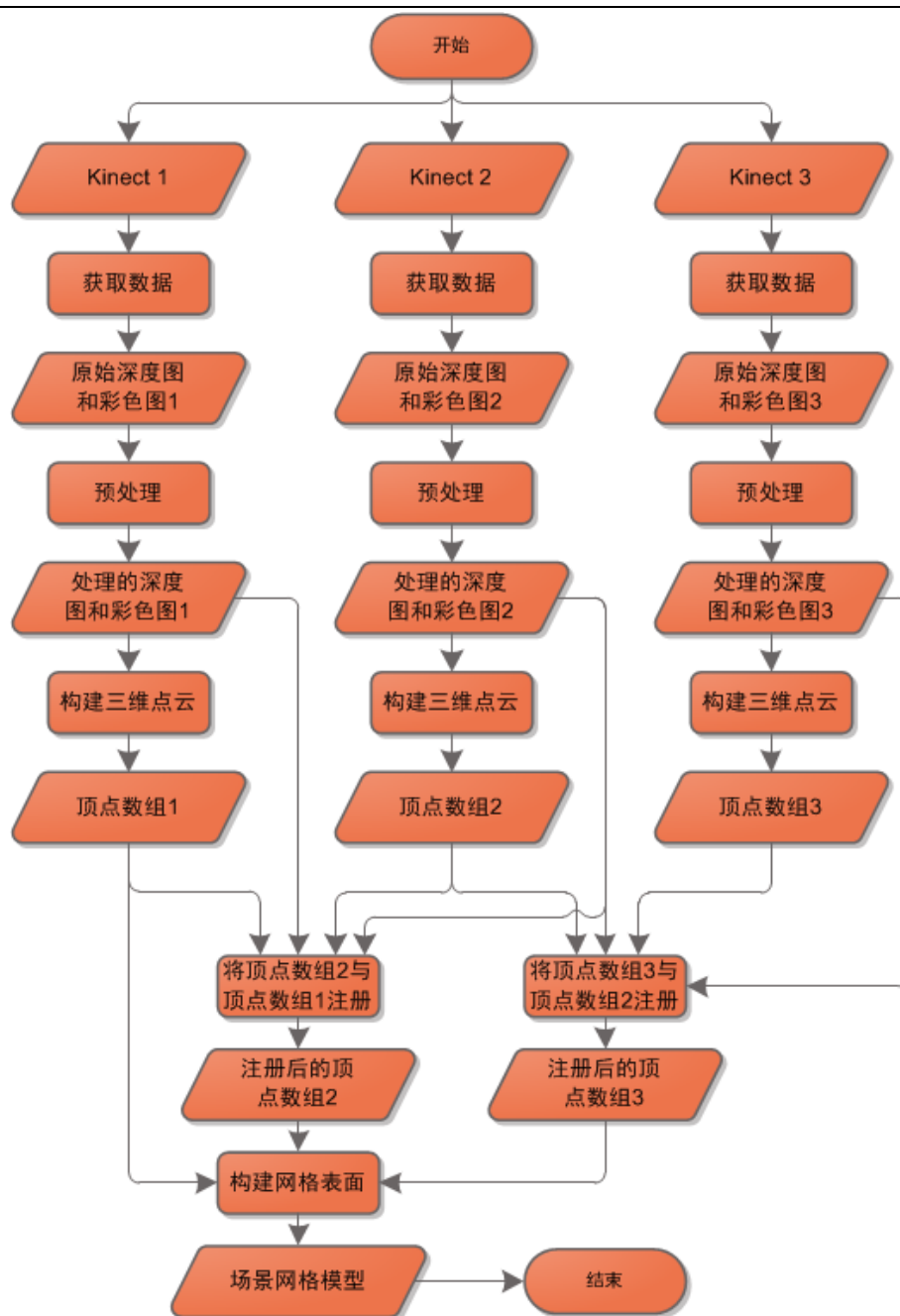


图 3 实时动态场景重建算法的主要流程（以使用 3 台 Kinect 为例）

对数据进行预处理：由于传感器的原理不同，获取的数据也存在各种不同的误差分布特性。例如在[6]中提到，光学扫描仪的误差往往与物体表面法向量和光轴所成的夹角有关，并且在物体的边缘获取的数据具有较强的不确定性。根据这些误差分布特性，需要对原始的深度数据进行特殊的预处理以降低噪声的干扰，从原始数据的层面减少误差。

根据深度数据构建三维点云：该步骤输入预处理后的深度数据和色彩纹理数据，主要执行以下操作：根据传感器的内参和外参信息计算深度数据中的某一像素对应应在三维场景坐标下顶点位置；根据该像素对应的色彩纹理数据标示该顶点的颜色；根据该顶点及其周围的邻居顶点¹计算该顶点的法向量。将几组传感器获得的数据中的所有像素均进行上述计算后，可以得到对应的几组三维点云。这时将他们可视化已经可以看出一定的重建效果，但是由于

¹ 在本文中，除特别说明外，一个顶点的邻居顶点都是指，在有效场景区域内，由其在深度图中对应像素的 8 邻域像素生成的顶点，且它们之间的距离小于某一阈值。

基于棋盘格的外参标定并不十分精确，从结构和纹理上可以看出多组点云并不能完全匹配。

对多组点云进行注册：该步骤旨在减小多组点云之间存在的变换偏差。注册即指将两组或两组以上的点云拼合在一起，使它们的重合区域匹配度最高。许多有关三维重建的论文（例如[9, 14]）都使用迭代最近点算法（Iterative Closest Point，以下简称 ICP）完成注册工作。ICP 算法最初是在[15]中被提出的。由于[14]中所提到的三维重建算法的输入中包含纹理色彩图，故其综合顶点的位置和颜色信息对 ICP 算法作了改进，增强了它的鲁棒性。本文中所使用的注册算法便是基于上述改进的 ICP 算法，同时提出了自己的改进方案以加快该算法的执行速度。

从点云构建网格表面：该步骤是重建算法的最后一步。虽然多组点云注册完成后，一个场景的重建基本完成，但是一些离散的点无论是用于可视化还是用于虚实物体接触检测都是远远不够的。该步骤即是从三维点云中构建较为完整的表面网格模型，具体涉及顶点合并、空洞填补和三角面片生成等算法。

2.4 本章小结

本章主要从总体层面阐述了本课题所研究的实时动态场景重建系统的实现方法。首先阐述了该系统所用到的硬件设备和布局设计，并给出了如此设计的理由。其次，简要介绍了系统所使用的软件框架、工具库等，并通过图示的方式描述了系统的主要工作流程。最后对本课题的主要研究内容—实时动态场景重建算法的执行步骤作了概要性的描述。

第三章 基于彩色图和深度图构建三维点云

3.1 彩色图和深度图的预处理

在上文 2.3 节中已经提到，由于传感器的原理不同，获取的数据会产生不同分布特征的误差。然而不管是何种传感器，获取的数据的误差是不可避免的。Kinect 上的深度传感器基于结构光技术，属于光学传感器的一种，故其获得的深度数据的误差符合[6]中所描述的光学传感器的误差分布特征。另外，由于硬件层面的原因，Kinect 获得的彩色图呈现出明显的条纹状噪声（图 4）。因此，在使用这些数据生成三维点云前，首先需要先进行二维图像层面的降噪处理。最后，预处理操作还根据 Kinect 的内参数据补偿彩色图和深度图的畸变，使后面的步骤可以不需要考虑畸变因子的影响，简化逆投影变换的运算。本节将针对彩色图和深度图分别阐述所使用的降噪算法。

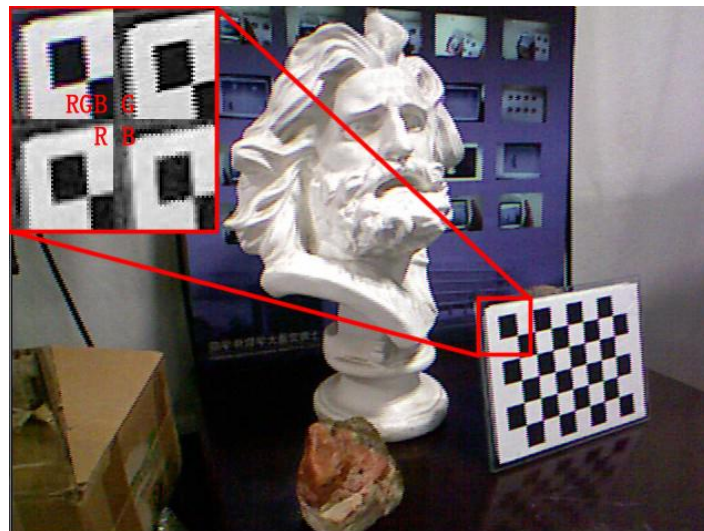


图 4 Kinect 获得的原始彩色图

从左上角的局部放大图中可以看出有明显的横向条纹状噪声

3.1.1 彩色图的预处理算法

根据上文所述，Kinect 获得的彩色图呈现出明显的条纹状噪声。虽然从总体上看这一噪声并不明显，但是它会影响到外参标定中的棋盘格角点提取的准确度和最近点查找算法的精度。

为了进一步分析条纹状噪声的特性，将彩色图的 RGB 三通道分别分开显示效果如图 4 右上角所显示的那样。从三通道显示的结果可以看出，G 通道的横向条纹较为明显，R 通道不太明显，而 B 通道没有横向条纹，但是存在轻微的纵向条纹。同时，这些条纹都呈现出各行分错的特性。

因为条纹状噪声特性简单，故降低这一噪声的方法也较为简单。对于原图像的三个通道，分别针对奇偶行（列）相对原图像做不同程度的偏移，可以明显降低条纹噪声（图 5）。

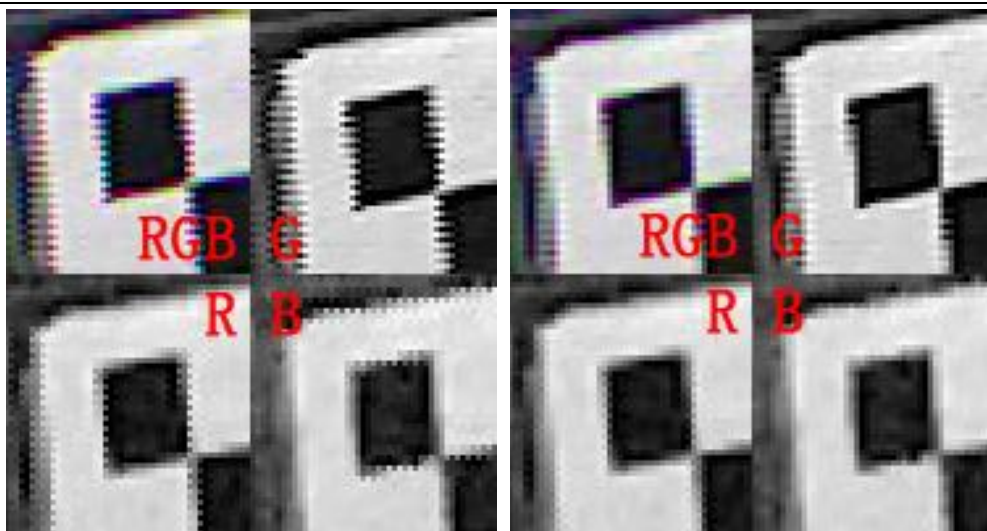


图 5 彩色图预处理的效果

左图：处理前，条纹明显；右图：处理后，条纹几不可见

3.1.2 深度图的预处理算法

Kinect 的深度传感器属于光学传感器中的一类。因此它所产生的误差收到多个因素的影响，例如光强度（对于 Kinect 来说是环境红外光场的强度），距离等。但是由于室内的环境红外光强一般都比较弱，而且在一定的距离范围内误差随距离的变化并不明显，因此本课题并不考虑深度误差受光强和距离的影响因素，而是着重考虑了和场景中物体表面结构有关的一些重要的影响因素。另外 Kinect 获得深度数据也具有较强的噪声。

在本节，将首先介绍深度图的降噪处理算法。在后文 3.4 节中将考虑其他的误差特性。

本课题中，降噪处理使用高斯滤波的变种。该处理算法主要步骤如算法 1 所示。

算法 1 深度图预处理算法

输入： 原始深度图 d_{in} ，格式为 16 位单通道
输出： 预处理后的深度图 d_{out} ，格式与输入相同
过程：
初始化输出图像，所有像素值归 0
对于图像中的每个像素点 p ：
遍历以 p 为中心的 5×5 区域，对于每个像素点 q ：
倘若 $ d_{in}(p) - d_{in}(q) > t_d$: $d_{out}(p) = \omega(q - p)d_{in}(q)$;
否则: $d_{out}(p) = \omega(q - p)d_{in}(p)$;
归一化 $d_{out}(p)$;
结束

该变种在进行高斯模糊的基础上仍保留了强边缘的清晰。经过这一处理的深度图所构建的三维模型不会在边缘部分产生毛刺（图 6）。

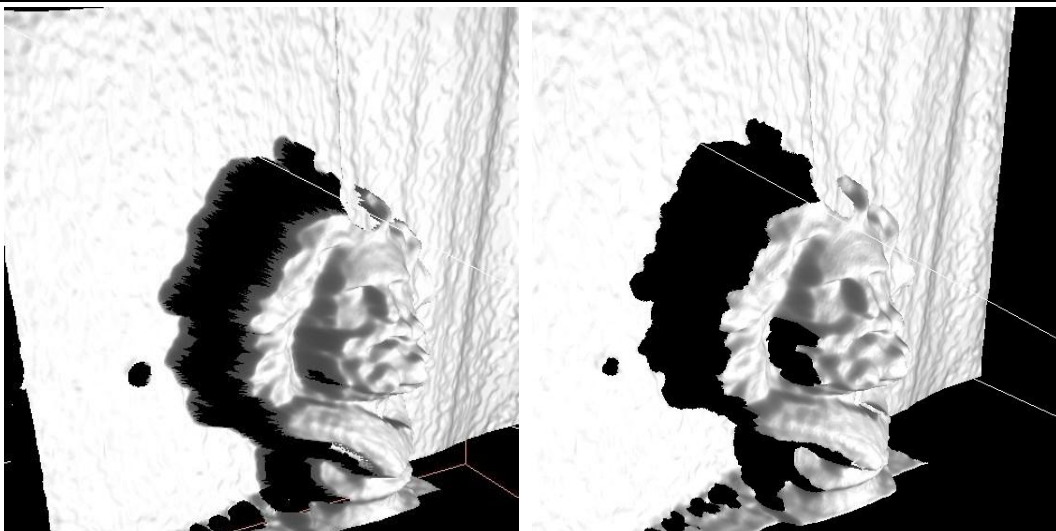


图 6 高斯滤波和本课题设计的处理算法对深度数据的处理效果对比
左图：基于使用高斯滤波后的深度图重建的人物雕像模型边缘毛刺明显；右图：基于使用本课题设计的处理算法处理后的深度图重建的人物雕像模型边缘没有毛刺

3.2 基于棋盘格的 Kinect 标定

在进行三维点云的构建前，首先需要知道 Kinect 设备的内参和外参数据。在计算机视觉领域中，内参是指光学传感器的内部参数，该参数与硬件有关，而与传感器在场景中所处的方位无关；外参则是用于表征传感器在场景中所处的方位的参数。

3.2.1 内参标定

根据计算机视觉领域常用的针孔摄像机模型（图 7），在场景中的某个点通过与摄像机光心的连线投影到摄像机平面，这之间的投影关系满足

$$sp' = AP' \quad (1)$$

其中 $P' = (X, Y, Z)^T$ 为一个点在摄像机坐标系下的三维坐标值， $p' = (u, v, 1)^T$ 为该点投影到摄像机图像平面上的齐次坐标， $A = \begin{pmatrix} f_x & 0 & c_u \\ 0 & f_y & c_v \\ 0 & 0 & 1 \end{pmatrix}$ 为摄像机内参矩阵。

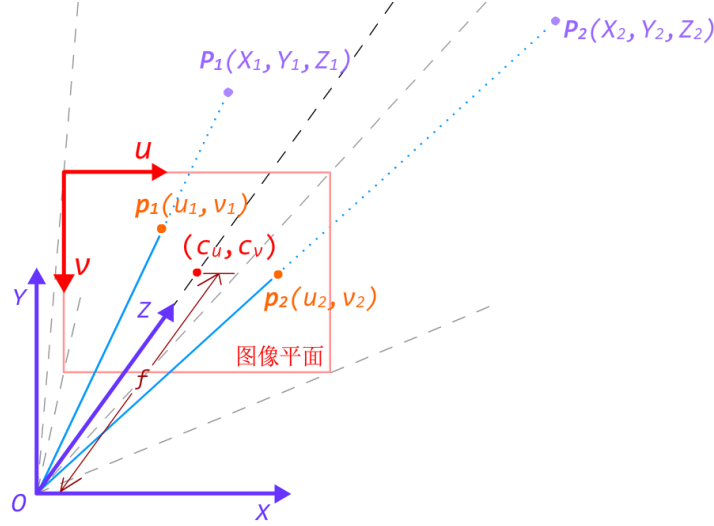


图 7 针孔摄像机模型

上述公式仅适用于理想摄像机。在实际中，考虑到实际结构与理想模型的差异和硬件制造工艺上的误差，上述投影变换关系还应加入畸变因子的影响。实际的投影变换公式如下：

$$\begin{cases} u = f_x(x'(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x'y' + p_2(r^2 + 2x'^2)) + c_x \\ v = f_y(y'(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_2x'y' + p_1(r^2 + 2y'^2)) + c_y \end{cases} \quad (2)$$

其中 $x' = \frac{x}{z}$, $y' = \frac{y}{z}$, $r^2 = x'^2 + y'^2$ 。

公式(1)中的 A 和公式(2)中的 $(k_1, k_2, p_1, p_2, k_3)^T$ 合在一起被称为传感器的内参数据。由于内参数据仅和硬件本身有关，故仅需在系统部署前求得一次即可。本课题选用基于棋盘格的张正友标定法对使用的 Kinect 设备进行内参标定。张正友标定法是由张正友在[16]中所提出的摄像机内参标定方法。由于该方法使用起来简单快速，故在提出后即被广泛应用。OpenCV 函数库还实现了针对该方法的内参计算函数可供开发者直接调用。

3.2.2 外参标定

外参标定用以决定 Kinect 设备在场景中放置的方位。由于本课题使用多个 Kinect 设备重建场景，由不同 Kinect 设备所提供的数据中构建的多组点云必须在一个统一的三维坐标系下才能拼合在一起，所以预先需要得到所有 Kinect 设备相对某一三维坐标系的方位。该三维坐标系可以随意定义，故本课题中使用棋盘格标定板定义该三维坐标系，以下称之为场景坐标系（图 8）。



图 8 使用棋盘格标定板定义场景坐标系

外参可以用一个 4 阶矩阵 M_4 表示，它表示从场景坐标系到 Kinect 设备的坐标变换。传统的摄像机外参标定法可以在已知摄像机内参的条件下从棋盘格角点在场景坐标系中的坐标及它们投影在摄像机图像平面中的坐标之间的多组一一对应关系求得摄像机的外参矩阵。本课题中所采用的外参标定方法利用了 Kinect 设备的深度数据，其标定结果比传统方法更加精确。该方法较之传统方法的区别在于使用 Kinect 坐标系下的三维坐标而非图像平面内的二维坐标。由于 Kinect 内参已知，根据某个像素的深度值可求出其对应 Kinect 坐标系下的三维坐标。利用 4 个（或 4 个以上）不共面的点在场景坐标系下的三维坐标和 Kinect 坐标系下的三维坐标可以列出线性方程组：

$$M_4(P_1 \ P_2 \ \dots \ P_n) = (P_1' \ P_2' \ \dots \ P_n') \quad (3)$$

其中 P_1, P_2, \dots, P_n 为 n 个棋盘格角点和根据角点计算的不在 z 轴平面上的点（为了满足不共面约束）在场景坐标系下的坐标， P_1', P_2', \dots, P_n' 为它们在 Kinect 坐标系下的坐标。通过最小二乘法即可求解系数矩阵 M_4 。显示了外参标定过程中的角点检测和结果验证。

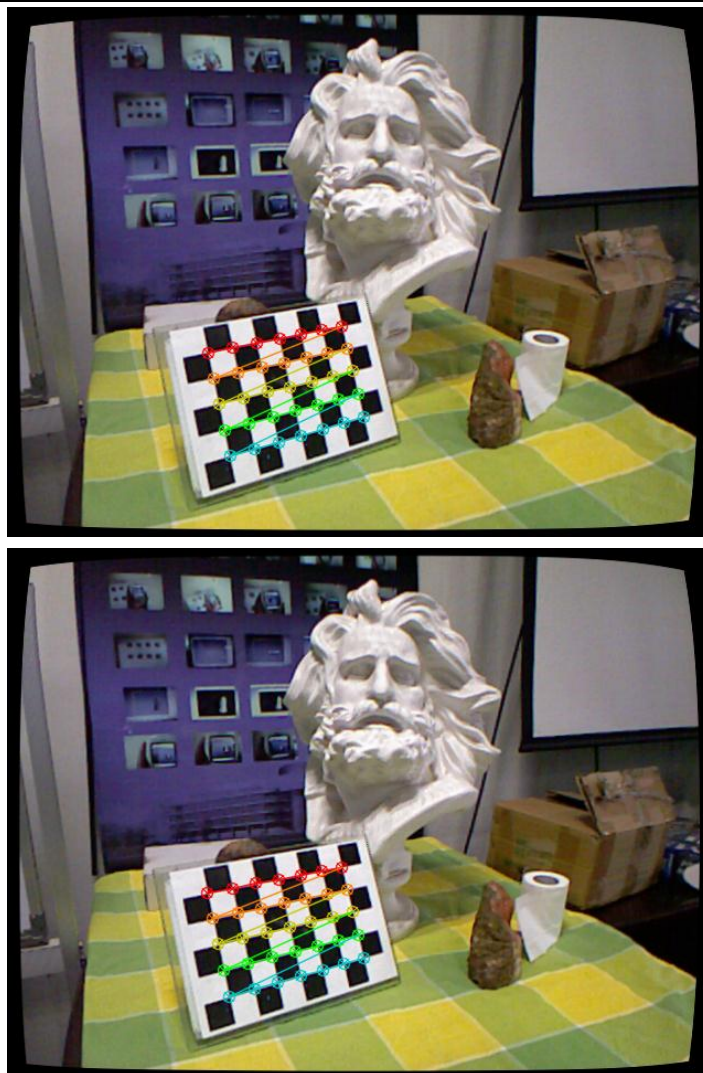


图 9 外参标定过程

上图：对彩色图像做角点检测识别棋盘格；下图：外参计算结果的验证

3.3 基本三维点云的构建

场景重建算法的第二步是从深度图和彩色图构建三维点云。对于每个 Kinect 设备，都会由该步骤构建一组对应的三维点云。该点云中的每个顶点包括位置、法向量和颜色、权重等信息。本节将阐述如何由深度图和彩色图生成上述信息中的位置、法向量和颜色，权重的生成将单独列入下一节阐述。

在该步骤中，并没有顶点的颜色做过多的处理，仅是将其设为对应的像素在彩色图中的颜色。

由深度图生成顶点位置涉及两个坐标变换。首先根据内参将深度图中的像素做逆投影变换，求得 Kinect 坐标系下的三维坐标。再根据外参的逆矩阵将该三维坐标变换到场景坐标系下，即得到对应顶点的位置。由于在预处理过程中已经根据内参中的畸变因子进行了畸变补偿，故此处的逆投影变换不需要再考虑畸变因子。整个变换求解过程如公式(4)所示。

$$\mathbf{P} = M'_4 \mathbf{P}' = M'_4 \begin{pmatrix} \frac{f_x}{d(u,v)}(u - c_x) \\ \frac{f_y}{d(u,v)}(v - c_y) \\ d(u,v) \\ 1 \end{pmatrix} \quad (4)$$

最后，顶点的法向量由该顶点和其周围的邻居顶点的位置计算而得。假设由该组点集生成的网格模型中，任意三个邻居顶点之间都会生成一个三角面片。根据一般网格模型的法向量计算方法，每个顶点的法向量等于以它为顶点的所有三角面片的法向量之和（图 10）。

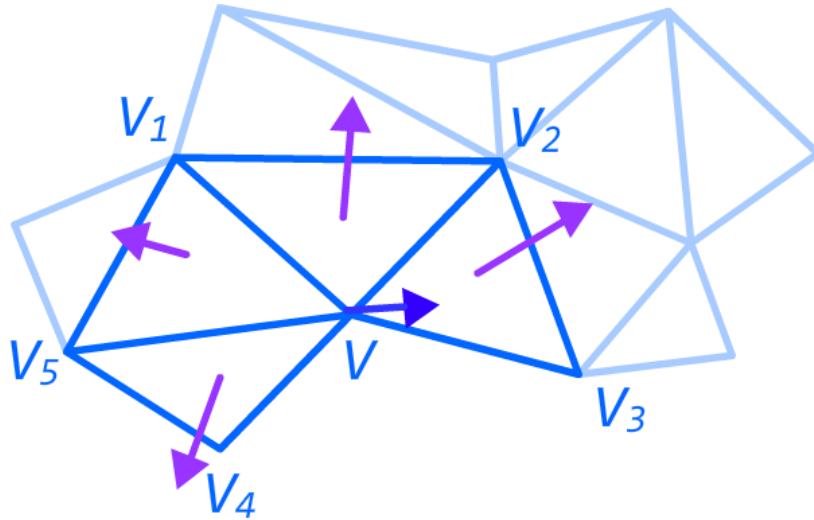


图 10 顶点法向量由所有以它为顶点的三角面片的法向量加和而得
整个构建算法如算法 2 所示。

算法 2 三维点云构建算法

输入：彩色图 c （格式为 8 位三通道）和深度图 d （格式为 16 位单通道）

输出：一组顶点元素，其中每个元素包含位置 P 、法向量 N 、颜色 C

过程：

对于图像平面中的每个像素 p ：

添加一个新的顶点元素 V ；

倘若 $d(p) > d_{max}$ ：

$V.P = \text{无效坐标}$ ；

否则：

由公式(4)计算 $V.P$ ；

$V.C = c(p)$ ；

对于图像平面中除第一行和第一列外的每个像素 $p = (u, v)^T$ （对应顶点 V ，记场景坐标为 P ）：

倘若 $V.P$ 是有效坐标：

定义 $p_1 = (u - 1, v - 1)^T, p_2 = (u, v - 1)^T, p_3 = (u - 1, v)^T$ ，

对应场景坐标 P_1, P_2, P_3 ；

对于 P_1, P_2, P_3 中任意两个符合 P 邻居顶点要求的顶点 P_a, P_b （ PP_aP_b 逆时针排列）：

记 P_a, P_b 对应的顶点为 V_a, V_b ；

构建一个三角面片 T ；

$T.N = (P_b - P) \times (P_a - P)$ ；

$V.N = V.N + \frac{T.N}{|T.N|}, V_a.N = V_a.N + \frac{T.N}{|T.N|}, V_b.N = V_b.N + \frac{T.N}{|T.N|}$ ；

结束

3.4 点的权重和三维点云中的散点去除

由于 Kinect 深度传感器内在的误差特性，其获得的深度图中每个像素的可信度都不一样。在重建过程中将该可信度考虑进来可以使得重建结果更加精确。因此，每个顶点都额外包含一个权重属性。在构建点云的过程中，根据 Kinect 深度传感器的误差特性计算每个顶点的权值。

B. Curless 等人在[6]中总结他人工作并指出基于光学的深度传感器往往具有如下特性：

- （1）误差和表面法向量与传感器光轴的夹角有关，误差随夹角增大而增大。
- （2）靠近物体边缘的深度数据具有很强的不稳定性。

Kinect 深度传感器基于结构光技术，因此其特性也满足上述两点。本课题在对 Kinect 获得的深度图进行分析后，得出与和上述两点相对应的结论：

（1）深度数据可信度与该深度数据所生成顶点的法向量与 Kinect 光轴夹角的余弦成正比。即

$$\omega(P) = \cos \frac{N(P) \cdot v_0}{|N(P)||v_0|} \quad (5)$$

（2）在物体边缘顶点¹所对应像素的 7×7 邻域内，深度数据是不可信的。即

$$\omega(P) = 0, P \in \{P_i | p_i \text{ 的 } 7 \times 7 \text{ 邻域内有至少一个边缘顶点, } i = 1 \dots n\} \quad (6)$$

¹ 边缘顶点是指那些邻居顶点不足 8 个的顶点。即倘若某个顶点邻居顶点满 8 个，则它是一个内部顶点。

根据以上两点结论，在求得所有顶点的位置和法向量后，即可计算它们的权重。加入权重计算后的完整三维点云构建算法如算法 3 所示。

算法 3 加入顶点权重计算的三维点云构建算法

输入：彩色图 c （格式为 8 位三通道）和深度图 d （格式为 16 位单通道）

输出：一组顶点元素，其中每个元素包含位置 P 、法向量 N 、颜色 C 、邻居个数 n 和权值 ω

过程：

对于图像平面中的每个像素 p ：

添加一个新的顶点元素 V ；

倘若 $d(p) > d_{max}$ ：

$V.P = \text{无效坐标}$ ；

否则：

由公式(4)计算 $V.P$ ；

$V.C = c(p)$ ；

对于图像平面中除第一行和第一列外的每个像素 $p = (u, v)^T$ （对应顶点 V ，记场景坐标为 P ）：

倘若 $V.P$ 是有效坐标：

定义 $p_1 = (u - 1, v - 1)^T, p_2 = (u, v - 1)^T, p_3 = (u - 1, v)^T$ ，

对应场景坐标 P_1, P_2, P_3 ；

对于 P_1, P_2, P_3 中任意两个符合 P 邻居顶点要求的顶点 P_a, P_b （ PP_aP_b 逆时针排列）：

记 P_a, P_b 对应的顶点为 V_a, V_b ；

构建一个三角面片 T ；

$T.N = (P_b - P) \times (P_a - P)$ ；

$V.N = V.N + \frac{T.N}{|T.N|}, V_a.N = V_a.N + \frac{T.N}{|T.N|}, V_b.N = V_b.N + \frac{T.N}{|T.N|}$ ；

$V.n = V.n + 1, V_a.n = V_a.n + 1, V_b.n = V_b.n + 1$ ；

对于图像平面中的每个像素 p （对应顶点 V ）：

倘若 $V.P = \text{无效坐标}$ 或 p 的 7×7 邻域内存在某个像素，其对应顶点的邻居个数不等于 8：

$V.\omega = 0$ ；

否则：

根据公式(5)计算 $V.\omega$ ；

结束

需要注明的是，所有权值为 0 的顶点被视为无效顶点，也即不再参与到在后续的计算中。同时可以发现，经过上述算法计算后，所有小于等于 8×8 的独立点集将被移除，这样也就去除了三维点云中由于各种外界因素或内部噪声所造成的散点集。

3.5 本章小结

本章详细阐述了本课题所设计的场景重建算法的第一个步骤。该步骤的输入为从 Kinect 接收到的原始深度图和彩色图，输出为包含位置、法向量、颜色和权重信息的顶点数组。该步骤主要流程为：原始数据的预处理和三维点云的构建。对于从每个 Kinect 设备接收到的数据，都要分别经过这一步骤的运算。所得到的多组点云将作为场景重建算法的下一个步骤的输入。该步骤的重点在于通过预处理和权重计算降低由于各种外在或内在因素而产生的误差的影响，提高最后的重建结果的精度。另外该步骤中所需要的 Kinect 内参和外参数据的求取方法也在本章中予以阐述。

第四章 基于改进的 ICP 算法注册多组三维点云

虽然事先已经对所有的 Kinect 设备进行了外参标定,使得对于每个 Kinect 设备数据分别构建出的点云都在统一的场景坐标系下,但是外参标定过程中的微小误差会在构建三维点云时的逆投影变换放大。在实践中发现,几组点云在同一坐标系下被可视化后可以很轻易地发现它们之间存在的偏差,离场景原点越远偏差越大。这使得离场景原点较远的那些区域的重建结果十分地不精确。因此仅通过外参标定所得到数据统一这些点云的坐标系是不够的。

常规的三维重建算法,为了重建一个完整的模型,也都会涉及到将多个传感器数据组合在一起的操作。这需要找到一个三维变换,使得一组点云(源)在经过变换后和另一组点云(目标)重合部分的偏差最小。ICP 算法即是用来求这个三维变换的算法。它最初是在[15]中被提出的,而后被许多三维重建工作引用。例如[9, 14]中都提到了使用该算法注册点云。本课题所设计的注册算法也是基于 ICP 算法思想,结合了[14]中提出的改进思想和本课题所提出的改进思想。

4.1 ICP 算法的基本原理

ICP 算法的基本思想其实很简单。它是一个迭代改进算法,每个迭代可以分为三个步骤:

- (1) 为源中的每个点在目标中寻找离它最近的点,形成一组点对;
- (2) 求一个三维变换,使得源中的每个点经过该变换后与其在目标中对应的那个点的距离平方和最小;
- (3) 对源应用该三维变换。

迭代终止条件为

- (1) 源和目标之间的偏差小于预先设定的阈值。算法报告说这两组点是可重合的;
- 或者
- (2) 迭代次数达到了设定的最大迭代次数。算法报告说这两组点是不可重合的。

作为每个迭代的第一个步骤,寻找点对的过程事实上是一个最近邻搜索算法。虽然最近邻搜索算法已经经过了数十年的研究,也已经存在了许多成熟的算法,比如 K-d 树、八叉树等,但它们都是基于串行思想的。在 GPGPU 时代到来后人们发现,在 GPU 上并行执行的简单蛮力搜索其效率甚至是相比这些经过精心设计的算法都极具竞争性^[17]。由于并行算法需要考虑许多诸如数据依赖、数据冲突等问题才能充分利用 GPU 的并行处理能力,传统的基于串行思想而设计的算法并不适用于在 GPU 上进行并发执行。近几年,人们也针对这一问题进行了许多研究,包括用并行思想重新设计传统的 K-d 树算法(例如[18])以及在简单蛮力搜索基础上设计更为高效的蛮力搜索算法(例如[17])。本课题同样在蛮力搜索算法的基础上,依据本课题中三维点云都是从高度结构化的二维深度图构建的这一前提条件,提出了更有针对性且更为高效的最近邻搜索算法。具体细节见下文 4.3 节。

在第二步中,求解三维变换的过程是一个求解最小误差平方和算法。一个三维变换可以分为平移、旋转和缩放三个部分组成。即

$$P' = sRP + r_0 \quad (7)$$

其中 s 为缩放标量, R 为 3×3 旋转矩阵, r_0 为三维平移向量。于是第二步骤的问题即求一组 s, R, r_0 满足

$$\min \sum_{i=0}^n (r_{r,i} - sRr_{l,i} - r_0)^2 \quad (8)$$

其中 $r_{r,i}$ 是目标中的一个点, $r_{l,i}$ 是源中的一个点。该算法由 B. K. P. Horn 在[19]中提出。该算法用单位四元数来表示旋转。B. K. P. Horn 通过严格的数学推导后给出了如下结论:

- (1) 最佳的旋转四元数是与一个对称矩阵的最大正特征值相关的特征向量。
- (2) 最佳的缩放标量 s 为两组点云到它们的质心距离的均方根的比例。
- (3) 最佳的平移向量 r_0 为目标的质心与经过旋转和缩放后的源的质心的差。

该结论详细推导过程请参阅[19]。

4.2 基于位置-色彩距离的 ICP 算法改进

如上文 4.1 中所述, ICP 算法的第一个步骤是要在目标中为每个源中的顶点搜索最近的顶点。传统的 ICP 算法是基于最一般的距离概念定义最近邻。基于这样的概念得到的点对关系可能与正确的点对关系相差很远(图 11)。而如果结合色彩信息对距离进行新的定义,能大大增加最近邻搜索的鲁棒性。基于 RGB-D 的最近邻搜索能够在结构特征不足的区域(如平坦的区域)依靠纹理色彩特征弥补,而在纹理色彩特征缺乏的区域(如纯色的区域)依靠结构特征弥补,从而提高了最近邻搜索的正确率^[14]。

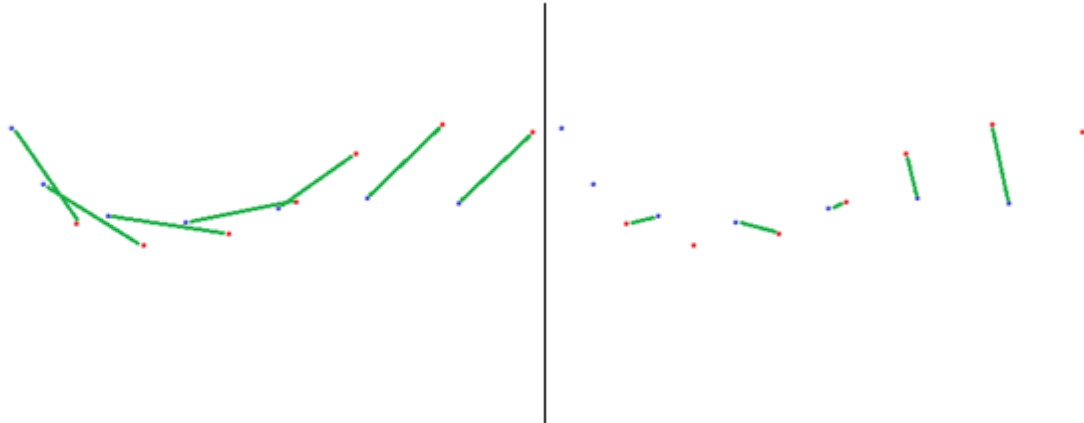


图 11 一种基于一般距离进行最近邻搜索产生不正确结果的情况

左图: 正确的点对关系; 右图: 执行最近邻搜索后产生的错误的点对关系

新的距离定义由位置空间下的欧几里得距离和 RGB 空间下的欧几里得距离相加而得。加入权重因子后可以调节两者对最终距离的影响程度。

$$\begin{cases} d = \omega_p |v_1 \cdot p - v_2 \cdot p| + \omega_c |v_1 \cdot c - v_2 \cdot c| \\ \omega_p + \omega_c = 1 \end{cases} \quad (9)$$

其中 d 为两顶点间的位置-色彩距离, $v_i \cdot p$ 为某顶点的位置坐标, $v_i \cdot c$ 为某顶点的 RGB 色彩值。 ω_p, ω_c 分别为位置和色彩对最终结果的影响权值。

4.3 基于投影关系的快速最近邻搜索算法

有了距离定义, 就可以通过最近邻搜索算法构建点对。上文 4.1 节中提到, 基于 GPU 并行执行的蛮力搜索算法执行效率可以和精心设计的串行最近邻搜索算法媲美。但是其效率仍不能满足实时计算的要求。为此许多人都在蛮力搜索算法的基础上研究了更高效的 GPU

最近邻搜索算法。

本课题使用了作者自己设计的 GPU 快速最近邻搜索算法也是基于蛮力搜索的基础，通过尽可能减少搜索范围来达到提高效率的目的。该算法不是一个通用的最近邻搜索算法，它依靠一个前提条件：每组点云都是从二维深度图中生成的，即他们中的每个点与某个深度图中的像素一一对应，形成已知的投影关系。由于这种投影关系的存在，每组点云都是高度结构化的，因此搜索范围可以极度地缩小。

基于投影关系的快速最近邻搜索算法的主要根据是在深度图中排列的像素通过逆投影生成的三维顶点也具有一定的排列关系。也即两个距离相近的顶点，其在深度图上的投影之间的距离也很可能很近（只要这两个点深度值不是很小）。因此若假设某个顶点 v_1 的最近邻顶点是 v_2 ， v_2 的投影 p_2 有很大的概率落在 v_1 的投影 p_1 的小邻域内。

根据以上思想设计的基于投影关系的快速最近邻搜索算法如算法 4 所示。

算法 4 基于投影关系的快速最近邻搜索算法

输入：源 $\{V_s\}$ ，目标 $\{V_t\}$ ，目标深度图 d_t ，目标内外参数 $param_t$

输出：点对记录 $\{Rec_{s,t}\}$

过程：

对于 $\{V_s\}$ 中的每个元素 $V_s(i)$ ：

$p'_s = \text{project}(V_s(i), P, param_t)$;

$d_{min} = \text{无穷大}, index_{min} = -1$;

对于以 p'_s 为中心的 $n \times n$ 矩形区域中的每个像素 p'_t ：

找到在目标中的对应点 $V_t(j)$;

倘若（由公式(9)计算得） $d(V_t(j), V_s(i)) < d_{min}$ 则：

$d_{min} = \text{无穷大}, index_{min} = j$;

$Rec_{s,t}(i) = j$;

结束

在具体实现过程中，对上述算法进行了针对 GPU 架构的修改和优化。

另外，由于该算法给出的是目标中距离源的某个点距离最小的点，因此点对中会产生源中的多个点对应到目标中的一个点的情况。为了消除这一现象，本课题中该算法被执行两遍，其中一遍是从源到目标，另一边是从目标到源，然后判断双方的最近点是否是对方（即是否满足 $Rec_{t,s}(Rec_{s,t}(i)) = i$ ）。只有满足该条件的点对会被加入最终结果。

4.4 ICP 算法的实现

在本课题中，ICP 算法的执行并不完全在 GPU 上进行。虽然在显存和内存之间传递数据存在较高的延迟，一般情况下应尽量避免。但由于 ICP 算法的后半部分的运算都是围绕着 4×4 的矩阵进行的，该部分的运算并行度并不高，复杂的分支也不适合 GPU 的并行运算，在 GPU 上完成这部分运算所需的时间会比将数据传回内存交由 CPU 进行运算更长。鉴于这一考虑，ICP 算法的实现采用的是 GPU 和 CPU 相结合的方式。图 12 具体流程步骤在 GPU 和 CPU 上的分布。

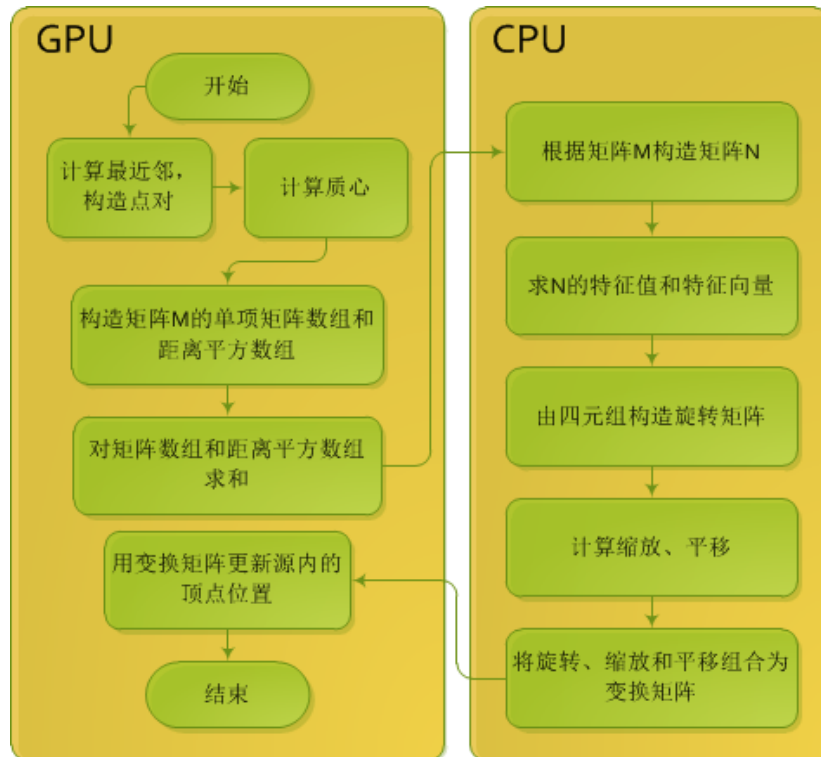


图 12 ICP 算法执行流程在 GPU 和 CPU 上的分布

注意到在移交给 CPU 继续计算的时候, 需要传给 CPU 的数据仅仅是两组点云的质心坐标、距离平方和以及 3×3 矩阵 M ; 而计算完成后再传回 CPU 的数据也仅仅是计算得的 4×4 变换矩阵。因此这部分计算的切换需要在 GPU 和 CPU 之间传递的数据量很小, 实际上传递数据所花费的时间几乎可以忽略不计 (在本课题的实验机器上大约需要 0.454ms), 远远比 GPU 直接做这部分计算相比 CPU 的时间差小。

4.5 本章小结

本章主要阐述了用于点云注册的 ICP 算法的设计与实现, 包括本课题中所采用的改进策略。首先 ICP 算法的主要思想和执行步骤。该算法是在[15]中被提出的, 而其理论基础则来源于[19]。[14]提出了一种改进策略, 即根据 RGB-D 定义距离, 使得色彩纹理信息也纳入最近邻搜索依据, 增加了算法的鲁棒性。本课题设计并实现的 ICP 算法也采用该方法定义距离。另外本课题设计了新的最近邻搜索算法, 利用由深度图生成的点云具有高度结构化的特点进行搜索范围的缩减, 在高度并发的 GPU 上进行蛮力搜索。本章中给出了这一算法的具体思路和流程。最后本章阐述了对 ICP 算法进行具体实现时在性能上一些考虑。本课题实现的 ICP 算法并不是完全在 GPU 上进行计算, 而是将一部分计算转移到了 CPU 上。本章阐述这么实现的理由以及具体计算步骤再 GPU 和 CPU 上的分布。

第五章 从三维点云提取表面

场景重建的最后一个步骤也是最重要的一个步骤是将散点变成表面模型。一般性，表面模型是由大量的三角面片构成。因此生成表面的算法主要要解决两个问题：

- (1) 模型有多少顶点，以及每个顶点的位置。
- (2) 顶点集的拓扑结构。即需要将哪些顶点连结起来形成三角面片。

在解决上述两个问题的基础上，该算法还应尽可能降低点云中的噪声的干扰，保留细节部分，以构建出和真实场景尽可能相近的场景模型。

由于原始点云中存在噪声，场景模型并不能直接使用原始点集作为顶点集。又虽然每组点云具有高度结构化（拓扑信息可以从深度图中的像素排列推导而来），多组点集之间却没有任何结构上的关系，因此顶点集的拓扑结构也不能直接从点云中获得。

5.1 从三维点云提取表面的基本步骤

表面提取算法的基本思想是采用某种空间结构将无结构的点云组织起来，并建立一个数学模型解出顶点的位置，最后结合两者提取拓扑信息。

最简单的三维空间结构是网格划分，又被称为体素（Voxel），例如[6]提出的方法所使用的便是这种结构。采用这种结构的优点是结构简单、实现方便，但时间和空间代价高昂。三维模型的点云具有高度集中性，其点云的覆盖范围往往远小于整个空间的体积，造成网格划分的大量浪费。

八叉树是另一种空间划分的常用结构（四叉树是其二维版本），采用层次化组织空间可以避免上述浪费（图 13）。[20]提出基于泊松方程的表面重建算法便是采用八叉树结构。[21]提出了该算法的 GPU 实现，其中提出了基于 GPU 的八叉树结构构建算法。

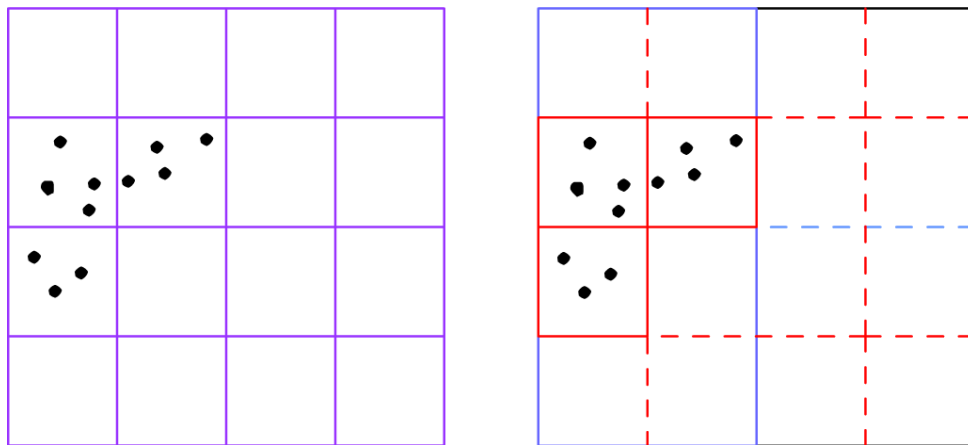


图 13 二维平面下网格划分和四叉树对比

左图：网格划分形成 16 个节点，但只有 3 个节点包含点集数据；

右图：四叉树仅需 6 个节点即可包含所有有效区域

在采用的数学模型方面，[6, 8, 22]等采用有向距离函数，而[20]等采用泊松方程。有向距离函数基于局部数据计算表面，而泊松方程基于整体数据计算表面，因此泊松方程较之有向距离函数重建结果更加平滑完整^[20]。但是泊松方程计算所需时间比有向距离函数长许多，也不宜处理含有多个分离模型以及部分模型的重建。

从本质上来说，上述几种方法的计算结果都是一个三维空间映射到实数域的函数，其某

个值所对应的空间集合是所求的模型表面,该空间集合被称为等值面。根据采用的数学模型不同,表征模型表面的值也是不同的,一般有向距离函数采用零值面作为所求模型的表面,但是也有通过最小二乘法计算该值的,例如[22]和[20]。

数学模型的构建和求解均被离散化到某种空间结构下,在此基础上构建顶点和三角面片。[20-22]均采用[23]中提出的 Marching Cubes 算法提取等值面并构建顶点集的拓扑结构。该算法基于网格划分结构的立方体单元,根据立方体的 8 个顶点分别在模型表面的外侧或内侧对立方体分类,并对每一类指定了在该立方体单元中的模型表面的拓扑结构。

本课题所实现的表面提取算法采用八叉树结构组织点云和划分空间,基于 GPU 的八叉树构建算法的实现参照了[21]中所提出的设计与实现方法;使用有向距离函数模型,有向距离函数的定义和计算借鉴了[22]中的思想;最后采用基于八叉树的 Marching Cubes 方法构建三角面片。

5.2 基于 GPU 的八叉树构建算法

本课题参照[21]中所述设计并实现了基于 GPU 的八叉树构建算法。八叉树结构在计算机图形学等领域有着广泛应用,但是基于串行思想设计的八叉树构建和查找算法并不能在 GPU 上得到有效计算。为了使八叉树构建和查找能在 GPU 上高效执行,[21]采用了广度优先的构建算法,并使用预先计算好的邻居查找表查找节点的邻居,将带有数据依赖而无法并行执行的操作降到最少。

5.2.1 八叉树的数据结构定义

本课题所实现的八叉树包含三类数据结构:节点、顶点和边。其中节点是八叉树的核心数据结构,顶点和边均为由于表面提取算法需要而增加的数据结构。[21]中还包含了面,但由于本课题所实现的表面提取算法并不需要建立关于面的结构信息,同时为了节省空间,本课题构建的八叉树并不包含面结构。

节点: 八叉树的每个节点包含有 1 个父节点(根节点没有父节点)和 0 到 8 个子节点,每个节点均覆盖三维空间中的某个区域,包含 1 到多个点云中的点。由于点云会事先根据其所属的节点进行排序,因此节点只需记录其包含的点集的起始位置和数量。为了构建和查找方便,将每个节点按照从根节点到该节点的访问路径进行编号。由于每一个节点最多只有 8 个子节点,可以用 3 个二进制位 xyz 表示子节点在其父节点中的序号,规定若子节点中心的 x 轴坐标小于父节点中心的 x 轴坐标,则 x 位为 0,否则为 1。 y 和 z 的规则同。将从根节点到当前节点的每一层序号连接在一起形成一个二进制串,可以唯一标识该节点在八叉树中的位置以及其在三维空间中的位置。除了上述信息以外,为了后续算法执行的效率,一个节点还包括其 27 个邻居节点、8 个顶点和 12 个边的地址。

顶点: 一个顶点最多能被 8 个节点分享,因此一个顶点包含有 1 到 8 个节点地址。根据表面提取算法,每个顶点包含一个有向距离值和它在三维空间中的坐标。另外为了将点云中的颜色和法向量传递到最终的表面模型上,每个顶点还包含有颜色和法向量等信息。

边: 一条边最多能被 4 个节点分享,同时一条边有两个顶点,因此一条边包含有 1 到 4 个节点地址和 2 个顶点地址。为了关联最终生成的模型顶点,一条边还包含 1 个最终生成的模型顶点地址。

上述数据结构中的链接数据均按照位置坐标先 z 后 y 最后 x 、先小后大的顺序排列。为了简单起见,给出按照这一规则在二维平面下的四叉树中各结构的顺序关系如图 14,八叉树可依此很容易地推广。

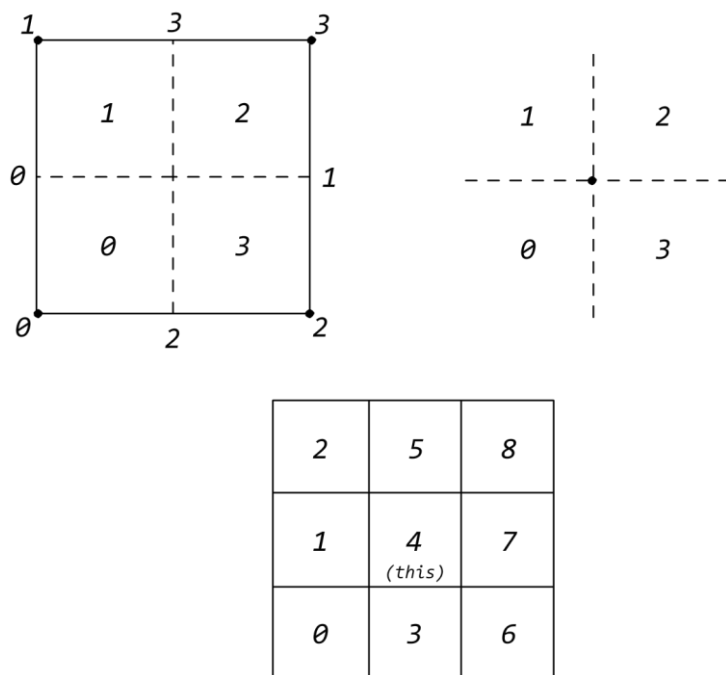


图 14 四叉树各结构之间的顺序关系（八叉树可依此推广）

左上图：节点结构中的子节点、顶点和边序；右上图：顶点结构中的节点序；

下图：节点结构中的邻居节点序（4 号为它自己）

5.2.2 邻居查找表和邻居查找算法

由于八叉树的邻居查找算法需要回溯遍历，这是一个无法并行执行的算法，因此在 GPU 上进行动态的邻居查找十分低效。但是注意到八叉树节点的邻居的局部性，其邻居事实上可以通过预先计算的查找表找到，这样可以极大提高邻居查找的效率。通过观察可以发现，八叉树节点的邻居仅分布于兄弟节点和父节点的邻居节点的子节点中，因此一个节点可以通过父节点的邻居信息快速地找到它的某个邻居（图 15）。

邻居查找表包含两张查找表，均为 8 行 27 列。第一张记录了当前节点 *this*（其在父节点中的序号是 *i*）的邻居 *neigh[j]* 的父节点在当前节点的父节点中的序号 *parentNeigh[i][j]*。第二张表记录了当前节点 *this*（其在父节点中的序号是 *i*）的邻居 *neigh[j]* 在其父节点中的序号 *childNeigh[i][j]*。

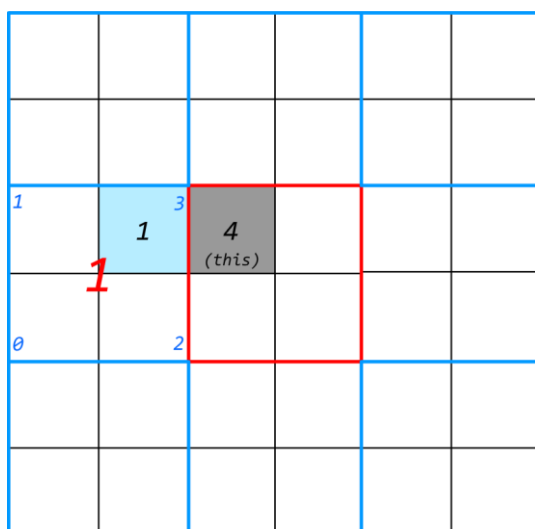


图 15 当且节点仅需根据父节点的邻居信息就能够快速找到邻居

因此邻居查找包含两个步骤：先根据 $parentNeigh[i][j]$ 找到某个邻居节点的父节点，再根据 $childNeigh[i][j]$ 从这个父节点中定位到具体的子节点。

5.2.3 八叉树节点、顶点和边的构建算法

由于八叉树的同层级节点之间不存在任何数据依赖关系，因此采用广度优先构建算法的每一层节点的构建都能够并行执行。八叉树构建主要分为三个步骤：构建节点树、连接邻居节点、生成顶点和边信息。

构建节点树采用自下而上的构建顺序，将信息层层上传构建节点。首先根据点云生成最底层节点。并行计算点云中的每个点所处的最底层节点的编号后对编号进行排序，去除重复编号即可得到最底层节点的编号列表以及每个节点包含的点集。对于其他层级的节点，由于具有相同父节点的节点其去掉最后三位后的编号是相等的，因此只要对下层的编号列表中的编号去掉后三位后执行去除重复编号操作即可得到当前层的编号列表。

需要注意的一点是，在连接父子节点时，为了便于找到某个节点的子节点，[21]在生成节点数组的时候将节点根据父节点分组分散存放，为每组都保留 8 个节点位置，并将节点按照其在父节点中的序号填入组内相应的位置中，这样在父节点寻找子节点的时候只需要根据父节点本身在编号列表中的位置即可在下层节点数组中找到对应的组，从而找到所有它的子节点。但由于节点结构包含大量信息，将其分散存放需要占用大量的空间，因此本课题在实现时采用了间接寻址数组以减少空间的使用，即节点本身在数组中仍旧紧密排列，使用额外的数组记录节点的地址，并将这些地址分组分散排列。这样在父节点寻找子节点的过程中仅需多一个从寻址数组中获得节点本身的地址的步骤即可访问子节点，而空间占用减少了 80% 以上（图 16）。

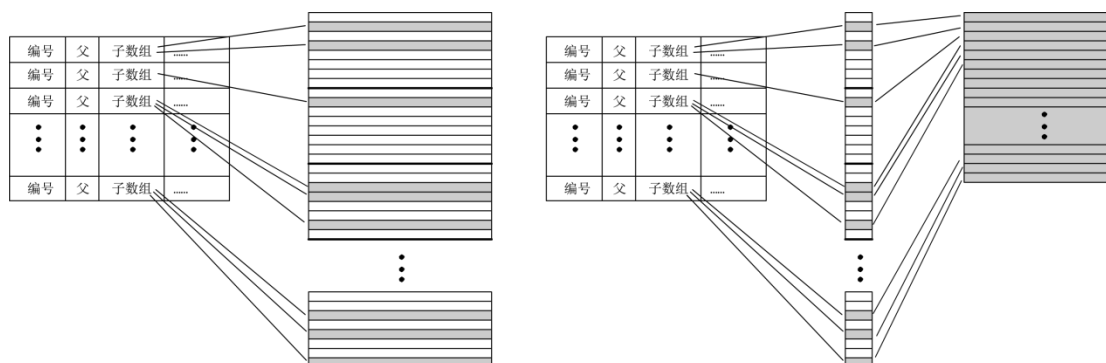


图 16 相比直接将节点分散存放，分散存放节点的地址能节省大量空间

由于下层节点查找邻居需要依赖上层节点的邻居信息，因此连接邻居节点采用自上而下的执行顺序。对于每一层的所有节点，根据上文 5.2.2 节中描述的查找算法并行查找其 27 个邻居并记录在其邻居地址数组中。

在形成节点树后，需要生成所有最底层节点的顶点和边。一个最直接的方法是为每个节点都生成 6 个顶点和 12 条边，很显然生成的结果中会产生大量冗余。这一部分[21]中并没有给出详细的生成算法。本课题设计并实现的方法是根据顶点在节点中的序号顺序创建顶点。即在每一次并行创建顶点的过程中，仅创建当前指定序号的顶点，并将所有共享该顶点的节点与该顶点连接。若该序号的顶点已经存在则不执行任何操作。在该算法中，共享某序顶点的邻居节点也是通过查表法得到。执行创建操作 8 次后即可完整而没有重复地创建所有节点的顶点。算法 5 显示了顶点生成算法的流程。边的生成算法和顶点类似。

算法 5 八叉树节点的顶点生成算法

输入：八叉树最底层节点数组 $\{n\}$ ，共享顶点邻居表 $table[8][8]$
输出：顶点数组 $\{v\}$ ，包含顶点信息的八叉树最底层节点数组 $\{n\}$
过程：
顶点序 i 从 0 到 8：
 对于每个 $\{n\}$ 中的节点 n ，并行执行：
 倘若 $n.vertex[i]$ 不存在则：
 创建顶点 v 并存入顶点数组中；
 遍历 $table[i]$ 中的 8 个元素 $table[i][j]$ ，找到对应的节点 n' ；
 倘若 n' 存在则：
 $n'.vertex[7-j] = v, v.node[j] = n'$;
结束

5.3 基于有向距离函数的等值面计算

如上文 5.1 节中所述，在选取了合适的空间划分结构后，需要选择一个数学模型计算模型表面在空间中的区域。本课题采用有向距离函数的思想，其有向距离的定义类似[22]中所使用的定义。然后针对上文 5.2.3 节中构建的所有顶点计算有向距离。

在本课题所实现的表面提取算法中，对于空间中的某点 P 和点云中的某点 V 有向距离被定义为向量 VP 在 V 的法向量上的投影。而空间中某点 P 的有向距离函数值被定义为 P 到点云中所有与 P 邻近的点的有向距离之和。由于在表面上距离表面某点很近的区域内所有点与它的有向距离应为 0，因此表面模型近似于有向距离函数的零值面。

邻近范围的选取会影响到最终表面模型的生成效果。邻近范围越大，表面模型越抗噪也越粗略；而邻近范围越小，表面模型中的细节越多也越容易受到噪声的干扰。在本课题的实现中，选取邻近范围共享某个顶点的 8 个节点，主要是考虑到这样的范围可以直接通过顶点的节点数组获得，而其大小也在合适的范围内。

在每个顶点都具有了有向距离值以后，等值面与所有的边的交点通过插值得到。容易发现，当且仅当某边的两个顶点的有向距离值符号相反时，它与等值面有交点。这些交点是对等值面进行了基于八叉树最底层节点的离散化。由于等值面就是要求的表面，因此这些顶点就是最终生成的表面模型的顶点。

5.4 Marching Cubes 表面提取算法

Marching Cubes 算法^[23]是基于网格划分的表面提取算法，提出至今已经被许多研究三维重建技术的论文引用。其基本思想是在每个网格中的小立方体单元中用平面结构近似表面。对于立方体上的 8 个顶点，根据它们中哪些点在模型的内部或外部可以将立方体分为 256 类，又由于内外对称、立方体旋转对称，这 256 个种类可以规约到 15 个基本种类^[23]。然后它分别给出了每个种类的立方体可以形成的三角面片结构（图 17）。Marching Cubes 算法的基本流程对于网格中的每个单元，判断其 6 个顶点的值（在三维模型的外部或内部），以这 6 个顶点的值组成索引查表获得该单元需要形成的一系列三角形顶点索引。

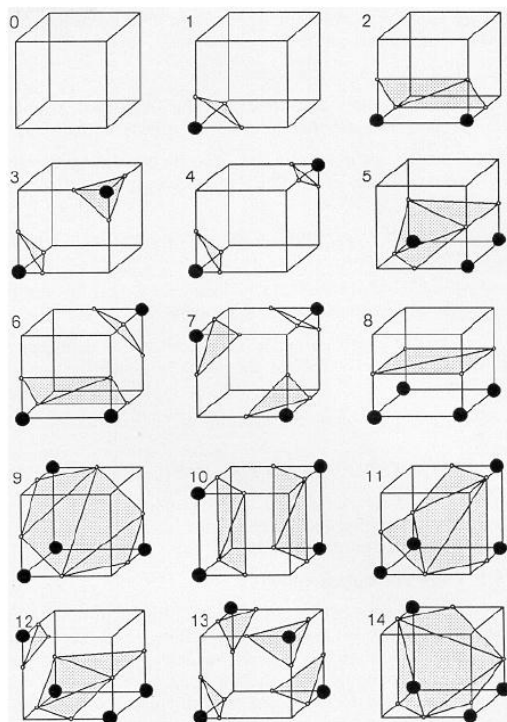


图 17-15 类立方体可以形成的三角面片（摘自[23]中的插图）

5.4.1 拓扑结构表的构建

根据上文 5.3 所述，本课题中的顶点值分为正值和负值两种，正值表示该顶点在三维模型的外部，负值表示该顶点在三维模型的内部。Marching Cubes 算法的核心就是一张记录了所有类型的立方体所能形成的表面拓扑结构的表。虽然[23]中给出了 14 种基本类型的拓扑结构，但在实际实现过程中要将这 14 种基本类型应用各种对称性扩展为完整的 256 种类型是十分困难的。

通过研究立方体所形成的拓扑结构和它的顶点值之间关系，本课题设计并实现了一种拓扑结构表的生成算法。其构造的思想是正值点扩展规则，即假定正值区域从各个正值顶点开始逐步扩展，吞掉所有正值边（两个顶点均为正值的边）后，正值区域与负值区域的交界面即为所求的表面。

可以先从二维平面的角度考虑该问题。如图 18 所示，一个矩形区域的边界上含有 3 个正值顶点 0,1,2 和一个负值顶点 3。根据正值点扩展规则，正负区域的交界线初始情况如图 18 的中图所示。经过扩展吞并所有正值边后，正负区域的交界线如图 18 的右图所示。这一交界线即为最后所需求的结果。从这一例子中可以看出，所谓扩展的过程就是将折线段以其首尾相连的直线所代替，以达到最大的扩展面积。

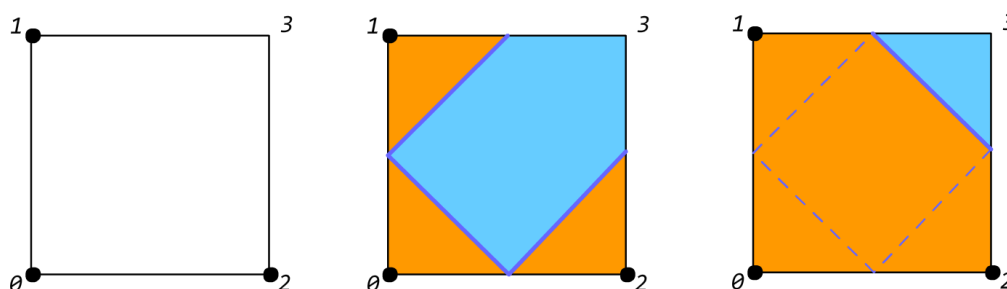


图 18 二维矩形区域中的根据正值扩展规则求解边界线的过程
黑点标记表示该顶点为正值

在三维情况下，只需对立方体 6 个面分别求解，并将结果连接在一起即可得到交界面与立方体各面的交线（图 19）。然后将其内部分割为一个一个三角形即可。

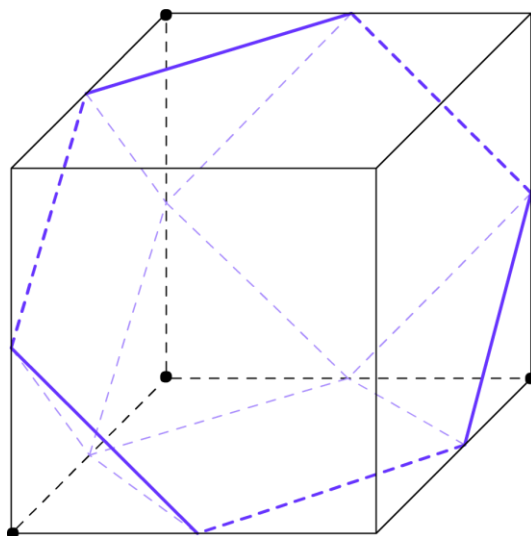


图 19 在三维立方体中通过对立方体各个面进行二维求解得到界面的边界线
黑点标记表示该顶点为正值

5.4.2 基于八叉树的 Marching Cubes 算法

基于八叉树的 Marching Cubes 算法大体上和传统的 Marching Cubes 算法没什么不同。本课题中实现的表面提取算法基于八叉树的最底层节点计算有向距离函数并提取等值面。在得到所有顶点的有向距离值后，对每个节点根据它的 8 个顶点的值形成的索引查表建立三角面片并添加到三角面片数组中。与传统的 Marching Cubes 算法不同的是，由于八叉树的最底层节点仅存在于有原始点云中的点的区域，因此会有部分三角面片因节点的缺失而无法构建，导致最终生成的表面中包含许多小洞（图 20）。

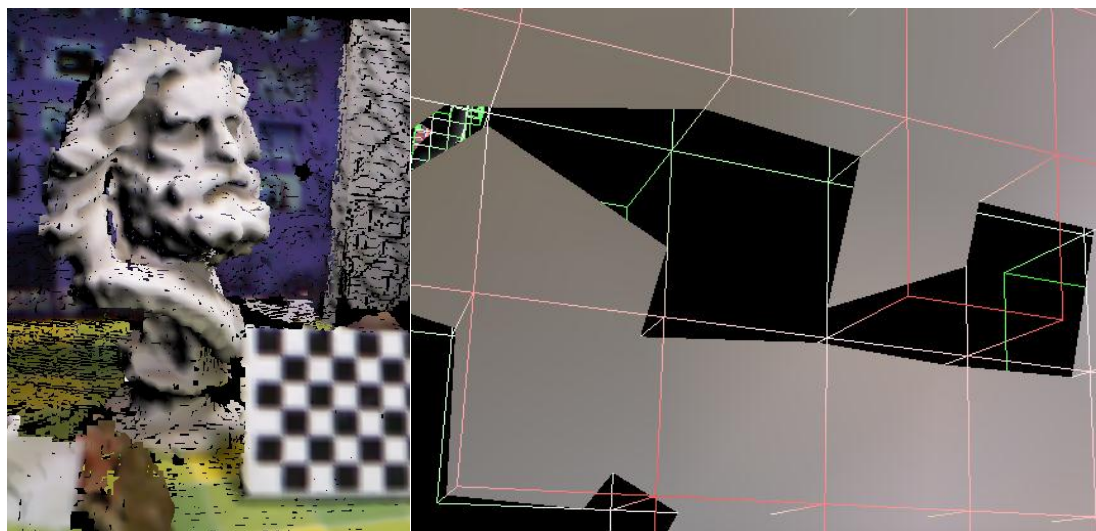


图 20 由于节点的缺失，单纯执行传统的 Marching Cubes 算法会产生许多小洞

为了解决这一问题，本课题提出了构造伪节点的方法。其基本思路是对于所有在面 f 上与表面存在交线的节点 n ，构造一个伪节点 n_{fake} ，与 n 在 f 上邻接，并将在结构关系上属于 n_{fake} 的顶点和边赋给 n_{fake} 。然后再对所有伪节点执行 Marching Cubes 算法生成三角面片，便可补上这些空洞（图 21）。

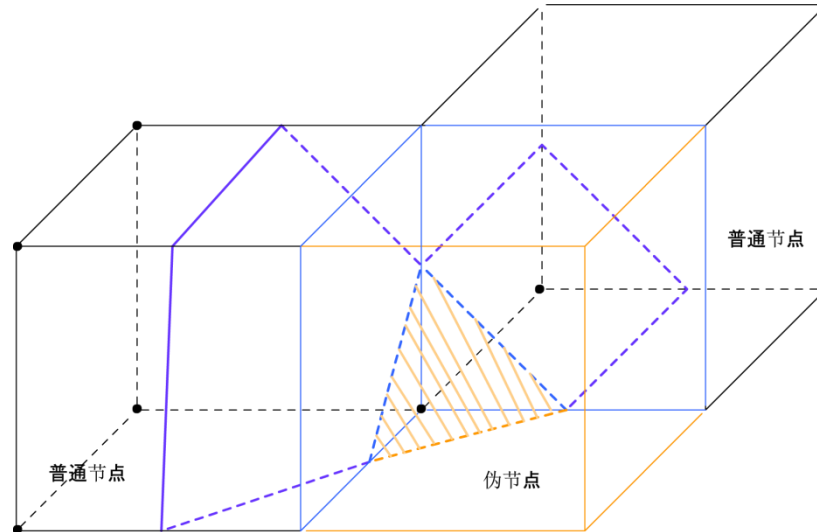


图 21 通过构造伪节点可以填补由于节点缺失造成的空洞（如图中画阴影的三角形）

另外由伪节点的构造而引出的一个问题是，由于伪节点中不包含任何原始点云中的点，因此由于伪节点而新增的顶点（可以称其为伪顶点）并不存在一个有向距离值（可以假定其为无穷大）。一个无穷大的值并不属于正值也不属于负值，这导致无法明确识别该立方体的类型。因此，本算法采用就近原则，将一个无穷大值的符号定义为该顶点的三个邻接顶点中非无穷大值的符号（如图 22 的左图）。若三个邻接顶点中存在不同的符号（如图 22 的中图）或三个邻接顶点均为伪顶点（如图 22 的右图），则抛弃该伪节点，不生成任何三角面片。

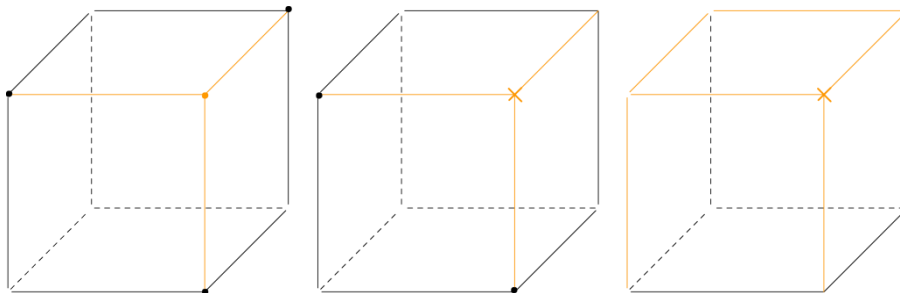


图 22 就近原则处理无穷大值的符号

左图：无穷大值点的三个相邻顶点符号相同，无穷大值的符号跟随相邻顶点的符号；

中图：无穷大值点的三个相邻顶点符号不同，抛弃该伪节点；

右图：无穷大值点的三个相邻顶点均为伪顶点，抛弃该伪节点

而在生成三角面片方面，由于为伪顶点赋予了符号，Marching Cubes 生成的三角面片中可能会存在顶点在伪边上的情况。对于这种情况，仅需要将该三角面片删除即可。

使用构造伪节点的方法后重建结果中含有的空洞明显减少了（图 23）。

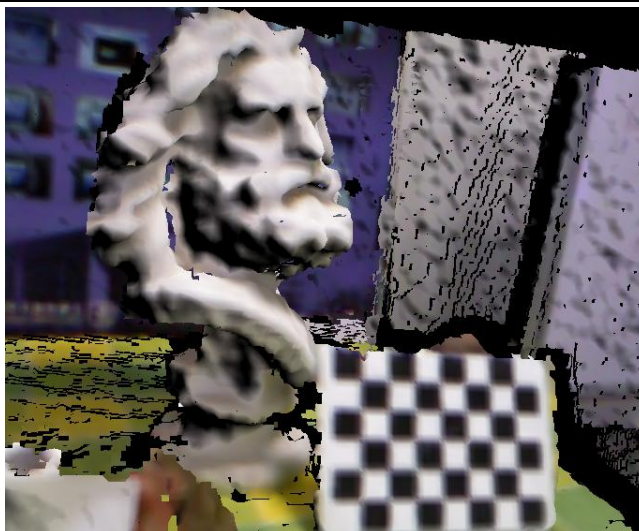


图 23 使用构造伪节点的方法改进 Marching Cubes 算法后的重建结果

5.5 本章小结

本章主要阐述了从点云中提取场景模型表面的算法。本章首先提出了表面提取算法主要需要解决的问题是生成顶点集以及顶点之间的拓扑结构。紧接着阐述了解决这一问题的基本思想是利用合适空间结构和数学模型求解表面区域的离散解并组织成三角面片，并依据这一框架介绍了下的各种的表面提取算法及其优缺点，结合这些优缺点，本课题选取了基于八叉树和有向距离函数的表面提取算法，并使用 Marching Cubes 构造模型顶点之间的拓扑结构。然后本章分节依次详细阐述了表面提取算法中每个主要步骤的设计与实现细节，并针对一些参考算法中存在的问题提出了自己的解决方案。由于本课题实现的三维重建需满足实时的性能要求，因此这些步骤所使用的算法均针对 GPU 进行了改进设计，尽可能压缩数据依赖关系使得算法具有很高的并行度。

第六章 实验结果和性能

6.1 对人物雕像的重建实验结果

6.2 对办公环境的重建实验结果

6.3 性能分析和调优

6.4 本章小结

第七章 总结

7.1 研究成果和问题

本课题的最终成果是设计并实现了基于多个 Kinect 的实时动态场景重建技术，其中涉及到的所有步骤除棋盘格标定和 ICP 算法的变换计算外都是针对 GPU 进行了设计并在 GPU 上得到了实现，性能基本满足实时要求。其包含的主要研究成果有以下几点：

- (1) 提出了基于投影关系的快速最近邻搜索算法，充分利用从深度图生成的点云所继承的高度结构化特征缩小搜索范围。
- (2) 参照[21]中所提出的算法实现了基于 GPU 的八叉树构建算法，并对参照算法中没有阐明的八叉树顶点和边的生成算法提出了自己的解决方案。
- (3) 提出了基于正值扩展规则构建 Marching Cubes 算法中所需的拓扑结构表。并对基于八叉树的 Marching Cubes 算法存在问题提出了自己的解决方案。

根据上文第六章所展示的实验结果来看，本课题所实现的重建技术的，但是重建的效果不如预期。从重建的结果来看，离 Kinect 较近的区域重建效果较好，形成的模型表面空洞较少，但是离 Kinect 较远的区域重建的效果离预期效果相差较大。主要原因有以下几点：

- (1) 多个 Kinect 同时对同一区域进行拍摄时会相互干扰。由于 Kinect 是通过投射红外结构光并捕获其反射光的结构分布来获取场景的深度信息，不同的 Kinect 投射在同一区域的红外光会相互叠加，干扰反射光的结构分布。当两台 Kinect 同时工作时，它们所覆盖的重叠区域中的噪声和不可识别点显著增加。
- (2) 由于执行性能和空间限制，本课题所实现的重建只能基于 320×240 分辨率的深度和彩色图像，这导致点在空间中的分布，尤其是在较远区域的分布比较稀疏。在较远的区域，其平均间距已经大于算法中所构建的八叉树最底层节点的边长，导致该区域的重建结果含有大量空洞。

另外，重建模型的纹理效果较差，也是影响模型重建效果的重要因素之一。目前本课题并没有针对色彩纹理做过多的处理，这会导致不同 Kinect 捕获到的色调和亮度各不相同的纹理被简单地融合在一起。另外，本课题目前仅是简单地采用领域平均法计算得到模型顶点的颜色和法向量，这会使纹理产生非常模糊的效果，影响了其真实感。

7.2 下一步工作

本课题在未来将重点研究如何在保证性能的基础上进一步改进重建效果。根据上文中提出的一些目前的重建算法存在的问题，未来将考虑从如下三个方面改进算法：

- (1) 优化空间使用率。重建算法目前只能接受 320×240 的分辨率其主要原因在于构建八叉树所需的内存过大。由于动态分配 GPU 缓存资源的开销极大，一般都是在初始化时就创建了足够大的缓存。例如基于极端情况下每个八叉树最底层的节点可能只包含一个点云中的点，因此目前为八叉树最底层节点分配的内存与分辨率和 Kinect 的个数的乘积一致，这一数量是极大的，并且在输入为 640×480 的分辨率时数量还将翻 3 倍。但是事实上由于点集分布的集中性，上述极端情况是不可能出现的。另外由于分辨率的增加只会导致点云的密度增加，并不会增加许多覆盖区域，因此实际上分辨率的增加并不会导致最底层节点的大幅增加。诸如此类的待优化的部分在目前的实现中还有许多。未来的工作将

会通过实验确定一个较为精确的上界以减少空间浪费。

- (2) 改进表面提取算法。未来的工作将深入研究更多的应用在表面重建中的数学模型，以期提出一个更好的数学模型减少空洞和噪声敏感度并提升重建的模型的细节。另外还将考虑借鉴基于网格划分的空洞填补算法设计基于八叉树的空洞填补算法，以弥补由于多个 Kinect 之间的干扰造成的数据空洞。
- (3) 引入纹理色彩分析算法。未来的工作还将考虑研究色调补偿和表面材质提取等方面的内容，并改进模型纹理的生成方法，提高模型纹理的真实感。

另一方面，未来的工作还将进一步探究本课题所研究的动态场景重建技术的应用价值和前景，并设计多种增强现实应用以挖掘、验证和展示该技术所蕴含的价值。

参考文献

- [1] O. Bimber and R. Raskar, *Spatial Augmented Reality: Merging Real and Virtual Worlds*: A. K. Peters, Ltd., 2005.
- [2] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs, "The office of the future: a unified approach to image-based modeling and spatially immersive displays," presented at the Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998.
- [3] P. Lincoln, G. Welch, A. Nashel, A. Ilie, A. State, and H. Fuchs, "Animatronic Shader Lamps Avatars," in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, 2009, pp. 27-33.
- [4] P. Lincoln, A. Nashel, A. Ilie, H. Towles, G. Welch, and H. Fuchs, "Multi-view lenticular display for group teleconferencing," presented at the Proceedings of the 2nd International Conference on Immersive Telecommunications, Berkeley, California, 2009.
- [5] S. Deng and B. Yuan, "A method for 3D surface reconstruction from range images," in *Speech, Image Processing and Neural Networks, 1994. Proceedings, ISSIPNN '94., 1994 International Symposium on*, 1994, pp. 429-432 vol.2.
- [6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," presented at the Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.
- [7] C. Schutz, T. Jost, and H. Hugli, "Free-form 3D object reconstruction from range images," in *Virtual Systems and MultiMedia, 1997. VSMM '97. Proceedings., International Conference on*, 1997, pp. 69-70.
- [8] R. T. Whitaker, "A Level-Set Approach to 3D Reconstruction from Range Data," *Int. J. Comput. Vision*, vol. 29, pp. 203-231, 1998.
- [9] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, D. Molyneaux, S. Hodges, P. Kohli, J. Shotton, A. J. Davison, and A. Fitzgibbon, "KinectFusion: real-time dynamic 3D surface reconstruction and interaction," presented at the ACM SIGGRAPH 2011 Talks, Vancouver, British Columbia, Canada, 2011.
- [10] 杨敏 and 沈春林, "基于场景几何约束未标定两视图的三维模型重建," *中国图象图形学报 A 辑*, vol. 8, p. 5, 2003.
- [11] F. Remondino, S. F. El-Hakim, A. Gruen, and Z. Li, "Turning images into 3-D models," *Signal Processing Magazine, IEEE*, vol. 25, pp. 55-65, 2008.
- [12] J. Wei and L. Jian, "Panoramic 3D Reconstruction by Fusing Color Intensity and Laser Range Data," in *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*, 2006, pp. 947-953.
- [13] P. J. Besl and R. C. Jain, "Three-dimensional object recognition," *ACM Computing Surveys (CSUR)*, vol. 17, pp. 75-145, 1985.
- [14] D. Neumann, F. Lugauer, S. Bauer, J. Wasza, and J. Hornegger, "Real-time RGB-D mapping and 3-D modeling on the GPU using the random ball cover data structure," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, 2011, pp.

- 1161-1167.
- [15] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239-256, 1992.
 - [16] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 1330-1334, 2000.
 - [17] L. Cayton, "A nearest neighbor data structure for graphics hardware," in *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2010.
 - [18] T. Foley and J. Sugerman, "KD-tree acceleration structures for a GPU raytracer," presented at the Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Los Angeles, California, 2005.
 - [19] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, pp. 629-642, 1987.
 - [20] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," presented at the Proceedings of the fourth Eurographics symposium on Geometry processing, Cagliari, Sardinia, Italy, 2006.
 - [21] K. Zhou, M. Gong, X. Huang, and B. Guo, "Highly parallel surface reconstruction," *Microsoft Research Asia*, 2008.
 - [22] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," University of Washington, 1994.
 - [23] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 163-169, 1987.

谢辞

在本课题的进展过程中,包括导师杨旭波老师在内的一些老师和同学给我提供了许多帮助和支持。在此我要表达对你们深深的谢意。

首先我要感谢我的导师杨旭波老师。在整个课题进展的过程中,他对我严格要求,每周都前来了解课题情况,以了解课题是否进展顺利。同时他也为这个课题提出了许多有价值的指导意见。

其次我要感谢曹阳和何贞毅两位同学。在我遇到困难时,他们给了我莫大的支持和帮助。

最后我还要感谢我的学院和学校,让我在一个良好的环境中完成了本课题。学校所提供的大量学术资源库是整个课题得以顺利完成的关键。

REAL-TIME DYNAMIC INDOOR SCENE RECONSTRUCTION BASED ON MULTIPLE KINECT

Scene reconstruction is one of the underlying technologies in computer vision. The real-time dynamic scene reconstruction plays an important role in the application of augment reality and robot control. For example, we can apply this technology to our office to make it digitalized. Thus all natural interaction in the scene can be handled by computer. Also the office scene reconstructed can be combined with a remote scene to implement a local-like telecommunication. However, restricted by the technologies of hardware and software, real-time dynamic scene reconstruction has never been implemented. With Kinect published, restriction on hardware has been solved. This subject is to study how to implement the real-time dynamic scene reconstruction based on Kinect.

The hardware system is simple. Multiple Kinect devices are fastened around the scene to capture the scene from different views at the same time. Fixed Kinect devices can make reconstruction simple and robust. After the system is deployed, all kinect devices need to be calibrated using chessboard. The calibration contains intrinsic parameter calibration and extrinsic parameter calibration. This process need to be done only once.

The reconstruction method follows a regular framework. Firstly, depth and RGB images are obtained from Kinect. Secondly, image preprocessing is applied on the depth and RGB images to reduce noise. Then, each pair of depth and RGB image is used to generate point cloud in 3D space. After that, registration between point clouds is done to optimize the extrinsic parameters of Kinect devices and minimize the transform error between point clouds. The last step is extracting surface from point clouds. To gain a real-time performance, all processes a designed and implemented according to the GPU architecture, that is, designed to be highly parallel.

Key algorithms implemented in this subject includes image denoising in image preprocessing process, projection-based nearest neighbor search and CPU-GPU combined transform estimation in registration process, GPU based octree construction and GPU based Marching Cubes in surface extraction process.

The denoising method for depth image is a variation of Gauss filter. Unlike ordinary images, the borders of depth images represent the boundary of different objects in the scene. So keeping borders clear in denoising is important. The main idea is to mask the neighbors of a pixel according to the difference between two adjacent pixels. If the difference is larger than the specified threshold, it should be seen as a border. The neighborhood of a pixel should not across the border, thus only neighbors on the same face with the pixel will contribute to the pixel.

The projection-based nearest neighbor search is the first step of Iterative Closest Point algorithm. Its essence is brute force search. Implemented on GPU, brute force search has almost the same performance as a well designed search algorithm implemented on CPU, such as KD-tree. The main idea of the projection-based nearest neighbor search is to restrict the search area to a small region. Note that if two points are near in 3D space, their projected points in 2D plane are also near. So if a source point is projected onto target image plane, its nearest neighbor in target point cloud can be found in a small rectangle area whose center is the projected source point on

the target plane. This restriction can make a brute force search more efficient on GPU.

The transform estimation is the second step of Iterative Closest Point algorithm. It needs simple computing on large scales of data as well as complex computing on small data such as a four dimension matrix. The latter has low degree of parallelism and will be slow if run on GPU. So to gain the highest performance, CPU and GPU should divide the work. Tasks such as computing mass centers of points should be done on GPU because it can be run highly parallel, while other tasks such as computing the eigen values and eigen vectors of a 4×4 matrix should be done on CPU to make use of its high speed.

The octree structure is used to divide the space, organize the point cloud and solve the discrete solution of signed distance function in the surface extraction process. Unlike the traditional construction method, the GPU-based construction of octree should be designed as a highly parallel algorithm. The construction of nodes goes down-to-up and is in parallel for each level. A key is assigned for each node, which is the path from root to it. Because the paths of siblings are only different in the last level, it's easy to construct higher level nodes from the lower ones. Another optimization is the use of a look up table when connect links between neighbors. While the traditional neighbor search need to trace back several levels, the search using the look up table only need to visit the parent or the parent's neighbors. The look up table contains the parent's neighbor which contains the neighbor and the index of the neighbor in its parent. Vertices and edges of the nodes in lowest level are also constructed for later processes.

The signed distance function model is used in the surface extraction process. The signed distance is defined as the projected distance of a point in space to a point on the surface onto the normal of the point on the surface. Thus the value of the signed distance function at a point is calculated as the average of signed distances of the point to near points on the surface. Because the values of signed distance function at points on surface are all zero, the zero-set of the function can be presented as the surface extracted. The discretization is done by calculating the value of signed distance function at every vertex of octree nodes in lowest level. Vertices on the surface are generated by doing a linear interpolation between two vertices of an edge.

Marching Cubes method is used in the last step of surface extraction, which is to generate triangles between vertices on the surface. As described above, each vertex has a signed value, positive if outside the model and negative if inside it. Thus there are 256 different node types according to the sign of values of nodes' vertices. Nodes with the same type generate the same topology between the model vertices on its edges. A table is used to record all types and all nodes in lowest level are processed in parallel to generate triangles by looking up the table. The table is constructed automatically by applying a method called positive areas expanding. The idea of positive areas expanding is to set positive areas at every positive corners firstly and expand the positive areas as much as possible. Problems in doing Marching Cubes on octree is that triangles in empty area cannot be generated because the absent of nodes. This problem is solved by generating fake nodes in those area and then doing Marching Cubes on the fake nodes.

The scene reconstruction is implemented in C++ and using DirectX's compute shader to implement the algorithms on GPU. Experiments are done on a laptop with Intel® Core™ i7-2570QM, 8.00 GB memory and AMD Radeon™ HD 6770M. Experiments include reconstruction of a statue and reconstruction of an office scene. The statue reconstructed is fine but the office scene reconstructed is undesirable. When the distance of objects is larger than 1.0 m, the models reconstructed have too many holes. The interference between Kinects is also found in

experiments. A reconstruction process needs about 300ms, which led to the performance of 3 fps.

The experimental results prove that, the reconstruction process implemented meet the requirements of the real-time dynamic scene reconstruction. But the result and the efficiency still need to be improved. Low memory usage in constructing octree is a bottleneck, which currently restricts the RGB and depth images captured by Kinect to 320×240 . The outcomes of the subject showed it possible to reconstruct real-time dynamic scene based on the Kinect and GPU parallel computing.