Kinect数据修复方法研究
及其在立体视频中的应用

作者姓名 王蓬金 **导师姓名、职称** 武筱林教授

一级学科 电子科学与技术 **二级学科** 电路与系统

申请学位类别 工学硕士 **提交学位论文日期** 2014 年 11 月

学校代码 10701　　学　号 1202120904
分 类 号 TP39　　密　级 公开

西安电子科技大学

硕士学位论文

Kinect 数据修复方法研究及其在立体视频中的应用

作者姓名：王蓬金

一级学科：电子科学与技术

二级学科：电路与系统

学位类别：工学硕士

指导教师姓名、职称：武筱林教授

提交日期：2014 年 11 月

A Study of Kinect Data Repair Method and Application on Stereo Video

A thesis submitted to
XIDIAN UNIVERSITY
in partial fulfillment of the requirements
for the degree of Master
in Electronic Science and Technology

By
Wang Pengjin
Supervisor: Prof. Wu Xiaolin
November 2014

西安电子科技大学

学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：_____　　日　　期：_____

西安电子科技大学

关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究成果再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于保密，在___年解密后适用本授权书。

本人签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘要

远程视频的出现极大地方便了现代人们之间的相互沟通，而立体视觉技术的发展使得视频交流的效果有了很大的上升空间。在 Kinect 的革命性发明之后，深度获取就不那么昂贵了，由 Kinect 的数据构建立体模型，进而实现立体视频会议系统是一种切实可行的方法。但是对于计算机视觉应用来说，由 Kinect 捕获的深度数据需要额外的处理来填补丢失的部分，进一步讲，使用 Kinect 数据重建三维模型过程中，由于视角的变换，彩色数据的缺失也需要在最终显示前进行修补。本文首先研究了 Kinect 构建立体数据的方法；然后针对立体数据中各种原因的缺失进行修复，分别用结构导向的框架修复了深度数据以及用快速行进的方法修复了彩色数据；最后设计了一套立体视频会议系统，并将修复算法应用其中，同时研究了多种加速策略来提高立体视频的播放帧率。

在进行深度数据的修复时，因为没有足够的信息来对场景结构做出准确的推断，针对彩色图像的传统修复方法并不能直接应用于深度图像。本文提出了一种新的结构导向的修复方法来提高深度图像的修复质量。这种修复策略结合了传统方法和最近发展起来的非局部滤波方案。在待参考点的选取上，提出了一种在 8 个方向上有间距的选取方法，可以很好的体现待修补点周围的彩色信息和深度特征。距离、深度、颜色三种参考信息之间良好的平衡保证了修复效果。实验结果上，无论是主观视觉的评价，还是客观定量的误差评估，都证实了该修复框架的有效性。

立体视觉的一个典型应用就是立体视频会议系统，本文在对立体数据进行修复之后，设计并实现了一个利用左右分屏 3D 电视以及 Kinect 设备进行三维远程视频会议的系统。系统中首先需要解决的是设计稳定高效的设备驱动模块，它要在不同的平台上以相同的格式向上层模块提供稳定的数据流，OpenNI 具备这样的特性。为了让后台的修复算法与前端的界面显示都能流畅的运行，多线程是不可缺少的，本文采用了固定大小的循环缓冲区来实现各个模块的各个线程之间的数据共享，实际的运行效果证明这个高速缓冲区模块可以很好的平衡多线程并发编程中常见的线程安全性问题和运行效率问题。为了提高视频会议系统的流畅度，需要采取多种方法对计算过程进行加速，本文用到了 CPU 多线程对修复算法进行加速以及使用 GPU 中 CUDA 自定义核函数对模型重建与视图转换进行加速的方法。实验证明，这两种加速方法能够显著提高深度数据修补的速度与立体视频显示的帧率。

关 键 词：深度修复，立体视频，计算加速

论文类型：应用基础研究类

ABSTRACT

In modern times, the emergence of remote video greatly facilitates the communications among people, and the development of three-dimensional visual makes video communication more promising. Depth acquisition becomes inexpensive after the revolutionary invention of Kinect. It's a practical approach to construct three-dimensional model data from Kinect, and achieving stereoscopic video conferrencing system.But for computer vision applications, depth map captured by Kinect require additional processing to fill up missing parts. In the reconstruction process of three-dimensional model with Kinect data, missing color data needs to be repaired before final display due to the perspective transform. This paper firstly studies the method of Kinect to build three-dimensional data, then fixes the missing of various reasons for three-dimensional data with structure-oriented framework to repair the depth data and Fast Marching Method to repair the color data, finally designs a conference system of three-dimensional video and applies the repair algorithm on it, at the same time studies a variety of acceleration strategies to improve the playable frame rate of the stereoscopic video.

While repairing the depth data, conventional inpainting methods for color images cannot be applied directly to depth maps as there are not enough cues to make accurate inference about scene structures. In this paper, we propose a novel fusion based inpainting method to improve depth maps. The proposed fusion strategy integrates conventional inpainting with the recently developed non-local filtering scheme. In choosing the reference point, we propose a selection method which has space in eight directions, which may reflect the color and depth characteristics around the point to be required well. The good balance among distance, depth and color information guarantees an accurate inpainting result. Experimental results confirms the effectiveness of the proposed method whether from the subjective evaluation or the quantitative accuracy analysis.

One typical application of Stereo Vision is the remote video conference system. This paper designs and implements a system with split-screen 3D TV and Kinect device for remote video conferencing after repairing the three-dimensional data. What the system needs to solve at first is to design stable and efficient device driver modules which should

provide stable data streams to upper modules on different platforms at the same format, ant OpenNI possesses such characteristics. Multithreading is indispensable to make both background restoration algorithm and front end interface operate smoothly. This paper adopts circular buffer of fixed size to achieve data sharing among various threads of each module. Actual operating results prove that this high-speed buffer module can balance common thread safety issues in multi-threaded programming and operational efficiency well. In order to improve the fluency of video conferencing system various methods need to be adopted to accelerate the calculation process. This paper uses a multi-threaded CPU to accelerate the repair algorithm and CUDA custom kernel in GPU to accelerate the reconstruction of the model. Experimental results confirms that both of the acceleration method can significantly improve the speed of repairing depth data and the frame rate of stereoscopic video to display.

Keywords: Depth Inpaiting, Stereo Video, Accelerated Computing

Type of Dissertation: Applied Basic Research

插图索引

表格索引

符号对照表

符号	符号名称
I^{ir}	红外图像
I_r^{rgb}	彩色图像红色分量
I_g^{rgb}	彩色图像绿色分量
I_b^{rgb}	彩色图像蓝色分量
w_r	红色分量的权重
w_g	绿色分量的权重
w_b	蓝色分量的权重
M	符合条件点的数量
$\boldsymbol{l}$	左摄像机参数
$\boldsymbol{r}$	右摄像机参数
cm	厘米
$\boldsymbol{u}$	图像坐标系行坐标
$\boldsymbol{v}$	图像坐标系列坐标
X_c	世界坐标系中的点
$\overline{X}_c$	摄像机坐标系中的点
f	摄像机焦距
P	摄像机矩阵
K	摄像机内参数矩阵
$J(\bullet)$	彩色图像的结果信息
$J(p)$	像素 p 的结构信息
$d(p)$	像素 p 的深度信息
∇	梯度算子
$\mathrm{N}(p)$	点 p 附近的一组像素点
$w'(p,q)$	由点 p 对点 q 进行修补的权重
s_1	距离信息权重
s_2	深度信息权重
s_3	彩色信息权重
h_1	距离信息权重因子
h_2	深度信息权重因子
h_3	彩色信息权重因子

E	完整图像区域
Ω	待修复图像区域

缩略语对照表

缩略语	英文全称	中文对照
3D	Three Dimension	三维
RGB	Red Green Blue	红绿蓝
TOF	Time Of Flight	光飞行时间
1080P	1080 Progressive scan	1080 逐行扫描
SDK	Software Development Kit	软件开发套件
MAE	Mean Absolute Error	平均绝对误差
PCA	Principal Component Analysis	主成分分析
PDE	Partial Differential Equation	偏微分方程
CDD	Curvature Driven Diffusion	曲率驱动扩散
FMM	Fast Marching Method	快速行进算法
ABS	Arch Build System	Arch 编译系统
AUR	Arch User Repository	Arch 用户版本库
IDE	Integrated Development Environment	集成开发环境
GCC	GNU Compiler Collection	GNU 编译器套件
GTK	GIMP ToolKit	GIMP 工具集
OpenGL	Open Graphics Library	开放图形库
OpenNI	Open Nature Interface	开放式的自然交互
API	Application Programming Interface	应用程序编程接口
RTP	Real-time Transport Protocol	实时传输协议
CPU	Central Processing Unit	中央处理器
GPU	Graphic Processing Unit	图形处理器
FPGA	Field Programmable Gate Array	现场可编程门阵列
CUDA	Compute Unified Device Architecture	计算统一设备架构

目录

第一章 绪论

1.1 课题的意义和目的

随着互联网技术的发展，在现代化的生活和工作中，视频会议能够解决人们因地理位置不同而造成的沟通不便，特别是节约了双方在往返途中所花费的费用。同时，伴随近几年立体视觉的发展，3D(Three Dimension)视频也丰富了人们观看视频的效果和方式。

时至今日，3D 技术的身影不仅出现在科学研究和电影院中，也已应用在人们日常生活中的其他方面。如果在传统的远程视频会议中应用 3D 技术，能够实现视频画面与 3D 虚拟画面的全面结合，更加生动直观的进行网络视频会议,增强人与人之间沟通的真实性、互动性、立体性。而随着协同办公和视频会议的结合，商务沟通的方式在不断升级；在高清视频技术应用带动下，商务会议的质量的也在快速的提升，整个视频会议的市场也正在逐步扩大[1]。此外，3D 技术的应用也将会大大激发用户的兴趣和热情。由此可见，将来实体会议很有可能会被立体视频会议所替代，视频会议行业的快速发展时代即将到来。

图 1.1 3D 视频网络交流示意图

3D 技术应用的一个前提是深度数据的获取，深度数据作为一种独特的数据形式,从采集来源上与传统的彩色图像有着很大的差异,虽然在需要一定的染色或者渲染后才可以进行可视化显示,但是其中蕴含的立体场景数据却在 3D 重建中发挥着重大作用。

Kinect[2]是由微软公司在 2010 年发布的一款非接触式传感器。在外形上与传统的网络摄像机十分相似。从外形上看，Kinect 设计有三个摄像头，其中位于中间

位置的摄像头是 RGB 彩色摄影机，从正面看位于左边的是红外线发射器，位于右边的是红外线摄影机，这两个镜头共同组成 3D 结构光深度感应器。作为微软亚洲研究院的研究成果之一，它不仅能够提供三维的立体数据，还能捕捉玩家全身各部分的动作，通过特定的应用程序，使得玩家可以用身体来进行游戏。同时 Kinect 还搭配了底座马达，通过编程控制，Kinect 可以随着物体移动而移动。此外，Kinect 内部还建有由多组麦克风同时接收音频信号的阵列式麦克风，用于在比对后消除杂音[2]。

本文是在立体视觉技术发展的背景下，研究并设计一套 3D 视频会议系统，而 3D 视频会议系统的基础是立体数据的获取与显示。目前获取立体数据设备之中较为廉价并且实用的就是 Kinect，但是 Kinect 本身由于制作工艺上的问题会导致立体模型重建过程中所需要的数据不完整且有噪声，这会对立体视频会议的效果产生重大的影响。如何将采集到的三维数据进行修复，并按照计算机视觉方法利用彩色数据和深度数据重建出三维模型和立体视图，再将相应的理论和算法实现为一套可用的系统，是本文的重点研究对象。

1.2 深度数据修复研究现状

深度信息是理解现实世界场景的核心线索之一，故其在计算机视觉中非常重要。传统的获取深度信息的方法基于多视图几何，其灵感来自于人类的双目系统。近年来，许多新的设备，包括飞行时间传感器[3]、结构光[4][5], 以及革命性的体感设备 Kinect，都引入了深度采集功能。

Kinect 可捕捉某个场景在数米范围之内的深度数据和彩色图像，然而，因为捕捉到的深度图像有一些不足之处，比如遮挡、反射以及其他光学因素，如果将其直接用于场景重建的话，会导致很多问题。

Chiu 等人在 2011 年提出了一种将模型进行交叉的方法用来提高 Kinect 的深度图像质量[6]。它的算法依据就是假设对彩色图像的 RGB 通道的进行线性组合后，可以合成红外图像。合成的红外图像和深度的红外图像是对同一立体场景的观测数据。若给定一个红外图像 I^{ir} 和一个 RGB 图像 $(I_r^{rgb}, I_g^{rgb}, I_b^{rgb})$，则期望得出各频道的权值 $w=(w_r, w_g, w_b)$，因为转换后的 RGB 图像更加类似标准红外图像，且在立体匹配过程中有更多对应点。我们通过计算可以用匹配点的数量来对立体匹配的性能进行评估，表示为 $M(I^{rgb}, I^{\text{ir}})$。由此产生的优化问题如式(1-1)所示：

$$\sum_{w_r+w_g+w_b=1}^{\max} M(w_r * I_r^{rgb} + w_g * I_g^{rgb} + w_b * I_b^{rgb}, I^{ir}) \tag{1-1}$$

为避免受到投影图案的影响，方法中使用了由带外红外投影仪产生的红外图像。该优化问题可通过对平面 $w_r + w_g + w_b = 1$ 进行网格搜索和平均采样来解决。但若图像中物体表面颜色差异极大，导致深度值之间相差甚远，这种方法会导致误差较大。

Matyunin 等人在 2011 年提出了一种过滤方法以填补这种遮挡空洞，提高了 Kinect 深度图的时间稳定性[7]。基于 TOF(Time of Flight)原理的深度传感器捕获的深度图像所具有的缺陷与此十分相似。Kim 等人在 2010 年提出了一种基于双边滤波的方法，采用颜色和深度信息对深度图像进行空间增强[8]。Zhu 等人在 2008 年提出了一个方法，试图结合飞行时间深度数据和传统的被动式立体估计获得高精度的深度图像[9]。

图像修复技术[10]用于恢复彩色图像中的受损区域，这是另一个相关的研究领域。在这个领域的早期工作中，研究者是通过配置强度扩散实现的[10][11]。考虑到图像中结构信息的冗余，有的研究者提出了基于纹理合成的方法[12][13]。最近，Bugeau 等人针对修复问题提出了一个全面的框架，包括扩散、纹理合成，以及内容的连贯性[14]。

在深度图像中填补孔洞被视为修复难题。然而，与可见光图像的修复方法不同之处在于，在深度图像中很少有甚至没有纹理，确定对应于对象边界的终止位置就比较困难。此外，纹理合成也很难在深度图像本身中实现。

对于 Kinect，有一个与深度传感器几乎同步的可见光传感器，可以获取高分辨率的可见光彩色图像。本文提出了一种新的融合结构信息的修复框架，通过结合颜色、距离、深度、结构信息来对深度图像的缺失部分进行预测与修复。虽然本文专注于 Kinect 的深度颜色传感器对，该方法可以很容易地扩展到其他任何采用阵列传感器的多光谱系统中。

Kinect 设备自身也一直在发展，2014 年 7 月 15 日，微软发布了新一代的 Kinect for Windows 产品 Kinect2。相比与之前一代的产品，新的 Kinect 在深度数据获取上采用了 TOF 技术来得到更高的精确性以及对小物体的识别准确度，并提供高骨骼追踪的稳定性。同时彩色图像的画质也升级到了 1080P 全高清，并且拥有更宽广的视角。对于骨骼追踪功能，除了更加稳定外，完整骨骼也从一代的 2 个支持到了 6 个，全身的骨骼点从 20 个上升到了 25 个。跟踪的姿势更为精确和稳定。此外，在体感应用逐渐进入大众日常生活的趋势下，Kinect2 增加了许多细节元素，提供了可自由开启关闭的手势识别、手指指尖追踪等功能。可惜的是，这一系列的升级，并没有改善深度图中一直存在的深度缺失问题。

1.3 立体视觉概念介绍与发展现状

1.3.1 立体视频概念介绍

研究表明,人类的视觉是一种立体的视觉，实质上是一种深度认知的过程，也是一个将动态的复杂信息进行协同处理的过程。当人类在两只眼睛同时注视某一物体时，这两只眼睛的视线会相交于一个注视点。物体的反射光通过晶状体折射回到视网膜上形成光点，此光点与之前的注视点相对应，这两个点将信号转入大脑的视觉中枢，即可合成一个物体完整的像，由此形成了人在观察三维世界时的立体感。通过这个过程，人眼不但能看清注视点的信息，而且注视点与周围物体间的深度、距离、凸凹等等信息都能分辨出来，人类的这种视觉就叫做立体视觉[14]。

在了解了人体视觉的成像原理之后，便可通过人为制造“欺骗”人体立体视觉的影像的方法来使人的眼睛对本来是平面的物体产生立体的大脑印象。

在图形图像学中，与人类的立体视觉感知的过程相似，立体视觉的基本原理是从两个或两个以上的视点同时观察同一物体,从不同的位置获取被观察物体的多幅图像,然后根据三角测量原理来计算图像各象素间的位置偏差（也叫做视差），以此来获取该物体的三维信息[15]。

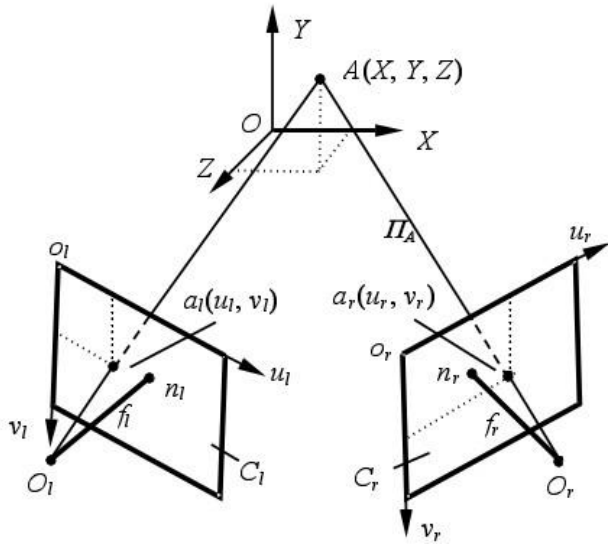


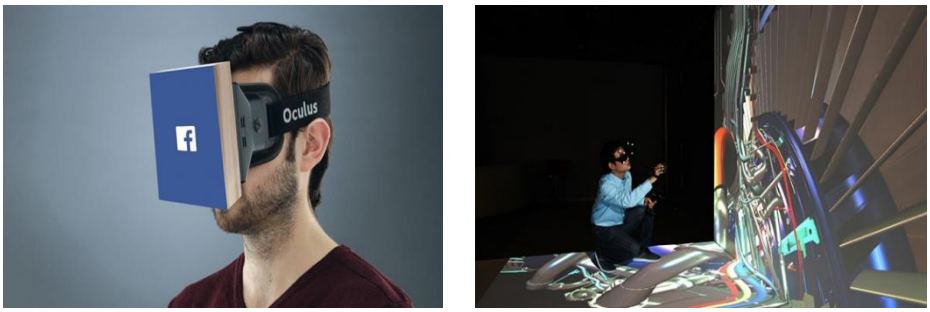
图 1.2 立体视觉原理介绍图[15]

立体视觉系统中的双目视觉，一般由两部按照人类双眼角度摆放的摄像机构成。空间位置关系如图 1.2 所示，图中的下标 l 和 r 分别表示左、右摄像机的相应参数。物理世界空间中一点 $A(x, y, z)$ 在两个摄像机的成像面 C_l 和 C_r 上所对应的像点分别为 $a_l(u_l, v_l)$ 和 $a_r(u_r, v_r)$。它们都是对象点 A 所成的像，也被称作“共轭点”。已

知了这两个共轭的像点，分别作它们与各自相机的光心O_l和O_r的连线，即投影线a_lO_l和a_rO_r，它们的交点即为世界坐标系中对应的像点$A(x,y,z)$。

1.3.2 立体视频发展介绍

随着图形图像学的发展以及深度设备制造工艺的提高，目前常见的二维平面视频表现力单一，不能真实地呈现场景的立体效果，已经不能满足人们对于视频高质量和高交互性的需求，由此产生了 3D 立体电视。根据捷孚凯市场咨询（中国）有限公司的数据，在中国市场，3D 电视的销量在整个电视市场中所占的百分比在 2011 达到了 12%，在 2012 年则达到了 25%，全年的销量超过了一千万台。在各大主流厂商，如三星等的支持下，普通家庭在置办电视时也已倾向于选择带有 3D 功能的电视，业界公认 2010 年为 3D 元年。据有关机构预测，在最近几年，3D 电视将成为发展最为迅速的高科技产品之一，到 2018 年，全球 3D 立体电视的销售量很有可能达到六千万台。将来 3D 立体影视制作将会有一个很大的发展机会。在国家的“十二五”规划中,3D 电视被广电总局列为重点规划项目。而 3D 电视涉及多项关键技术，包括 3D 视频源的制作、传输和立体显示等。



(a) Oculus 头戴式显示器　　(b) 工业设计中的立体模型

图 1.3 立体视频特效

如图 1.3 所示，(a)为 Facebook 公司收购的 Oculus 头戴式虚拟显示器，图(b)是工业设计中使用立体视觉来进行模型的设计与观察。立体视频的优点是临场感强、交互性好，但缺点是立体视频数据获取、制作的不方便。

世界经济快速的发展节奏，使越来越多的企业开始应用网络会议，在降低成本的同时，得到实时高效的决策。网络会议是指以互联网为媒介，将参会者的视频、语音、图片、文字等多媒体信息进行分发，到达各个与会者的网络终端界面上，使得在地理上相隔很远的参会者都可以通过多种非接触式的方式相互交流，加强与会者之间的沟通，增强其对会议内容的理解。目前主流的是视频网络会议。视频网络会议能够直观、全面地传达会议信息，但存在这样几个问题：视频数据对网络有

很高的要求，基于高端专线和硬件设备的视频会议产品昂贵，无法普及；与会者采取视频源切换或多个显示窗口观察其他与会者；会议临场感较差，与会者无法进行逼真生动的交流。这种情况下，基于立体视频技术的虚拟会议系统应运而生[16]。

目前的视频会议创新层出不穷，刚刚推出支持 4G 手机应用、Android 系统和平板电脑的视频会议系统，创维等多家视频会议系统制作的厂家就紧接着又开始研究 3D 视频会议系统。这种系统在功能、性能和技术发展前景上都有多个领先业界的优势，包括技术、产品和服务等多方面，因而受到了越来越多高端用户的喜爱。

1.4 本文的主要内容

本文的工作受启发于三维立体视频的发展与深度数据修补现状，从融合结构信息对深度数据进行修复，并尝试采用深度和彩色信息进行三维重构，再对立体视图中的彩色数据进行修补，最终实现了一套可用的三维远程视频会议系统。

本文下面章节的组织结构如下：本文的第二章详细介绍了使用 Kinect 立体数据进行三维重建的方法，以及立体视频的概念和目前 3D 视频的制作技术。第三章从分析 Kinect 设备深度数据缺失原因开始，提出了结构导向的修复框架来对 Kinect 的深度图进行修复，并对三维重建后的彩色图像进行修复。第四章是从工程角度讲述立体视频会议系统的设计与实现，其中包括各个模块的设计，以及对第三章算法的简化实现和计算加速。第五章对本文的工作进行了总结和展望。

第二章 Kinect 数据形成立体视频原理

2.1 Kinect 数据格式与获取原理

Kinect 提供的数据有深度数据，彩色数据，音频数据。通过相应的 SDK(Software Development Kit)还可以得到骨骼点数据。Kinect 获取场景数据的示意图如图 2.1 所示。

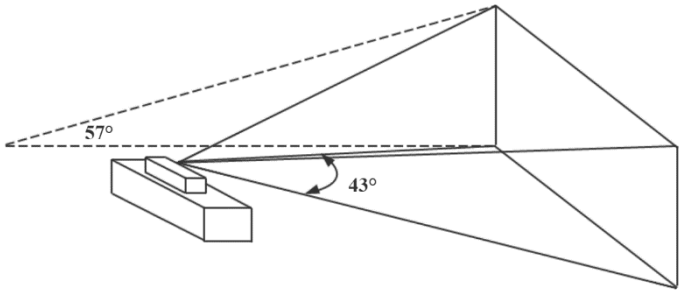


图 2.1 Kinect 获取场景数据示意图

其中红外摄像机视场的形状是个横着放的四棱锥。离摄像机越远，其视场横截面积就越大，也就是说影像的宽度和高度与摄像机视场所在物理位置并不是一一对应的。Kinect 的传感器获取的深度数据范围是从 1.2 米到 3.5 米，机身转动范围是±27 度，垂直方向的视角是 43 度，水平方向的视角是 57 度。Kinect 对深度图像的处理速度可达到 30 帧/秒，输出的深度图像有效数据位是 11 位，分辨率是 640× 480。在输出的深度数据中，如图 2.2 所示，每个像素所占空间为 16 比特，这叫做 BytesPerPixel 属性，也就是说每个像素所占的空间为 2 字节,且每一个像素的深度数据值只占用了 16 个比特中的 13 个，如图 2.2 所示。

图 2.2 Kinect 深度数据格式示意图

Kinect 的深度数据是通过两个光学镜头配合来进行获取的。一个红外发射器向空间中发出镭射光，另外一个红外摄像头接收镭射光。这种红外线是人眼所看不到的，在发射到立体空间之前，光线还会经过发射镜头前的光栅，这样镭射光就会

均匀地投射到待测量的立体空间中。用于接收的摄像头会记录下空间中的各个散斑，得到的原始散斑是会根据距离不同而呈现出不同的特征，经过特定的匹配算法，在内置的计算芯片进行计算，最后得到如同彩色图像一样的深度矩阵。摄像机拍摄到的激光散斑如图 2.3 所示。

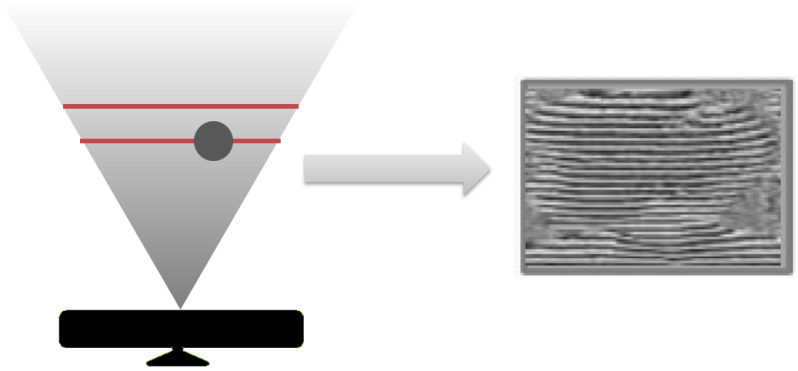


图 2.3 Kinect 散斑匹配示意图

具体的计算过程如下：已知 Kinect 能获取到的有效范围是距离摄像头一米到三米的空间，则从最近处开始，每隔一定距离（设定 1cm）,取一个参考平面进行标定。标定时选取的间距越小，最后标定得到的深度数据精度就越高。在计算深度数据的时候，在拍摄到待测量的激光散斑图像后，将该图像和已知深度的参考图进行互相关的匹配运算。将相关运算的结果绘制一幅峰值图像，然后再对这个图像进行插值运算就得到了整幅图的深度数据。

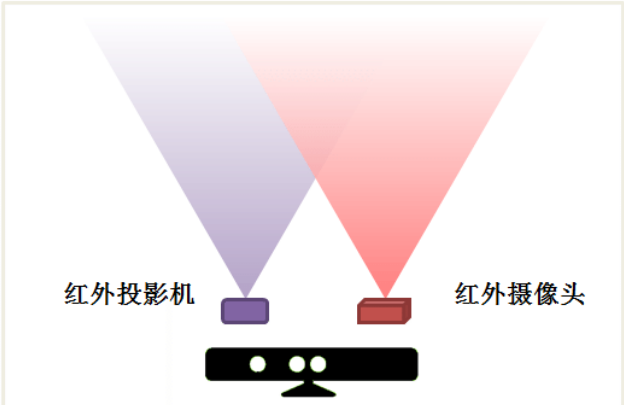


图 2.4 Kinect 深度数据获取摄像头的位置关系示意图

由于深度摄像头与彩色摄像头在光学参数方面的差异，以及如图 2.4 所示两个摄像头视角的不同，深度图像与彩色图像并非完全匹配。每个像素的测量精度都与其离 Kinect 摄像头距离的远近相关，距离越大，则精度越小。

2.2 使用 Kinect 数据重建点云数据

在计算机视觉应用中，三维点云数据的重建往往需要利用摄像机所拍摄到的图像来还原空间中的物体。要建立立体视觉，前提是先从两个及以上视角来观察同一个物体。但是从 Kinect 设备获取的数据只有一个视角，或者说是两个差异很小的视角。并且有视角差距的两幅图像并不都是普通人眼看到的彩色图像。因此如果需要用这些数据重建出立体数据，需要利用深度数据进行模型重建，再利用彩色数据对模型进行染色处理。最后，通过设置两个不同角度的虚拟摄像机，对重建出来的模型进行虚拟摄像来得到近似人双目视觉看到的效果[17]。

无论是从已标定图像进行三维重构，还是从未标定图像对三维物体进行重构，其采用的基本方法就是分层重构法，包括射影重构、仿射重构和度量重构三个步骤。重构前，首先要确定计算机视觉中的相机、图像以及世界这三个坐标系。世界坐标系就是 Kinect 观察到的场景坐标系，相机坐标系就是从 Kinect 本身的视角作为原点对场景观察得到的坐标系，在这里包括两种，一种是深度图的相机坐标系，一种是彩色图的相机坐标系。图像坐标系就是指从 Kinect 数据流中取出的视频帧中图像的坐标系，简单理解，就是根据分辨率确定的行和列以及对应的数据。如图 2.5 所示，是这三种坐标系在空间中的位置关系的简单介绍[18]。

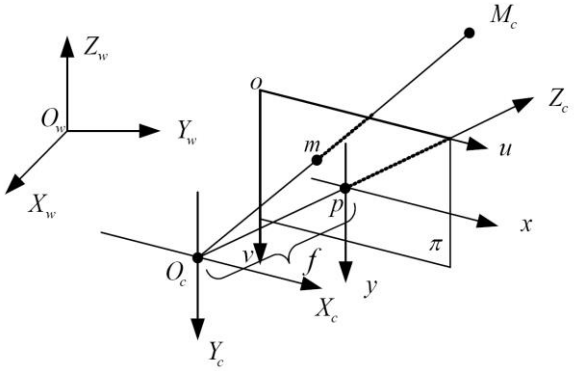


图 2.5 计算机视觉中三种坐标系示意图[17]

1) 世界坐标系（ X_w Y_w Z_w ）

用户自定义的三维坐标系，需要满足右手法则，用来描述三维空间中的物体和相机之间的相对位置关系。可以选取 Kinect 拍摄到场景的任意一点，但是需要确定好它与摄像机的关系。

2) 摄像机坐标系（ X_c Y_c Z_c ）

以彩色摄像机或者深度摄像机的光心点作为原点，Z_c 轴与摄像机的光轴重合，并垂直于成像平面，也就是 Kinect 前置的平面。采取摄影的方向，即 Kinect 面向的方向，为 Z_c 轴的正方向， X_c 、Y_c 轴与图像物理坐标系中的 x 、 y 轴平行。 O_c

O两点间的距离为相机的焦距 f。这里主要是深度红外摄影机与普通彩色摄像机坐标系。深度红外发射器只是发射红外线，不涉及到摄像机坐标系的建立。

3) 图像坐标系

一般按照图像的左上方作为原点。坐标系表示为（u、v），不同之处在于，坐标是以像素为单位。在 Kinect 中分辨率为640×480，u的范围为$[0,640)$，v范围为$[0,480)$，从捕获的数据帧中根据坐标换算得到相应的数据。

Kinect 得到的数据，是红外摄像机和可见光摄像机分别拍摄到的在图像坐标系下的数据。为了使用这两种数据构建三维点云数据，需要将它们转换到同一个物理坐标系中进行表示。这就首先要确定可见光摄像头与和红外摄像头的内参数矩阵和外参数矩阵。光线进入摄像机透镜时会产生径向畸变和切向畸变两种变形，为了对畸变的图像进行校正需要对摄像机进行标定，也就是求解摄像机的内参数。摄像机外参数刻画了摄像机坐标系与世界坐标系之间的移动关系；摄像机的内参数反应的是摄像机的内部结构。

常规的标定方法是用独特设计的黑白格子进行校准。如图 2.6 所示，通过标定图像上黑白格子交点处的三维坐标与其对应的图像点之间的位置关系计算出内外参数矩阵。

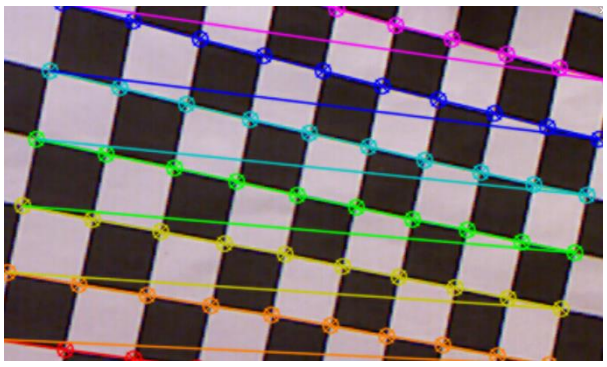


图 2.6 可见光图像内参标定版示例

得到摄像机的内参数矩阵之后，就可以从像素点的摄像机坐标系换算出它对应的空间点在世界坐标系中的坐标。这也就是三维计算机视觉中的较为核心的一个问题，由平面信息进行三维重构。一般来说，摄像机的投影变换是从三维空间到二维平面的射影变换，所以无法从空间中对一个景物所拍摄的单幅图像中恢复出来它的几何结构。因此三维重构一般是要从一个空间景物的多幅图像（多个角度）中来恢复景物的空间几何结构。如果已知摄像机矩阵，三维重构就可以通过三角原理来实现[18]。

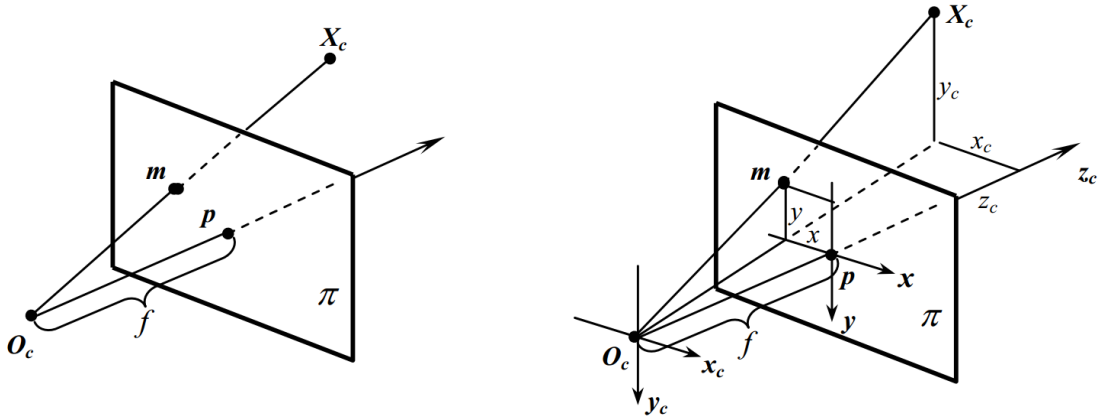


图 2.7 摄像机摄像的模型示意图[18]

首先确定一点，它在世界坐标系中用 X_c 表示，在摄像机坐标中的坐标用 $\bar{X}_c=(x_c,y_c,z_c)^{\mathrm{T}}$ 来表示。这个点大多选取标定盘上的中心点，并且实现安排好它到摄像机之间的距离。然后将该点投影到图像坐标系中，得到的点 m 在图像坐标系中的坐标用 $m=(x,y)^{\mathrm{T}}$ 来进行表示。根据三角形的相似性原理，可推导出世界坐标系中的点 X_c 与图像坐标系中的点 m 满足等式：

$$\begin{cases} x=\dfrac{fx_c}{z_c} \\ y=\dfrac{fy_c}{z_c} \end{cases} \tag{2-1}$$

上式可表述为矩阵形式：

$$z_c m=\begin{bmatrix} fx_c \\ fy_c \\ z_c \end{bmatrix}=\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} X_c \tag{2-2}$$

其中 $\bar{X}_c=(x_c,y_c,z_c,1)^{\mathrm{T}}$，$m=(x,y,1)^{\mathrm{T}}$，分别为世界坐标系中的点和图像坐标系中点的齐次坐标。这是从世界坐标系到图像坐标系内的一个齐次线性变换。如果记

$$P=diag(f,f,1)(I,0) \tag{2-3}$$

则变换(2-3)可表示为另一种相对简单的形式：

$$m=PX_c \tag{2-4}$$

式(2-4)中矩阵 P 是一个3×4的矩阵，也就是通常说的摄像机矩阵。在 CCD(Charge-coupled Device)相机数字离散化并加入摄像机内参矩阵后，这个公式可以写成：

$$m=K(I,0)X_c=PX_c \tag{2-5}$$

设置数字离散化后像素的矩阵长和宽为 d_x 和 d_y，并令 $f_x=f/d_x$，$f_y=f/d_y$，则有：

$$K=\begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-6}$$

通过公式的推导，并进行实际标定，可以最终得到深度摄像头和彩色摄像头在摄像机坐标系下的摄像机矩阵。进一步通过我们事先在摆放时确定好的标定板与摄像头之间的空间位置关系，得到一个描述旋转与偏移的矩阵，就可以得到标定版设定的世界坐标系与 Kinect 两个摄像机坐标系之间的转换关系，进一步得到两个摄像头在世界坐标系中的摄像机矩阵参数。

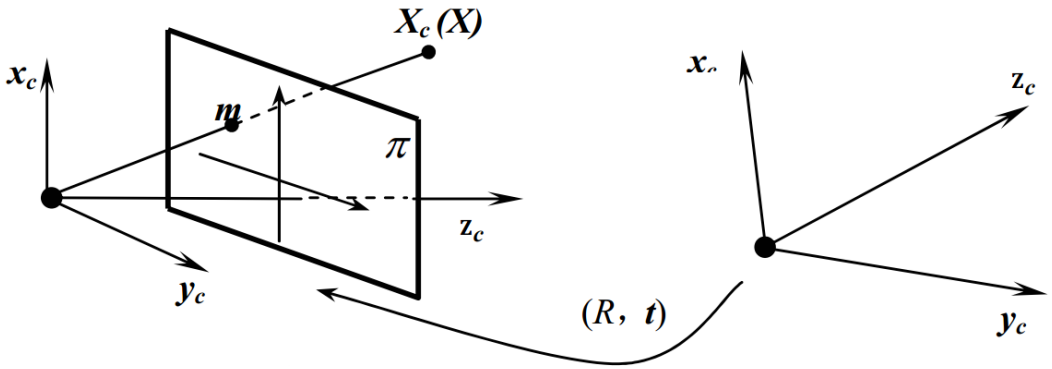


图 2.8 世界坐标系与摄像机坐标系转换的关系示意图[18]

如图所示，假设空间中的一点在世界坐标系的坐标为：

$$X=(x,y,z,1)^T \tag{2-7}$$

在摄像机坐标系的坐标为：

$$\mathrm{X_c}=(\mathrm{x_c},\mathrm{y_c},\mathrm{z_c},1)^{\mathrm{T}} \tag{2-8}$$

它们之间的关系，可以表示为：

$$X_c=\begin{bmatrix} R & -RC \\ 0^T & 1 \end{bmatrix} X \tag{2-9}$$

其中 $\tilde{C}$ 表示摄像机的中心在世界坐标系中的坐标，这个坐标是非齐次的，齐次坐标为：

$$\mathrm{C}=(\mathrm{C}^T,1)^T \tag{2-10}$$

综合上式,有：

$$m = K(\mathrm{I},0)\begin{bmatrix} R & -RC \\ 0^T & 1 \end{bmatrix} X = \mathrm{KR}(\mathrm{I},-\mathrm{C})\,\mathrm{X} \tag{2-11}$$

简写为：

$$P = KR(I,-\mathrm{C}) \tag{2-12}$$

上式即为摄像机矩阵包含了内外参数的表示形式。摄像机矩阵里的外参数矩阵是$R(I,-\mathrm{C})$。如果通过 $X_c = RX + t$ 来描述标定版与摄像机之间的关系，可以推导出摄像机矩阵[18]：

$$P = K(R,t) \tag{2-13}$$

摄像机矩阵是一个的3×4的矩阵，其中前三列构成的子矩阵 R 是一个可逆矩阵。根据以上公式的推导，设 Kinect 得到的深度数据中一点 $\mathrm{D}p(u,v,d)$,以及对应彩色数据中的一点 $Cp(u',v',r,g,b)$。 X_w 是它在世界坐标系中的位置。

$$d\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = PX_w \tag{2-14}$$

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = P'X_w \tag{2-15}$$

公式中 P 和 P' 分别为通过标定板标定后计算得出深度红外摄像头和彩色摄像头的摄像机矩阵。这两个矩阵对于一台 Kinect 来说是准确的，但是不同的 Kinect 估计没有普适性，需要重新进行标定。由上述两个公式可以推导出深度图图像坐标系与彩色图图像坐标系的转换公式如式(2-16)所示，其中 P^{-1} 为伪逆。

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = P'P^{-1}\,\mathrm{d}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2-16}$$

由此，可以将深度图中一点的深度数据对应到彩色图中一点的彩色数据中，满足了构造点云数据的前提。最终得到的结果是都在彩色图像的图像坐标系下，也就是对彩色图像中的每一点，都可以同时拥有深度数据和彩色数据。不过由于这两个摄像机在摄影范围上的不同，彩色图像的边缘可能会有一部分的深度信息缺失。这部分缺失并不是本文的研究内容。

完成重构之后，就可以从模型中设置虚拟摄像机来获取左右视图的数据。设置虚拟摄像机的过程实质上就是设置摄像机的旋转矩阵 R 和偏置矩阵 T 。

$$\begin{bmatrix} u_{left} \\ v_{left} \\ 1 \end{bmatrix} = P_{left} X_w d^{-1} \tag{2-17}$$

$$\begin{bmatrix} u_{right} \\ v_{right} \\ 1 \end{bmatrix} = P_{right} X_w d^{-1} \tag{2-18}$$

左右视图只要在偏置矩阵的 x 方向加一个正向或者负向的偏置即可得到左右视图的(u,v,d,r,g,b)。

2.3 立体视频显示技术介绍

2.3.1 立体视频显示技术分类

从制作方法来看，在最近几年，三维立体视频成像有四种方法比较常见，分别为色分法、光分法、时分法、以及全息法[16]。

色分法是一种较为成熟的 3D 立体成像技术，分为红绿模式、红蓝模式等，这些模式的原理相似。色分法会在同一副画面中分别以两种不同的颜色印制两个不同视角上拍摄的影像。在放映立体视频时需要通过对应的红蓝立体眼镜观看，仅通过肉眼观看只能看到模糊的重影。以红绿模式为例，使用的红绿眼镜在红色镜片下只能看到红色的影像，绿色镜片只能看到绿色的影像，双眼获得的不同影像使人脑产生立体画面。采用色分法时，需要在放映机前放置滤光片，投影出的光线将通过滤光片分成红绿蓝三原色光。通过红绿分色眼镜，人眼可以分别接收这些光谱的不同频率的部分，从而可以实现立体效果。色分法成本较低，播放传统电影的放映机在安装滤光片后就可以放映立体电影。

光分法也被称为偏振光技术。如图 2.9 所示。在放映立体视频时，光分法采用两部带有偏振镜的放映机，使放映机分别同步播放左右眼看到的两路视差影像。因此如果用户通过肉眼直接观看立体视频时，在显示屏幕上会看到画面是重影影像。所以用户需要通过配带 3D 眼镜来观看立体视频。光分法的 3D 眼镜实际上是两个相互垂直的偏振光镜片，通过这两个偏振光镜片，人眼可以分别看到屏幕上放映的不同视角图像，从而产生立体效果。采用光分法的立体视频在拍摄的时候，使用的摄像机通常有两个镜头，镜头间距离的与人两眼间距大致相等。光分法通常应用于 3D 电影。

时分法通过特殊的眼镜和显示屏，以超高速进行镜片透光性的切换，在调成不

透光的黑色时，可以分别遮蔽人的左右眼，使左右眼看到的是角度不同的画面。即显示屏在播放右眼画面时左侧镜片变为黑色，在播放左眼画面时右侧镜片变为黑色。镜片和屏幕画面的快速切换使得左右眼分别获得了左右视角的图像。时分法对显示器的要求较高，在显示过程中，需要以较高的频率轮换着显示左右视图的图像。与此同时，3D 眼镜也需要以同样的频率开关左右眼的显示镜片，从而使左右眼分别看到不同视角的影像。所以分光法的实现需要显示器支持较高频率的刷新速度，一般最少要达到 120Hz 以上，以至于人眼在观看立体视频时不至于太晃眼。相比于光分法，时分法的成像效果更好，但成本较高。

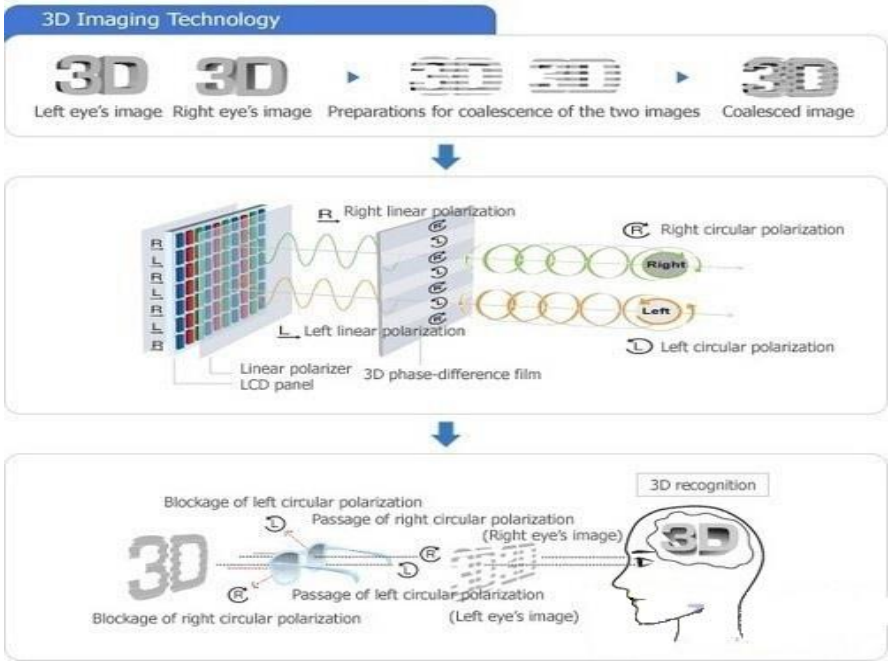


图 2.9 光分法示意图

全息法也被称为全息投影技术，通过利用干涉、衍射等光学原理记录并再现物体真实的三维立体图像。使用全息法成像，即使是在环境光线非常明亮的情况下，也能显示清晰、明亮的影像，从而产生令人震撼的演出效果，例如日本的虚拟偶像初音未来的演唱会。但是全息法更多地还是处于实验室研究阶段，产品较少而且价格昂贵。

2.3.2 立体视频显示效果的影响因素

一般来说，影响三维立体视频效果的因素有三个[19]：

1) 三维影片在制作、传输过程中所采用的编码格式和显示标准。
2) 三维视频源的可用性以及兼容性。
3) 显示终端的体验。

在本项目中将使用左右分屏的 3D 显示器，其左右视图均通过从重构的三维模型摄影而来。因此最终的体验主要收到模型重构与虚拟摄像方法的制约。其中重构出的模型质量是至关重要的因素。

三维模型的质量受到两个指标的影响，一个是模型中点云的完整度，一个是模型的分辨率。通过上一小节的分析可得到，模型中的点云数据完整度包括了深度数据的完整度和彩色数据的完整度。在模型建立之前，彩色数据是完整的，模型的不完整主要是由于深度数据的缺失。而在模型建立之后，需要输出左右视图的彩色图，这时候缺失的是彩色图，因此需要对彩色图进行修复。而模型的分辨率，一方面是指模型本身图片的分辨率，另一方面也受到摄像机噪声的影响导致数据不精确。

虚拟摄像方法在本文中指的是在模型重构后通过设置两个虚拟视点对模型进行摄影。因此虚拟视点的选取方法将很大程度上影响最后的立体视频效果。根据人眼立体视觉的原理，虚拟视点一般在原有模型视点的基础上分别平行左移和平行右移一段距离。移动的距离跟人的双眼之间距离有关。

2.4 本章小结

本章主要介绍了 Kinect 设备获取到的数据类型以及基本的数据格式，并简述了数据获取的基本原理。在这些数据的基础上，可以通过计算机视觉中三维重建的理论进行模型重建，重建的前提是对摄像机的内外参数进行求解。在重构好的三维模型中，依据与重建时同样的立体几何原理可以得到左右视图的深度图和彩色图。在第二小节中介绍了常见的三维视频的格式以及制作方法，本文采用了光分法中的左右分屏，最后对影响立体视频效果的因素进行了分析和总结，并简述了本文获取立体视频时左右视图的虚拟摄像设置方法。

第三章 立体数据修复方法

3.1 深度数据噪声原因分析

从 Kinect 获取数据的原理进行分析，深度数据的噪声形成主要有三个方面的原因。

1) 散斑不完整

从深度的获取过程来看，影响精度的关键是红外线激光散斑。当这种特定的镭射光线照射到物体表面的时候，会由于距离的不同，形成形状各异的斑点图案，称之为散斑。因此，如果物体表面没有办法形成散斑图案，后续的匹配算法就无法计算出来物体的深度信息，这是深度缺失的一种原因。实际情况中有很多因素会导致这种深度缺失的出现。其中最为常见的就是光学上的特殊材料，特别是吸收红外线或者对红外线进行全反射的材料，比如说镜子，或者透明的玻璃杯等。

另外在景深距离较远的两个物体边缘处，散斑图案有可能一半在前景物体上，一半在背景物体上，而距离的不同导致散斑的大小不同。匹配算法在得不到完整的匹配单元的图像时，匹配的结果可能会有误差，或者是匹配不到。

2）常规的图像噪声。

由 Kinect 软硬件本身所造成的噪声。比如说 Kinect 拍摄到场景中只有一个垂直地面的墙壁，那得到的深度图也会有一定的波动和误差。

3）视角导致的深度缺失。

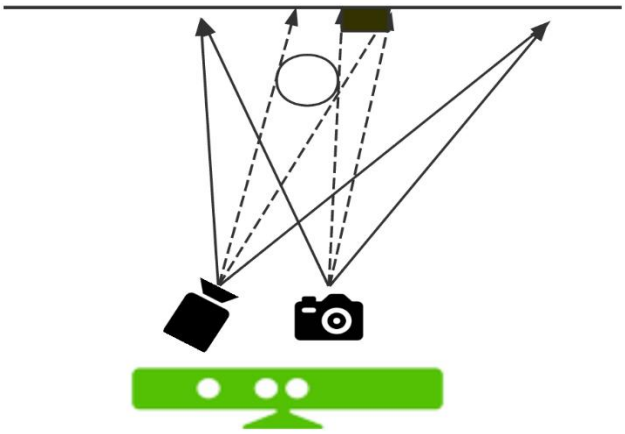


图 3.1 Kinect 视角差距示意图

在实际获取的场景深度数据中，有很大一部分的深度数据缺失，实际上是来自于红外发射摄像头和红外接收摄像头之间的视角差距。红外发射器和红外摄像头分别在 Kinect 摄像头阵列的最两边，如图 3.1 所示，左边的为红外发射器，后边的

为红外接收器。黑色部分就是接收器接收到的没有深度信息的部分。

3.2 融合结构信息的修复模型

上文分析了深度数据产生缺失的原因，本小节将介绍一种融合结构信息以及彩色图和深度图信息的框架进行深度修复的方法。

为了方便下文的讨论，首先介绍本小节中使用的符号。我们用一个字母 p 来表示深度图像和彩色图像上的像素位置。为了区分这两个图像坐标系统，我们添加一个后缀来表示他们的不同。比如说，用 p 表示彩色图像中的像素，那 p' 就表示深度图像中的对应点。$d(p')$ 表示像素 p' 的深度数值，而 $J(p)$ 表示像素 p 的彩色信息。彩色图像的结构信息用 $J(\bullet)$ 表示。此外，我们使用 Ω 表示深度信息不可用的区域。

基于深度和彩色摄像机之间的几何关系，根据多视图几何[20]可推导出映射函数(式 3-1)。由于对准有些复杂，只是将矢量映射函数表示为

$$p = m(p', d(p')) \tag{3-1}$$

公式(3-1)中深度图像中的像素 p' 对应于彩色图像中的 p 。为了精确对准，采用张氏标定方法[21]。相关原理已经在第二章第二小节进行了介绍。

3.2.1 基于融合的图像修复

有了映射函数(3-1)，我们可以对一个深度图像坐标上的特定点 p'，以及该点在彩色图像上的对应点 p，形成一个扩展的数据集合，用 $\{d(p'), J(p)\}$ 来表示。在这个数据集合中，颜色信息是对于所有像素点均可用的。然而，对于 $p \in \Omega$ 的像素来说，深度信息会丢失。因此，在已知深度和颜色像素的条件下，深度图像修复需要修复丢失的深度信息。传统的图像仅是根据像素修复，没有额外的先验信息，对于 Kinect 捕获的深度图像进行修复也可视为局部修复。

投影之后，现在就可以仅使用彩色图像的坐标系统。在一般的修复方法中，像素 p 的未知深度 $d(p')$ 可由附近像素 q 的已知深度 $d(q)$ 预测出来。假设一级泰勒级数足够精确，该预测可用公式(3-2)来表示：

$$d(p) = d(q) + \nabla_{qp} d(q) \|p\text{-}q\|_2 \tag{3-2}$$

其中算子 ∇_{qp} 表示沿 qp 方向点 q 处的方向导数，$\|p\text{-}q\|_2$ 表示标准 L2 范数。根据微积

分原理，方向导数可由梯度得出。因此，公式(3-2)可表示为

$$d(p)=d(q)+\langle\nabla d(q),\text{p-q}\rangle \tag{3-3}$$

其中$\langle\nabla d(q),\text{p-q}\rangle$表示两个向量的内积，$\nabla$表示梯度算子。

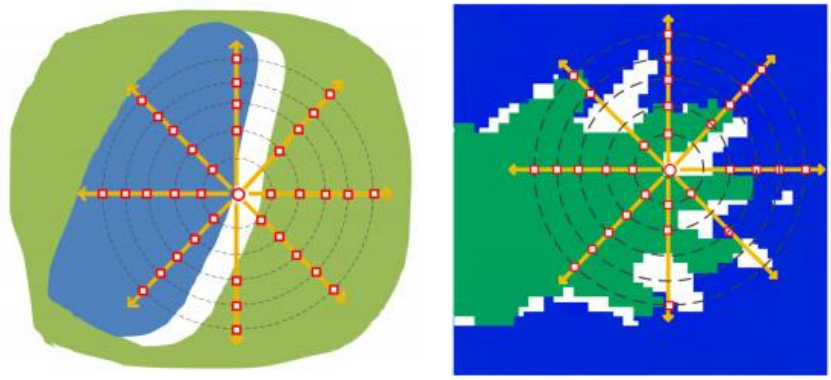
在深度修复和噪声去除的过程中，只利用一个参考像素进行预测是不够的。最近的去噪研究表明，非局部均值方案[22]通过利用图像的纹理冗余提供了一个准确的估计。最近的视觉图像修复框架也在其纹理合成部分中应用了类似非局部均值的方案[23]。在这些方法的基础上，我们使用以下深度图修复框架:

$$d(p)=\sum_{q\in \mathrm{N}(p)} w(p,q)[d(q)+\langle\nabla d(q),p-q\rangle] \tag{3-4}$$

其中$\mathrm{N}(p)$表示像素点$p(p\in\Omega)$附近的一组像素，$p\in\Omega$即为观察到的深度信息，权重$w(p,q)$将根据几何距离、颜色结构信息的连贯性以及深度的平滑性来综合确定。完成修复后，我们使用一个简易的中值滤波器来进一步提高深度图像的质量。

修复框架（公式 3-4）与基于快速行进的图像修复方法[24]有着相似之处。不同之处在于参考像素集$\mathrm{N}(p)$、相邻像素集以及权重分配的选择。

在基于快速行进的修复中，相邻像素一般从梯度方向选择。在本方法中，从已知像素p开始，相邻的像素集合$\mathrm{N}(p)$是从 8 个方向的射线中挑选出来的。图 3.2(a)说明了相邻像素集$\mathrm{N}(p)$的结构。具体的选择策略是，以参考像素为中心，在 8 个射线方向上进行搜索，若搜索到的点深度信息可用，就将像素添加到集合中；同时搜索的过程中有最大搜索半径R，以及最多搜索点n，其中任何一个条件满足，都会导致在这个方向上的搜索过程结束。如图 3.2(a)和图 3.2(b)所示，分别是抽象的搜索点展示与实际中含有手指的样本图搜索点展示，带箭头的射线代表搜索的方向，空心的小点代表被选中的参考点，白色区域为深度缺失部分。



(a)相邻像素集结构示意图　　(b)相邻像素选择草图

图 3.2 相邻像素选择示意图

深度图像通常比其相应的彩色图像更为平滑，因为实际拍摄到的景象，特别是在室内场景中的景象，其表面具有丰富的纹理，同时也保持了连续的纹理边缘。因此，在预测公式(3-4)中的一阶泰勒展开式中，相比于彩色图像，深度图像有更大的相邻区域。

确定深度边缘的难度也来源于对象表面的激光散斑的不完整。若仅仅基于深度线索，这个问题将难以解决。权重 $w(p,q)$ 的目的是通过考虑多个因素，尤其是彩色图像提供的结构信息，来评估 $N(p)$ 中每个像素对于修复 p 所作的贡献。在下一小节将详细讨论如何分配权重 $w(p,q)$ 。

3.2.2 结构导向的信息融合

与对应的彩色图像相比，深度图像的纹理较少，因此在修复区域中从外到内传播深度信息通常比在彩色图像中效果好。然而，难点在于如何确定在何处停止传播。为了解决这个问题，通过考虑几何距离、深度相似性以及彩色图像提供的结构信息，我们设计了一个加权函数来提高修复质量。加权函数形式如下：

$$w'(p,q)=\prod_{i=1}^{3}\exp\left(-\frac{s_i(p,q)}{2h_i^2}\right) \tag{3-5}$$

其中 $s_i(p,q)$ 是要考虑的三个因子，h_i 是与这三个因子相对应的系数。公式(3-4)中的权重是公式(3-5)中的标准版，其归一化形式为

$$w'(p,q)=\frac{w'(p,q)}{\sum_{q\in N(p)}w'(p,q)} \tag{3-6}$$

式(3-5)中的第一个因子是 p 和 q 之间的几何距离，其形式为

$$s_1(p,q)=\|p-q\|_2^2 \tag{3-7}$$

参数 s_1 表明，修复时越是相邻的像素就越是可靠。引入该参数是为了塑造预测时泰勒展开式中关于参考像素和待修复像素之间距离的可靠性。

公式(3-5)中的第二个因子代表深度感应的结构。由于修复区域中的很大一部分是因为遮挡形成的，这些区域的深度值往往较大。因此在修复的时候，给深度更大值的点赋予更高的权重，从而使用更远表面的深度值进行修复。为了给予更大深度值更高的优先权，我们使用如下函数进行评价：

$$s_2(p,q)=\left(1-\frac{d(q)}{d_{\max}(p)}\right)^2 \tag{3-8}$$

其中 $d_{\max}(p)$ 是选中相邻 $\mathrm{N}(p)$ 中的最大深度，即

$$d_{\max}(p)=\max_{t\in N(p)} \mathrm{d}(t) \tag{3-9}$$

最后一个因子考虑了彩色图像的结构连贯性，其形式为

$$S_3(p,q)=\sum_{t\in\beta}\| J(p+t)-J(q+t)\|_2^2 \tag{3-10}$$

其中 J 表示彩色图像的结构组成，$\beta=\{-l...,l\}\times\{-l,...,l\}$ 是 $(2l+1)\times(2l+1)$ 的矩形块构成的一组扩散区域。彩色图像的结构组成根据下式提取：

$$J=\arg\min_{J_s}\int\{|\nabla J_s|+\lambda(J_s-J)^2\}dp \tag{3-11}$$

可由全变分去噪模型实现[25][26]。

图 3.3 为结构组成提取的示意图。其中图 3.3（a）是原始的彩色图像，图 3.3（b）是原始图像的结构化示意图。图 3.3（c）表明原始图像的纹理组成，是通过从原始图像中去除结构化信息得到的，即图 3.3（c）可表示为 $J-J$ 。

（a）原始彩色图　（b）去噪后彩色图　（c）去噪前后图像差值

图 3.3 彩色图像的结构化分析

在非局部均值方法中，对于提高去噪性能[27]来说 PCA(Principal Component Analysis)是有效的。提取结构信息的过程类似于 PCA，不过 PCA 在图像中禁用了高频成分。由于我们的修补方案具有非局部均值结构，该结构的提取确实提高了修补的性能。

3.2.3 修复结果的评价方法

本小节设计了若干个实验来评估深度图像修复方法的性能。由于获得实测的

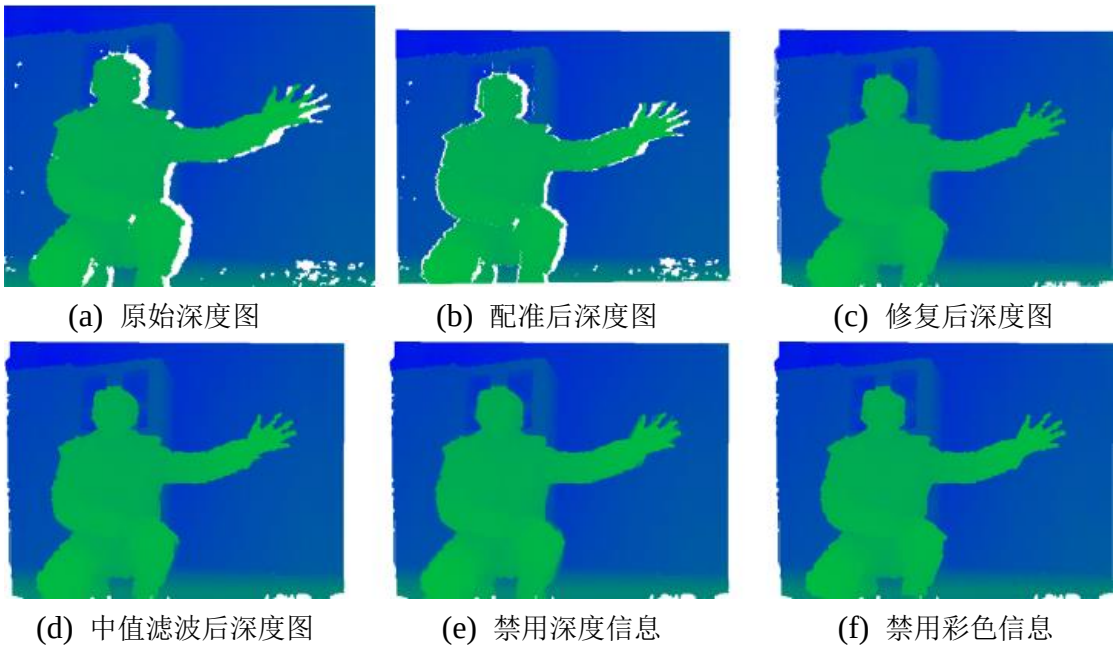
深度真值数据难度较大，本小节首先对含有非刚性前景对象的场景进行主观评价；然后，为了定量评估上述修复框架的准确性，也为了获取深度数据真值的便利性，我们使用刚性物体箱子来建立场景，并通过一个巧妙的办法来获取场景深度的真值。

1）主观评价

本节中的所有实验都将固定一些配置参数。若无特别指定，公式(3-5)中三个因子的系数分别被设置为 $h_1=1.0$、$h_2=0.112$ 以及 $h_3=0.045$。公式(3-6)和公式(3-9)中使用的集合 $N(p)$ 包含 8×5 个像素。这些参数的选取均是在大量实验的基础上能得到较为优异视觉效果的参数。如图 3.2(a)所示，对一个特定的像素 p 来说，8 个方向中各自选出 5 个像素。公式(3-10)中匹配矩阵 B 的大小被设置为 3×3。由于深度图像是一个 13 位的数据阵列，为了更加直观地显示深度图像，我们根据深度值来进行伪彩色染色处理，其中白色被用来表示深度信息丢失的区域。

由于彩色摄像机的视角范围大于深度摄像机，所以最终投影出来的深度图像要小于彩色图像。因此我们在评价时仅仅针对双方共有的内部区域修复，没有深度信息的外部区域会被忽略。换句话说，本文并不使用该修复框架进行外推。

图 3.4 显示了人体的深度图像修复效果。图 3.4(a)和图 3.4(b)分别为原始深度图像和投影到彩色图上的图像。图 3.4(c)和图 3.4(d)分别为未使用中值滤波和使用中值滤波进行处理的修复效果，在该场景中，使用中值滤波方法提高了视觉效果。从这两幅图中可以看出，人头部右边背景的深度信息得到了良好的修复。人手部边缘也恢复得很清晰。



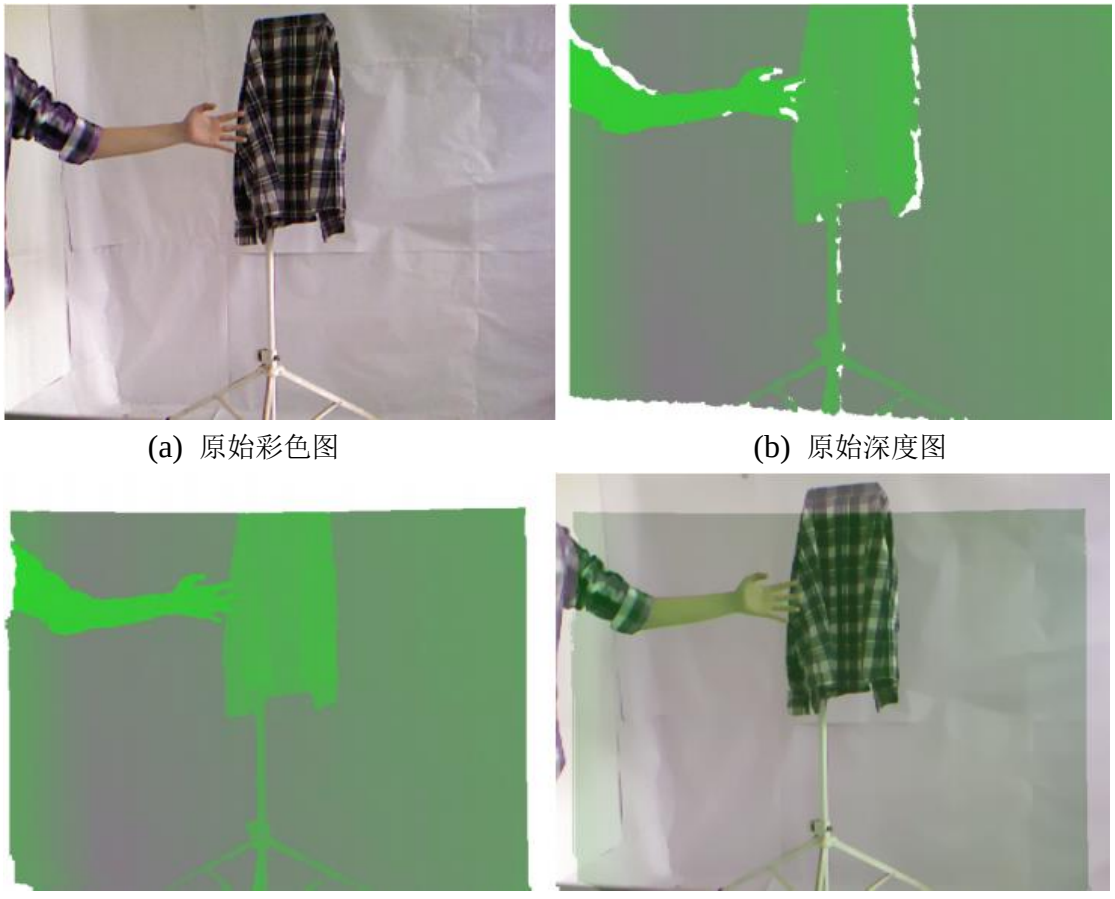
(a) 原始深度图　(b) 配准后深度图　(c) 修复后深度图

(d) 中值滤波后深度图　(e) 禁用深度信息　(f) 禁用彩色信息

图 3.4 人体深度图像修复效果

图 3.4(e)和图 3.4(f)所示修复结果由不同的融合策略产生。图 3.4(e)中禁用了深度感应结构，即设置式(3-5)中的 $h_2=\infty$ 。在该修复结果中，边缘较为模糊，尤其是人的手部周围。这一结果表明深度因子对在修复结果中生成清晰边缘作用巨大。图 3.4(f)禁用颜色结构信息，即令式(3-5)中的 $h_3=\infty$ 。该策略降低了颜色信息因子作为参考的权重，如人头部右方的背景区域，修复效果较差。

图 3.5 展示了手部的修复结果，设计该场景的目的是评估场景为纹理背景，前有非刚性物体对象时的修复性能。在该实验中，背景包括一块纹理杂乱的幕布。前景对象是非刚性的人类手部。场景的布置如图 3.5(a)所示,对应的深度染色效果为图 3.5(b)。如图 3.5(c)和图 3.5(d)所示，手臂、拇指和幕布附近的区域都得到了相当好的修复。图 3.5(d)表示将修复后的深度图像与彩色图像重合时的对比。本实验可证实，当背景含纹理时，该方法效果很好。



(a) 原始彩色图　(b) 原始深度图

(c) 修复后深度图　(d) 修复后深度、彩色融合图

图 3.5 复杂场景修复性能效果图

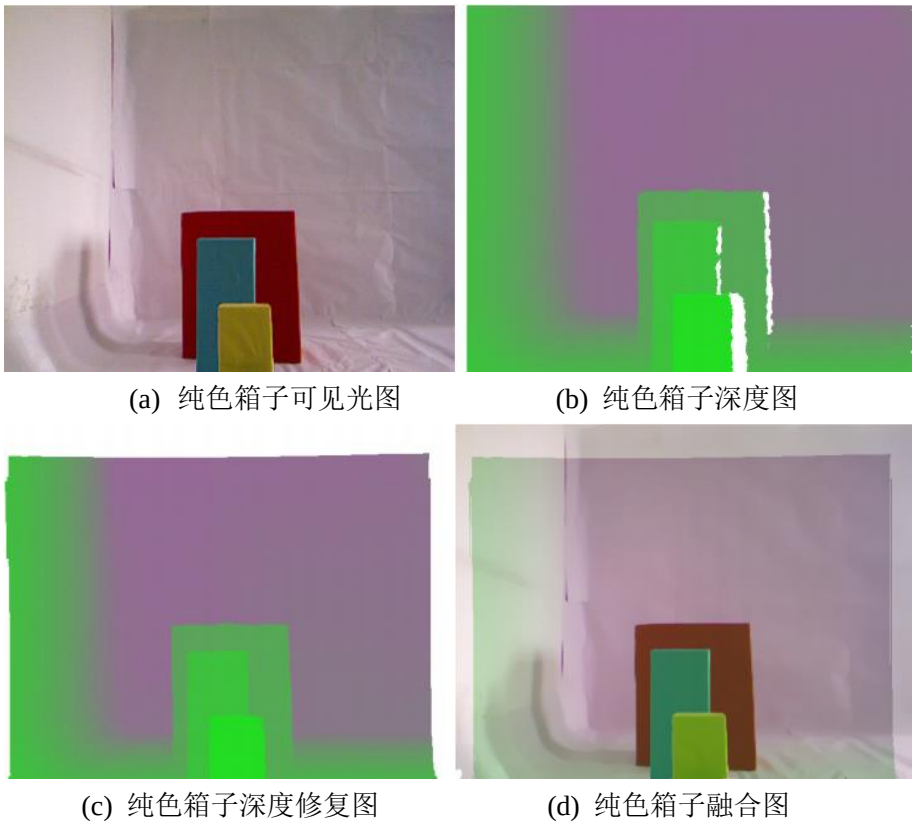
如图 3.5（b）所示，对于中指和食指之间小块区域的修复结果不够好。主要原因是该区域太小，无法匹配到正确的背景区域上。另一方面的重要原因是，这些区域是被一些不准确的深度测量占用了，本来应该是背景区域，却存在有前景的深度

信息，就是这些深度测量使得两个手指相连起来。为了增强该区域的深度信息，需要一种机制来填补这样的位置。

2） 定量评价

为了定量评估该方法的准确性，需要对场景中所有像素的深度获取真值。因为对于平坦表面，或者深度相差不大的两个物体，它们的深度真值更容易获取，所以使用三个刚性箱子来构建特殊场景。

通过 3.1 小节的分析，在 Kinect 深度图像中，未成功获取深度信息的像素可分为两类。第一类是 Kinect 投射红外光阴影中的背景像素。为了获取这些像素的真值，仅仅需要将前景对象移出该场景的阴影。同时为了对比颜色信息所起到的作用，本文设置了两组场景来进行真值获取的实验。



(a) 纯色箱子可见光图　(b) 纯色箱子深度图

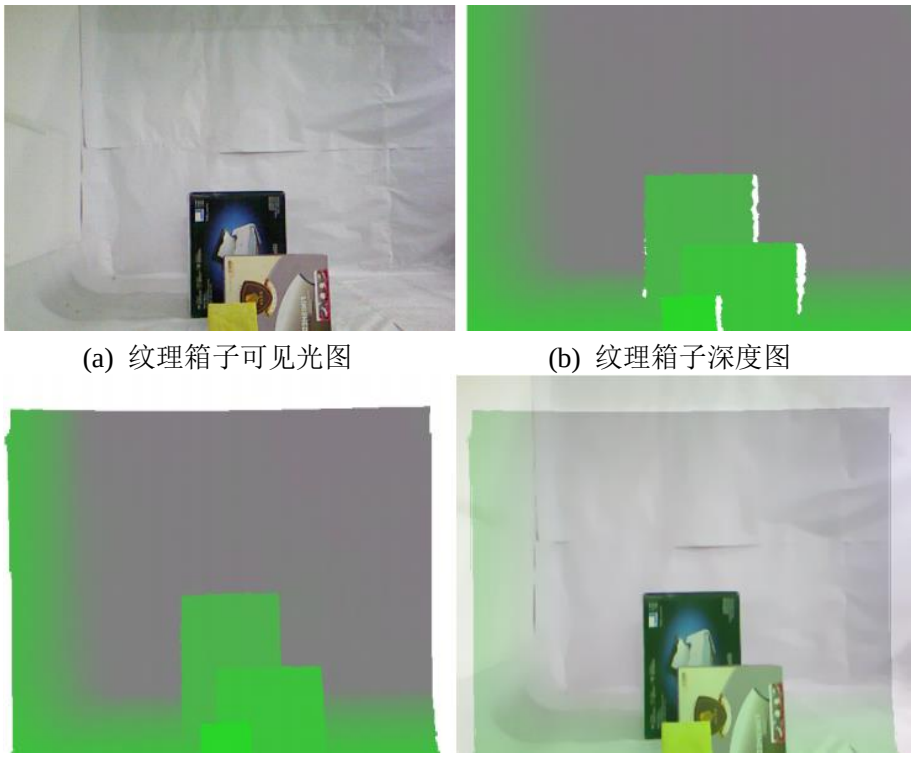
(c) 纯色箱子深度修复图　(d) 纯色箱子融合图

图 3.6 定量精度评价的色箱示意图

如图 3.6 和图 3.7 所示，若要得到这三个平坦表面的深度真值，可以按照由近及远的顺序将这三个箱子依次移出场景来实现。对于每一个表面，我们取 10 帧的平均值作为真值。然后手动拼接这些深度图像，即可获取这第一类深度缺失像素的真值。

第二类像素是在指对象表面的红外线散斑点受损的前景像素，这些像素点往往在边界附近。这类像素的真值可根据线性回归进行估计。首先手动分割出每个箱

子的矩形前景表面，然后根据每个表面的深度信息进行线性回归。最后根据回归进行线性预测，以估计这些边界像素的真值。因为箱子表面较为平坦，所以这种方法是可靠的。此外，前景像素进一步通过检查两幅彩色图像之间的差异来确定，其中一幅图像有前景箱，而另一幅没有。



(a) 纹理箱子可见光图　(b) 纹理箱子深度图

(c) 纹理箱子深度修复图　(d) 纹理箱子融合图

图 3.7 用于定量精度评价的纹理盒场景

一旦获得深度图像的真值，就可定量评估本文所提方法的准确性。为得到全面的评估结果设计了两个实验，每个实验都会用到一组箱子。如图 3.6 和图 3.7 所示，其中一个实验采用表面颜色一样的箱子，另一个实验采用有复杂纹理的箱子。根据已修复像素的深度的平均绝对误差(MAE)来衡量精度。

表 3.1 不同配置下的定量精度。MAE 单位为毫米(mm)

	彩色箱子			纹理箱子		
h1	1	1	1	1	1	1
h2	0.112	∞	0.112	0.112	∞	0.112
h3	0.045	0.045	∞	0.045	0.045	∞
MAE	19.03	19.97	97.26	20.67	45.73	121.98

根据若干参数的评估，表 3.1 提供了对本文所提方法的量化精度。可以看出，对于彩色箱子和纹理箱子，根据深度融合以及颜色的结构组成，修复结果都比仅仅使用一条线索更为准确。还有就是要修复的表面带有纹理时，精度只会轻微下降。

在彩色表面和纹理表面这两个实验中，由颜色与深度融合策略获得的精度可与 Kinect 本身的精度相媲美，即根据 Khoshelham 和 Elberink 的调查[27]约为 20mm。

此外，对只有一个颜色的平坦表面来说，颜色特征强大到足以实现精确的修复。然而，对纹理表面来说，颜色线索远远不够，误差会成倍增长，这就是深度数据缺失导致的结果，现实世界中的物体，纹理间往往是不确定的，特别是处于同一深度的两个物体边缘。而仅仅基于深度线索进行修复时，MAE 会非常大，显然这是因为对象边界周围的深度信息已受损，将大部分本属于前景的物体深度修复成了背景，而这个差值是非常大的。此外，颜色和深度的连贯性也会为深度图像修复提供可靠的线索。

表 3.1 中所述参数的取值，是在大量实验的基础上所得的较为优异的结果。各个参数的绝对大小没有实际意义。h_1 是距离项的参数，是必须要存在的，反映了相对几何关系。深度信息 h_2 其实是在给远处的信息更大的权重，但是对于近景的精细结构，其实并没有太大的作用。h_3 是彩色信息项的参数，因为彩色信息分辨率高，是补充区域边界最有效的参考部分，因此在整个参考点重要性的排序中占得比重最大。只从数值的相对大小来看，由于指数函数的特点，h_3 值最小，h_2 次之，h_1 最大。

图 3.6 和图 3.7 所示两个实验的误差分布如图 3.8 所示。可以看出，大部分 MAE 都小于 10mm。MAE 的偏差很大一部分来自于那些特别大的误差，比如上文提到的前景物体被修复成了背景物体。这意味着在提高修复精度时，确定深度捕获失败的类型是很重要的。从平均值的角度分析，虽然修复误差大于 20mm 的点所占相对的比重较少，不足 20%，但是由于绝对误差较大，因此对最终的 MAE 影响也至关重要。

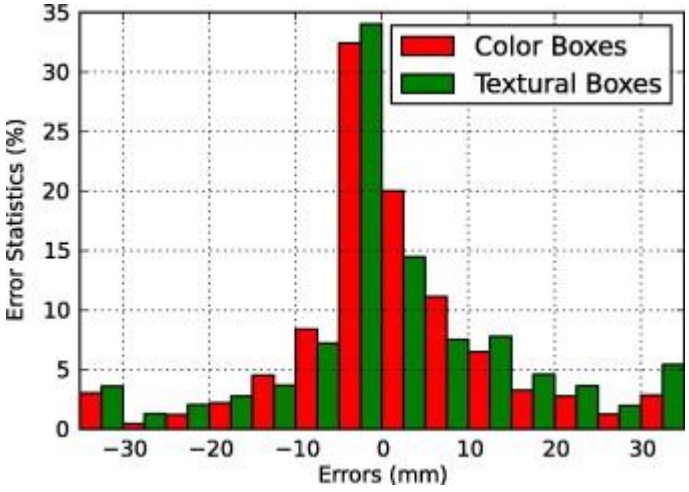


图 3.8 修复误差直方图

根据本小节定量精度分析，3.2 小节所提出的深度修复框架能够很好地平衡深度、颜色信息以及距离信息，并提供了相当准确的修复结果。

3.3 背景信息的应用

3.3.1 应用场合

在网络会议环境中，摄像机拍摄到的场景往往分为前景的与会者与背景的会议室。此时深度图像中缺失的深度将有较大一部分是属于背景场景中被前景人物遮挡的部分。而背景场景中的深度与彩色数据往往很少发生改变。缺失的深度在前景人物进行移动的过程中是可以间歇性地从背景中得到。综合以上因素，可以在人物进场景前进行一次背景的记录，然后进行深度修复的时候，结合这个记录下来的背景信息进行修复。

3.3.2 背景缓冲区模型

设 $d_1...d_n$ 为背景缓冲区中的深度图像。 $p_1...p_n$ 为经过背景缓冲处理后的深度图像。则背景缓冲区的计算方法为:

$$\begin{cases} p_n = \dfrac{p_{n-1} + d_n}{2} \\ p_1 = d_1 \end{cases} \tag{3-12}$$

式(3-12)中，需要特别指出的是，这个计算方法只针对于深度数据中没有缺失的部分，对于 $d_1...d_n$ 中每一幅深度数据中缺失的部分，仍然按照 3.2 小节提出的深度修复框架进行修复。

3.4 彩色图像的修复

3.4.1 修复原因分析

在实际应用中，为了更好的观看立体视频，需要获取场景的双目视角后显示在 3D 电视上。传统的方式是通过两个按照特定角度摆放的摄像头来获取双目视角。但是这种方式成本较高，并且在传输的过程中两路图像源的信息中冗余信息很多。

本文选择使用 Kinect 获取的数据构建立体视频，通过场景视角变换来生成双目视角。但由于视角的变换，会在变换后的图像中产生缺失信息，还需对这些缺失信息进行修复。

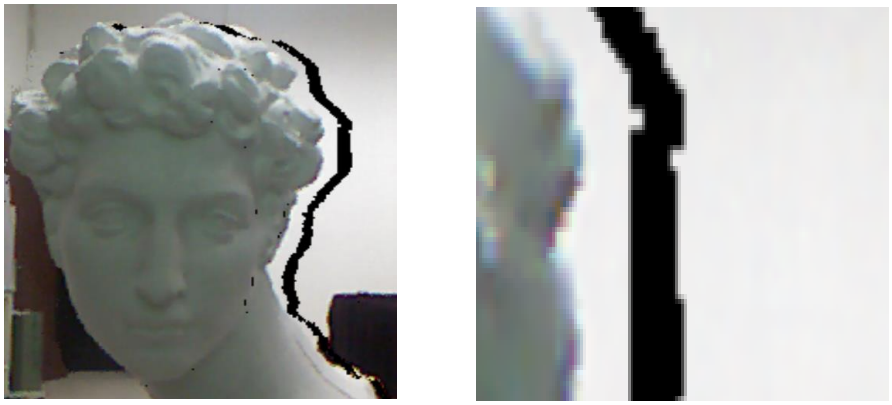
图像细纹修复，顾名思义，就是对图像中小部分的缺失进行修复。但对于较大区域范围的信息缺失，细纹修复的结果会出现不连续、不准确、失真的特征。本文中彩色图像是由于视角变换导致的微小数据缺失，因此可以采用图像细纹修复的相关算法。

3.4.2 修复方法介绍

图像修复技术中比较常见的就是使用基于 PDE(Partial Differential Equation)的算法。这种算法在修复时往往需要进行多次的迭代，来获得修复结果的平稳扩散。如公式(3-13)所示，n 表示修复时迭代的次数，修复点的像素用 x 表示，两者之间满足如下关系：

$$n \propto x^{v/2} \tag{3-13}$$

其中 v 为 PDE 修复算法的系数。不同实现方法的系数 v 不同。如图 3.9 所示是待修复的右视图彩色图像，在 Bertalmio 所提出的方法中 $v=1$，在如图 3.9（a）所示的待修复图像，修复时需要迭代的次数大约为三千次。在 Chan 提出的基于 CDD(Curvature Driven Diffusion)模型的修复方法中，$v=4$，在如图 3.9（b）中所示的修复问题中，待修复的破损区域大约是宽度为18像素的范围，实际中用到的迭代次数需要上千万次。这两种算法的在单帧图像的修复中，计算时间上的消耗是很巨大的。



(a) 待修复的右视图图像　　(b) 待修复图像细节

图 3.9 待修复的图像

在立体视频会议系统中，由于视频实时性的要求，需要快速有效地对图像进行修复。本文选用了 Telea FMM(Fast Marching Method)算法[24]，作为修复双目左右视角中彩色图像的算法。该方法根据图像本身的结构信息，计算参考区域内像素点归一化后的权重系数，并且将修复的顺序与缺失区域的几何结构信息相对应。

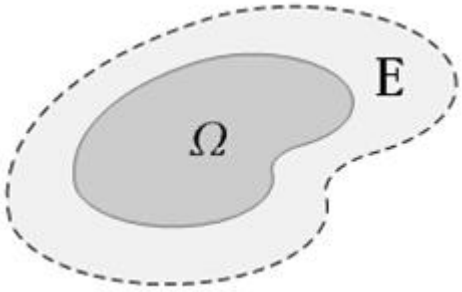


图 3.10 Telea 修复模型中图像缺失示意图

算法的模型如图 3.10 所示，E 表示完整图像的区域，Ω 表示待修复的区域，它包含了所有待修复点的集合，$\Omega=\{p_1, p_2, ..., p_N\}$，$E-\Omega$ 为图像中的已知区域，$\partial\Omega \subset \Omega$ 表示修复边界，是待修复区域与已知区域相邻的边界。在左右视图中，由于生成视图的方法一般是左移或者右移，所以 E 会分布在 Ω 的单侧，具体的修复过程是，对于待修复点 p_k，确定其邻域为：

$$B_\delta(p)=\{p \,|\, |p-p_k|<\delta, \text{且} p \notin \Omega\} \tag{3-14}$$

B_δ 代表与修复点 p_k 的距离小于 δ 的所有点，包括已经完成修复的点。待修复点 p_k 的像素估计值就是 B_δ 中点的加权平均值：

$$u_{FMM}(p_k)=\frac{\sum_{q\in B_\delta(p_k)} w(p_k,q)u(q)}{\sum_{q\in B_\delta(p_k)} w(p_k,q)} \tag{3-15}$$

其中 w 为权重系数，对于离散图像有：

$$w(p_k,q)=\frac{\nabla(p_k)\bullet(p_k-q)}{|p_k-q|^2} \tag{3-16}$$

其中 $|p_k-q|^2$ 代表参考点 q 与待修复点 p_k 在图像坐标系中的距离。
$\nabla(p_k)$ 表示 p_k 点到 q 点的梯度估计值，在连续没有缺失的图像中，可以表示成[24]：

$$w(p,q)=\frac{\nabla(p)\bullet(p-q)}{|p-q|^2}=\frac{\left|\cos\angle(\vec{n}(p),\overrightarrow{p-q})\right|}{|p-q|} \tag{3-17}$$

其中，$\vec{n}(p)$ 是待修复点 p_k 的梯度方向，$\overrightarrow{p-q}$ 为参考点到待修复点的方向。通过这

样的权重设计，可以使得临近待修复点的参考点集合，以及待修复点梯度方向的参考点集合，都可以得到较高的权重。

(a) FMM 修复前图像　　(b) 修复后图像

图 3.11 彩色图像修复效果图

最终的修复效果如图 3.11 所示，图 3.11(a)是修复前的彩色图像，图 3.11(b)是应用 TeleaFMM 算法修复后的图像。修复时按照 RGB 三通道的像素值为零划定待修复区域，修复后没有进行滤波操作。可见修复的效果在背景颜色简单的情况下还是不错的。

Telea FMM 中的邻域与梯度方向的对参考点进行加权平均预估像素值模型，在修复图像中单个像素点时可以根据待修复点周围参考点的信息较为快速的预估出待修复点的像素信息。不过当待修复区域较为广阔的时候，只能由内向外逐步修补，处于待修复区域中心的点修复的时候信息已经丢失了很多，因此会造成模糊。所以 FMM 算法适合修复图像中较小区域的像素丢失。

3.5 本章小结

本章首先对 Kinect 深度缺失的原因进行了分析，在此基础上提出了一种融合结构信息的深度修复框架，主要是在 8 个方向上选取有间距的参考点，选取的同时略过没有深度信息的点，然后融合参考点的距离、彩色、深度信息对待修补点的深度进行加权平均预测。在评估深度修复的效果方面，除了主观的对比与评价之外，还设计了深度真值获取实验来对修补的结果进行定量的评估，评估的角度也仔细区分了深度信息、颜色结构各自的效果。由于三维模型重建后获取左右视图的过程会导致彩色图中出现数据缺失，第三小节中采用了 FMM 方法对彩色数据进行了修复。

第四章 立体视频会议系统的实现

第三章中详细介绍了如何对 Kinect 采集到的深度数据和彩色数据进行修补。本章主要从工程角度，设计和实现了一种三维远程视频会议系统。与第三章相关算法在准确度上的高要求不同，视频会议系统对算法的执行效率更为敏感。因此本章将在第三章深度修复算法的基础上，在工程实现的角度进行设计与优化，相应的修复算法会得到简化，但在可视化和运行速度两个方面得到提升。本章将首先在整体需求上确定系统要实现的目标，然后分别从项目开发运行环境、具体的流程图设计、各个模块的协调运作等方面来介绍系统的实现过程和效果。

4.1 立体视频会议系统实现需求

4.1.1 需求分析

立体视频是近几年来视频处理领域的一大热点。它将传统的平面图像生动地延伸到三维空间，带给观众更为真实和立体的视觉体验。真实的会议场景是立体视频会议室的基础，系统的所有功能都需要在这个场景中完成。

立体视频会议系统采用的显示设备是 3D 电视，方式为光分法中的左右分屏。因此参会者需要佩戴 3D 眼镜，不过系统还需要提供另外一种 3D 模型的展示，用以在没有硬件显示条件下对会议室中的场景进行三维模型展示[28]。

系统采用的设备为 Kinect，采集到的数据为深度数据和彩色数据，因此需要对这两种数据进行三维建模，建模完成后，采用虚拟视点技术来对模型产生虚拟视点下的图像，然后应用双目视觉原理，将虚拟视点下的图像按照 3D 电视所需要的格式进行合成。

4.1.2 具体要求

立体视频会议的主要功能，一是展现真实场景；二是能够支持真实场景下的操作。除了修补的功能之外，还需要提供的功能点如下：

1） 立体视频的采集和处理。需要保证采集的自动化和流畅。

2） 视频的流畅度，为了让用户更好的进行视频会议，并综合目前算法的实现效率，我们要求立体视频的实时播放帧率最低不小于 10FPS。

3） 实时性，要求降低视频在网络传输过程中的延迟。

4） 较普通视频，三维视频的数据量很大，这给视频的网络传输带来问题，因此还需要对三维视频进行压缩。

总体来说，软件需要实时地采集 Kinect 设备的彩色数据与深度数据,同时设计算法进行 3D 模型的立体建模和修复,并采用并行计算来提高算法的处理速度，在进行压缩与网络传输后，进行可视化的立体显示。系统需要包含可执行的软件,包括设置界面、视频显示界面等。软件并不区分服务端与客户端,每一个软件实例既包含服务端也包含客户端，互为 C/S 结构。

4.2 项目开发环境与第三方库

4.2.1 开发与运行环境

项目采用 Linux 为开发所用的操作系统，具体的发行版为 Arch Linux。桌面环境为 Gnome。Arch Linux 最早起源于加拿大，是一份致力于软件更新速度快并且高度自定义的 GNU/Linux 发行版。它对 i686 架构以及 x86_64 支持都很好，并且还派生出了针对 ARM 平台的 Arch Linux ARM。Arch Linux 比较显著的特点就是滚动升级，发行版本身没有版本的概念。这就保证了大部分的软件库都是比较新的版本，使用者可以尝试最新的特性。另外 Arch 特有的 ABS (Arch Build System)是一种用于从源代码自定义编译软件的系统；以及 AUR(Arch User Repository)用户自定义软件版本库，这两者是 Arch 这个发行版的灵魂。基于这样的特性，我们可以定制功能强大的开发环境。

项目中采用 QT Creator 作为编写与调试项目代码所用的 IDE(Integrated Development Environment)。使用的开发语言为 C++，编译代码所使用的工具链为 GCC 4.9.2 版本。

GCC（GNU Compiler Collection）是一个由 GNU 组织开发的可以支持多种流行编程语言的编译工具集合。大多数的类 UNIX 操作系统（比如 Linux、BSD、Mac OS X 等）都选择 GCC 为标准编译器，同时，GCC 也适用于微软的 Windows 系列操作系统。最开始的时候，GCC 只能处理 C 语言，但是它发展很快，不久就支持处理 C++，并且越来越多的编程语言，如 Fortran、Pascal、Objective-C、Java、Ada、Go 等，也慢慢得到了支持。

项目的运行环境是 Windows 和 Linux。暂时未在 Mac OS X 进行调试。在不同平台上运行时需要安装不同的运行时环境。

4.2.2 第三方库介绍与选用原因

立体视频项目本身对于处理速度和显示效果都有比较高的要求，同时考虑到软件之后的应用场合，需要一种跨平台的运行环境。考虑到这些需求，本项目采用QT 作为项目的基本库。

QT 是一个跨平台的基于 C++语言编写的带有用户图形界面的应用程序框架。最早在 1991 年由奇趣科技开发，后几经收购，目前是 Digia 公司的开源产品。QT 可以提供一个相当美观的软件界面设计效果，同时 QT 也比较容易扩展，遵循组件编程的思想。QT 最大的优势还是在于它的跨平台特性，应用程序的开发人员使用同一个类库进行开发的 QT 应用程序，甚至是包含界面的程序，都可在各种服务器、桌面环境或者是嵌入式系统之间进行无缝移植[29]。

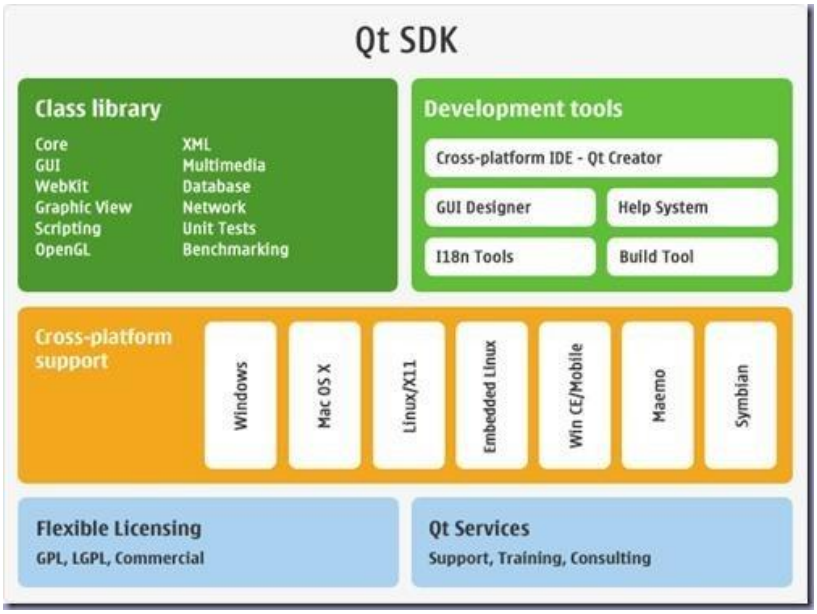


图 4.1 QT 架构说明[29]

对比来说，QT 是和 Linux 中 X Window 上的 GTK 图形界面库以及 Windows 平台上的 MFC、ATL 是同类型的东西。但是 QT 有着其他工具所不具备的特性如下：

1）优秀的跨平台特性。QT 所兼容的操作系统包括但不限于 Linux 及其各种发行版、Microsoft Windows 系列、Solaris、HP-UX、FreeBSD、AIX 等等。

2）组件封装设计。QT 优良的封装机制使得 QT 内部的模块化程度非常高，你可以在项目的代码中只集成用到的 QT 模块，而不需要加入整个 QT 的运行环境。同时 QT 提供了一种称为信号/槽的安全类型来替代传统的回调函数和消息队列机制，这使得各个组件以及各个线程之间能够非常简单地进行协同工作。

3)丰富的 API。除了 QT 的核心组件 Core 之外，QT 还包括了网络部分 WebKit、

Network 库,图形部分 Graphic View、OpenGL 的支持，同时也对底层的数据封装，比如 XML、JSON 等提供了非常方便的使用。

4)高效能的多线程库。QT 的线程库是独立于操作系统平台的，在不同的平台中均有着不同的、性能优异的实现。这使得 QT 能够充分地利用处理器中多核的优势，获得物理硬件上最佳的性能，并能根据可用的处理器内核数目自动地调整计算中所使用的线程数。

但以上只是 QT 中一部分比较有代表的特性，而 QT 本身是对 C++语言的一个集大成者，区别与 Windows 中 MFC 的消息响应机制，QT 的信号与槽是更为优雅和高效的实现。信号和槽是 QT 自行定义的一种用于对象之间的通信高级接口，主要由信号、槽和元对象工具组成。槽函数可以认为是与特定的信号相绑定的普通 C++成员函数。当与槽函数绑定的信号被发射时，该槽函数就会被触发，而绑定的方式也分为多种，比如线程内绑定，线程外阻塞绑定等等。元对象工具可以对 C++文件中的类声明进行分析，然后将对应的信号与槽函数的定义转换为相应的 C++初始化代码。

本项目中采用的 QT 版本是 5.3.2。

4.3 系统的框架流程与模块设计

4.3.1 系统组成框图与运行流程

为了方便单元测试与开发，我们将系统分成了 5 个模块，分别是设备驱动模块、修复算法模块、网络通信模块、视图生成模块、显示模块。主程序依次调用各个模块来完成系统功能。

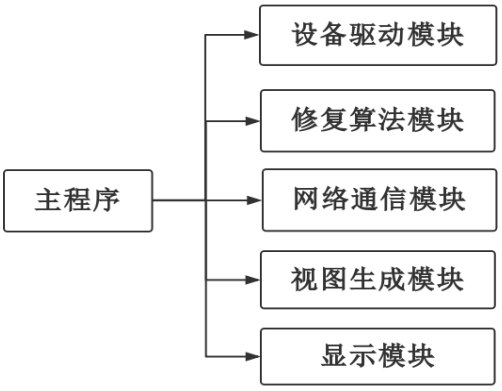


图 4.2 系统模块分解示意图

如图 4.2 所示，图中的五个模块是五个独立的 QT 项目，项目的输出设置为库的形式，即每一个模块在编译后都会生成一个扩展名为 so 的文件(Linux 系统)或者是扩展名为 dll 的文件(Windows 系统)。主程序在编译的时候需要指定这些库文件的路径，否则无法正常编译。而在需要更新的时候，也只需要重新编译被改动的模块即可，不需要将整个项目的代码都进行重新编译。

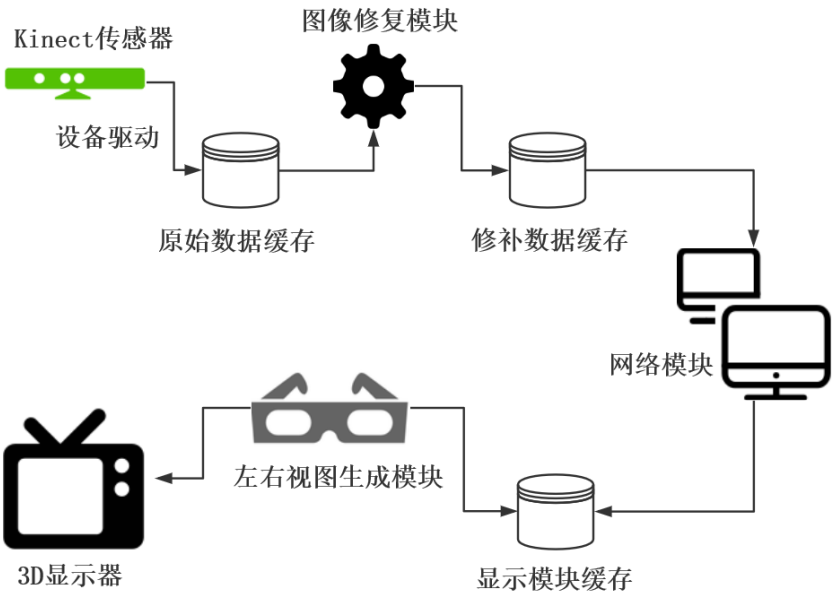


图 4.3 系统各模块运行流程示意图

各个模块的执行顺序如图 4.3 所示。每个模块均是一个单独的线程，主程序启动后，会依次启动各个模块的线程。数据流从设备驱动模块开始，各个模块之间的数据，均通过高速缓冲区进行共享。图像修复模块会将数据帧从设备缓存取出，进行修复计算，然后放入到修补缓冲区中。网络模块会从修补缓冲区中取出数据发送。同样，在网络的另一端，网络模块在接收到数据并进行重组后会将数据放入显示数据缓冲区。显示模块的线程将从显示模块的缓冲区取出数据帧进行三维重构，并生成左右视图发送到 3D 显示器中进行显示。

4.3.2 设备驱动模块设计

设备驱动模块的主要功能是获取 Kinect 设备的数据，进行格式的统一，然后放入缓存中供其他模块的调用。本模块采用了第三方库 OpenNI(Open Nature Interface)作为底层的依赖库。

OpenNI 是一个支持多种体感设备的多语言、跨平台的应用程序框架，它定义了一套用来进行自然交互的应用程序 API(Application Programming Interface)。

OpenNI 的主要目的是设计一套标准的 API，来搭建视觉和音频传感器与视觉和音频感知中间件通信的桥梁。维护 OpenNI 的是一个非营利性组织，该组织创建于 2010 年，Prime Sense 公司是其成员之一，也是 Kinect 的核心芯片的提供者。

OpenNI 提供了一组基于传感器硬件，并由中间件组件实现的编程接口，这就消除了传感器和中间件之间的依赖，使用 OpenNI 的 API 编写的应用程序可以忽略不同平台带来的兼容性问题。OpenNI 的 API 使得开发人员可以通过传感器输出的标准化数据来处理真实的三维数据（标准化数据的类型包括了人体全身、手的位置，甚至是一个含有深度信息的图像等）。现如今，OpenNI 已经成为 Kinect 在 PC 上开源驱动必须安装的一个组件，本文使用 Kinect 来获取场景中彩色图和深度图是通过 OpenNI 框架读取的，从而为后续工作的开展打下基础，选用的版本为 OpenNI2.2。具体读取图像数据的方法如下：

1）初始化设备

调用静态方法 openni::OpenNI::initialize()，此方法返回一个类型为 openni::Status 的变量，表示设备是否初始化成功。在启动失败的时候，可以通过 getExtendedError()接口来获取启动失败的原因。启动成功后，可以通过调用 openni::Device 类型变量的 open 方法来打开设备。

2）设置数据流类型与参数

在设备成功打开后，可通过 openni::VideoStream 类型变量的 create 方法来打开设备的流数据，第一个参数是打开设备时所用的 openni::Device，第二个参数就是数据流的类型。可选的类型有 openni::SENSOR_DEPTH(代表深度数据)和 openni::SENSOR_COLOR（代表彩色数据）。每种数据流还可以通过接口来设置数据流的参数。setFps 接口设置流数据的帧率，setResolution 接口设置流数据的分辨率，setPixelFormat 接口设置流数据的格式。其中深度数据和彩色数据的格式分别为 openni::PIXEL_FORMAT_DEPTH_1_MM 和 openni::PIXEL_FORMAT_RGB888。

3）产生数据

设置好流数据的类型以及参数后，就可以调用 openni::VideoStream 类型变量的 start 方法来产生数据。

4）读取数据并处理数据

在数据流处于打开状态时，可以通过 openni::VideoStream 类型变量的 readFrame 接口来主动地获取数据。数据帧的类型为 openni::VideoFrameRef。深度数据的格式为 openni::DepthPixel，通过 getData 方法获取。每一个数据帧都可以通过接口 getTimeStamp 来获取这一帧数据对应的时间戳，用于后续的深度和彩色数据同步。

虽然通过 OpenNI 框架可以方便地读取 Kinect 设备所提供的数据，但是还有

一个必要的前提，那就是安装驱动。在 Windows 平台中，可以采用 Microsoft Kinect SDK。在 Linux 平台中，可以采用 Github 中的开源项目 libfreenect。由于驱动获取数据格式的不一致性，在本模块中将彩色图和深度图格式进行统一，定义了一个全局可使用的结构体 DepthFrame 以及 VideoFrame。

```
struct DepthFrame{
    unsigned short* data;
    int timestamp;
    int timestampUTC;
    int isEmpty;
};
```

```
struct VideoFrame{
    unsigned char* data;
    int timestamp;
    int timestampUTC;
    int isEmpty;
};
```

结构体中的 data 代表深度图和彩色图的二进制数据。Timestamp 代表从设备获取的时间戳，这个时间戳是从打开设备开始后的时间。另外一个 timestampUTC 是指当前世界的时间戳，是指从 1970 年 0 时算起到当前时间所经过的秒数。

从 OpenNI 的接口可以看出，Kinect 设备是以流的形式产生数据，而深度数据和彩色数据是两个流，这就有可能导致数据时间不同步的现象。因此需要对两个流的数据进行同步后才可放入缓冲区中。缓冲区数据的格式为:

```
struct ThreeDFrame
{
    int ID;
    unsigned char* color;
    unsigned short* depth;
    int color_device_timestamp;
    int depth_device_timestamp;
    int color_timestamp;
    int depth_timestamp;
    int timestamp;
    int type;
    int state;
};
```

两种数据流的同步原理就是先根据两个数据流中当前帧的时间戳确定出同步状态 type，计算方法如式(4.1)所示，然后再根据 type 的值进行不同的处理。

$$type=\begin{cases}1 & |T1-T2|>100\\ 2 & 5<|T1-T2|<100\\ 3 & |T1-T2|<5\end{cases} \tag{4-1}$$

type 为 1 代表两个数据流的时间戳相差过大，这时需要清空两个数据流中的设备缓存。type 为 2 代表两个数据流存在时间差，但是这个差不大，则让时间相对靠前的数据流丢弃当前帧的数据，这也是实际运行中最经常遇到的情况。type 为 3 代表两个数据流的时间差在允许的范围内，可以认为两个数据流是同步的，则将深度数据帧和彩色数据帧合成一帧三维数据帧，加入到与修复模块共用的缓冲区中。

4.3.3 高速缓冲区设计

由系统的整体框图可以看出，整个系统的设备驱动模块、修复模块、网络传输模块、显示模块这几个核心模块之间，都存在有数据的交互。为了提高运行的效率，通常的办法就是设计缓冲区来进行数据的交互。缓冲区就是单独开辟一块内存空间来存放共有的数据集，在多线程环境中，也被称作临界区。需要注意的是，各个模块其实都是以单独的线程来运行。因此缓冲区设计的效率和安全性都是需要注意的问题。

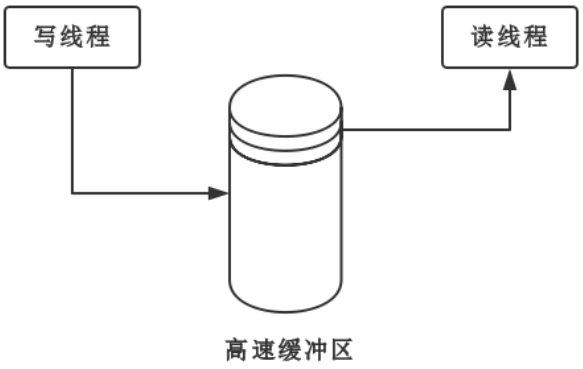


图 4.4 高速缓冲区工作原理示意图

如图 4.4 所示，写入模块将数据写入缓冲区中，读出模块将数据读出缓冲区，而缓冲区中的数据大小是不固定的。通过这种设计，程序会自动根据写入模块和读出模块的数据情况调节他们各自的运行效率。这种设计使得两个模块的速度互相

之间减少了牵制，速度在过渡时相对平滑。不会因为一个模块数据处理速度的陡然变化导致另外一个模块速度的骤然改变。另一方面，这个设计也可以增加整个系统处理的并发性。比如对于同一个缓冲区，可以设计多个读模块，各个读模块在获取数据的时候可以是并发进行的，而不需要相互等待，只需要处理好跟写模块的先后关系即可。同时对缓冲区大小的设定，也可以调整系统的运行效率。

缓冲区的设计方法有多种，具体可以分为固定缓冲区和动态缓冲区。固定缓冲区，顾名思义，就是在程序初始化的时候就为缓冲区分配固定大小的空间。在程序运行期间这部分缓冲区不会被释放，而是每次都会被覆盖，这样做的好处就是避免了系统每次分配和回收内存的消耗，缺点就是缓冲区的大小是固定的，如果要修改缓冲区大小将是一件很麻烦的事情。而动态缓冲区不会有这样的限制，它的数据是不断地分配并释放的，因此往往没有大小的限制，随之而来的就是运行效率的低效，因为系统分配和释放内存是一种操作系统底层的调用，频繁的调用会非常消耗系统的资源。

本文使用的是固定缓冲区，并采用环形队列来实现。示意图如图 4.5 所示：

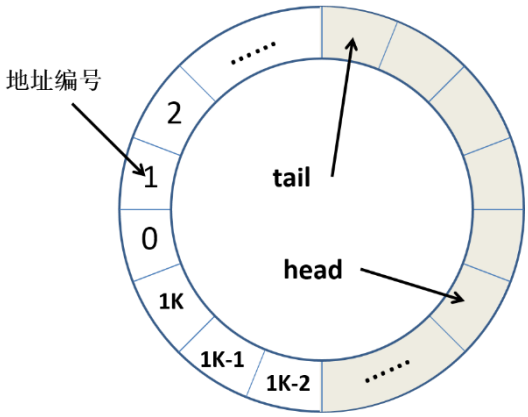


图 4.5 环形缓冲区示意图

环形队列拥有两个指针，头指针 head 和尾指针 tail。在实际的使用中，对应为读指针和写指针。考虑到项目中缓冲区的实际情况，本文中设计的缓冲区只有一个写指针，但是可以有多个读指针。那么当队列中没有数据的时候读指针和写指针指向同一片数据。写入数据时，写线程会将数据写入当前位置后将写指针向后移动一位。读取数据时，读线程会读取读指针当前指向的数据，并在读取数据后，将读指针向后移动一位。当读线程发现读指针与写指针相同时，读线程会忽略此次读操作，

让线程本身睡眠一段时间，以防止 CPU 满载。

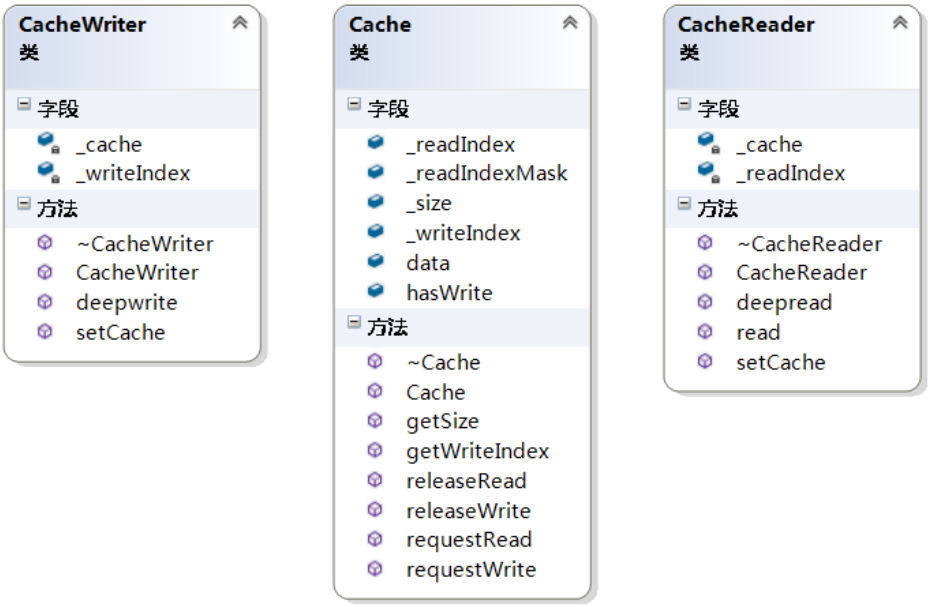


图 4.6 高速缓冲区类设计图

综合上述考虑，最终设计出的缓冲区的类视图如图 4.6 所示。Cache 类是核心的类，用来管理缓冲区中的数据。_readIndex 代表读指针,_writeIndex 代表写指针。CacheWriter 类是负责向 Cache 中写入数据的类，它的构造函数需要一个 Cache 类型的对象，在初始化时，会调用 Cache 类的 requestWrite 方法来获取 Cache 类的写指针方便后续写入。CacheReader 类是负责从 Cache 中读取数据的类，它的构造函数同样需要一个 Cache 类，但是会调用 Cache 类的 requestRead 方法来获取 Cache 类的读指针。CacheWriter 的 deepwrite 方法会将一帧 ThreeDFrame 格式的数据写入缓冲区中。CacheReader 分为 read 方法以及 deepRead 方法，不同之处在于对于指针数据的处理，read 方法只是读取一个 ThreeDFrame 类型的指针，而 deepRead 方法会将 ThreeDFrame 类型的数据完整地读取出来。

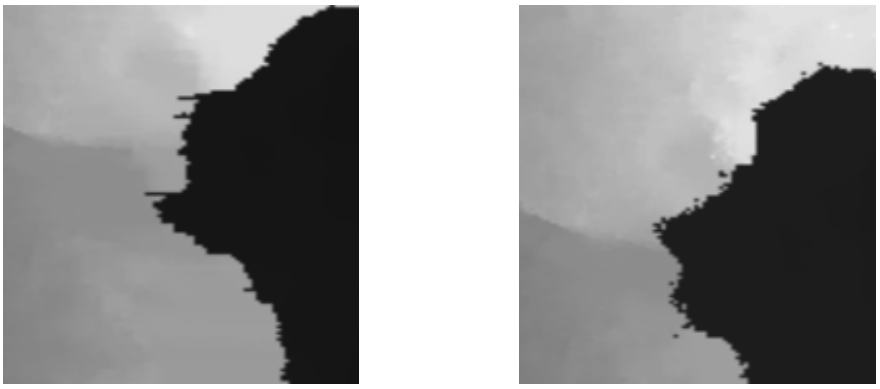
4.3.4 修复算法模块设计

1)模型的简化

修复模块是在第三章算法基础上的一个工程实现，并考虑到速度与可视化效果，对算法进行了一些精简，主要是对彩色数据的灰度化处理，并且淡化了深度数据在待修补点选择中的作用。作为替代，深度数据更多地用在了背景的判断中，这对修补的效果和速度均有很大的提升。

2) 对修复顺序的优化

在实施修复算法的过程中，首先是要确定哪些点是需要修复的，一般的步骤就是按照行优先的顺序来进行索引，判断对应矩阵中该点的深度是否为零。通过这样的操作可以得到待修补的下标队列。后面进行修补的时候，按照这个队列的顺序进行逐点修复。但是这种修复顺序存在这样的情况，后修复的点会比先修复的点拥有更多的参考点，先修复的点也会作为一个参考点供后面的点进行修复。这就会导致在视觉效果上，按行的特征会特别明显。如图 4.7（a）所示，修复的结果中会出现横向的条纹，原因就是在先修复图像中偏左边的点，而右边的点又参考了左边点的修复结果。



(a) 行优先索引　　(b) 随机索引

图 4.7 修复顺序对修复效果的影响

为了解决这个问题，可行的一种思路就是打乱初始得到的索引队列，得到一个看似随机的索引，修补的顺序按照此随机序列进行。通过对比发现，加入随机索引后的修复效果毛刺更少，后续可以通过滤波的方法进行平滑，得到更佳的可视化效果。

4.3.5 网络传输模块设计

视频会议系统的双方不在同一地点，那得到的立体视频数据必须要通过网络来进行传输。考虑到实时性与可靠性的因素，本系统采用 RTP(Real-time Transport Protocol)协议来进行立体视频的网络传输。具体的数据包封装、压缩、解压、重组本文不进行详细说明，主要介绍类中接口的设计，如图 4.8 所示。网络模块包括一个读操作器 m_netReader(继承 CacheReader)，以及一个写操作器 m_netWriter(继承 CacheWriter)。对于上层模块，m_netWriter 将立体数据写入网络模块的缓冲，并被网络模块发送；m_netReader 则可以从网络模块接受的数据中读取立体数据进行后续处理。

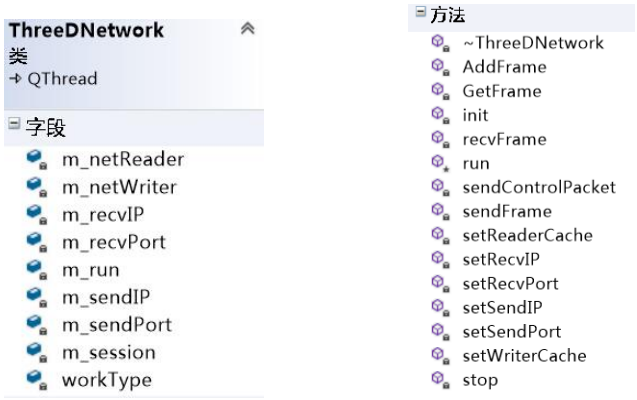


图 4.8 网络模块类设计示意图

4.3.6 立体成像模块设计

立体数据经过修复后，会送达显示模块进行显示。本系统包含两种方法对立体视频数据进行显示：左右视图配合 3D 电视进行立体展示，OpenGL 的 3D 模型显示。

OpenGL（Open Graphics Library）是一个支持多种编程语言、操作系统平台的应用程序接口的规范，它可以用来处理、制作、生成平面或者立体图像。这个接口中包含了几百个不同功能的函数集合，可以处理多种图像中常见的操作，比如从简单的图形数据绘制到复杂的三维景物重建。Microsoft Windows 上的 Direct3D 是另外一个与 OpenGL 拥有类似功能的组件，与之相比，OpenGL 更常用于游戏制作、虚拟现实等领域。

我们的项目中采用了 QT 的 QGLWidget 进行模型的 3D 显示。从摄像头的视角确定 OpenGL 坐标系。通过深度数据来确定视图中点的坐标信息，然后通过绘制纹理图的方法来给场景中的点进行染色。关键处代码为：

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_SHORT, 0, xyz);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(3, GL_SHORT, 0, xyz);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, gl_rgb_tex);
glTexImage2D(GL_TEXTURE_2D, 0, 3, 640, 480, 0, GL_RGB, GL_UNSIGNED_BYTE, rgb);
```

视角的改变是通过鼠标的拖动来实现的，并通过键盘事件来控制模型的远近以及大小。具体的方法是捕捉鼠标拖动的相对位移 dx、dy。然后通过接口函数

glRotatef 来设置视角的旋转。第一个参数即为移动的大小，后三个参数对应移动的位移。X 轴方向的为(1,0,0)；Y 轴方向的为(0,1,0)。

4.4 计算加速

目前计算机中的计算单元主要包括中央处理器 CPU(Central Processing Unit)和图形处理器 GPU(Graphic Processing Unit)。而现场可编程门阵列 FPGA(Field Programmable Gate Array)目前仍然处于高速发展的阶段，优点是功耗低，但是还没有在常见的计算任务中得到普及。本文中提到的算法以及软件，均在普通个人计算机中实现，因此对计算的加速主要分为 CPU 中多线程的加速以及 GPU 加速。

4.4.1 多线程加速原理分析

对于 CPU 计算能力的发展，在过去相当长的一段时间中，Intel、AMD 等处理器制造厂家推出的 CPU 型号，其主频在不断攀升，但是相对增长的速度却与之前相比下降很多。如果按照摩尔定律，每两年之内，处理器的核心芯片上可集成的晶体管数量就会翻倍。实际的情况是，目前 Intel 系列 CPU 单核心的主频，停留在 3GHz 左右，上升到 4GHz 及其缓慢，更不提 6GHz。而从另外一个方面来看，处理器的核心数目，从 2 核到 4 核，从四核再到八核甚至 16 核，核心数目的提升，大大提高了处理器整体的计算效率。

此外，在增加处理器核心数目的同时，Intel 公司还发明了一种超线程技术[30]。它的原理并不复杂。在此之前的单核心处理器，在同一时刻只能运行一个线程，或者说只能执行一项任务。如果要处理不同的线程时，需要进行上下文切换，而这个切换的代价是十分昂贵的，包括线程变量的压栈和弹栈，系统资源的重新分配与调度等等。超线程技术就是为了解决这个问题，它增加了上下文切换的效率，使得一个处理器核心可以拥有两个线程的上下文环境，进行高效的切换，从而效能有约百分之二十至三十增长。

硬件的升级必然带来软件的革命。既然处理器的核心数目慢慢变多，而且每个核心又有超线程技术可以加速，那如何开发一个可以高效利用 CPU 计算资源的程序，越来越受到程序开发者们的重视。相应的多线程库，比如 POSIX 的多线程库，开始被广泛的使用。就连 C++11 的新标准中也不得不把多线程模块进行重新设计来提高 C++在多线程方面的表现性能。本系统中使用了 C++作为开发语言，并采用了 QT 库作为底层依赖，值得一提的是，QT 中 QThread 在 Linux 平台中就是以 POSIX 的多线程库作为实现版本。

在应用具体的加速算法之前，需要了解常见的多线程编程技巧，其中最为重要的一点就是：如何在线程安全性与线程调度效率两个方面得到权衡。

4.4.2 多线程加速深度修复

在深度修复的描述章节中，我们提到深度数据的修复是逐点进行的，在一帧 640×480 分辨率的图像中，待修补的区域中往往至少有 3000 个深度缺失点，这些点在修复时互相之间的联系很少。可以这样说，如果两个点的距离隔着较远，大于了算法中搜索窗的半径，那么这个点的修补顺序对于最后的修补效果是没有任何影响的。这就满足了多线程并行修补的基本前提。

本文采用了 QT 中的 QtConcurrent 库来进行多线程修补，在 QtConcurrent 库中集成了包括 map,filter 等等系列函数，提供了多线程中对一个函数的调用以及结果的汇总。这一系列的函数其实是对 Map/Reduce 并行计算思想的一种实现。并行计算的一个非常重要的解决思路就是采用大量的计算节点，将庞大的计算任务分割成为每个节点可以接受的细小计算规模，最终再将计算的结果合并起来。这就需要进行并行计算问题本身是必须可以被分割的。另外，如何将计算任务自动地分配到节点上，这需要一个合理的框架。执行并行计算的函数为:

```
QtConcurrent::map(compenList,wrapper)
```

其中 compenList 为需要修补点的下标集合，范围 $0\sim640\times480$。Wrapper 是对修补函数的加载类，它会自动地针对 compenList 中的每一个元素并行地调用修补函数。然而，仅仅对 compenList 修补一次是不够的，因为某些周围空洞区域较大的待修补点，可能第一次修补的时候，会由于参考点不足而不进行修补。因此需要多次并行调用修补函数。第二次修补前，需要对 compenList 中的元素做一个判断，将已经被修补的元素剔除掉。过滤函数为：

```
QtConcurrent::filter(compenList,allowCompen)
```

其中 compenList 依然为待修补的点集合，不同之处为，这个函数运行结束后会修改 compenList 中的元素，判断的依据就是 allowCompen。allowCompen 本身是一个函数，定义如下：

```
bool allowCompen(const mPoint &point){
    return (point.mask == 1);
}
```

QtConcurrent::filter 函数会针对 compenList 中的每一个元素并行的自动调用 allowCompen 函数，而返回值为 true 的元素会继续保留在 compenList 中，返回值为 false 的元素就会从 compenList 中消失。

表 4.1 线程数目与加速结果关系图

最大线程数目	1	2	3	4
单帧修补时间	227ms	150ms	99ms	75ms
最大线程数目	5	6	7	8
单帧修补时间	68ms	63ms	58ms	58ms

如表 4.1 所示，设置不同最大线程限制时单帧修复的时间不同。多线程编程的另一个重要问题是究竟同时运行多少个线程最为合适，其实这个最优线程的数目是跟实际 CPU 的处理核心数目有关的，考虑到上文提到的超线程技术，因此在支持超线程的 CPU 中，推荐的线程数是 CPU 物理核心数目的 1.5 倍左右。即对与一个物理 4 核心的 CPU 来说，以同时运行 6 个线程为效率最佳。虽然通过超线程技术可以达到 8 核的数值，但是同一个核心中的两个超线程调度性能还是不如两个物理核心之间互相不干扰的效率高。在 QtConcurrent 库中，会自动根据 CPU 的物理核心数来设置系统中 ThreadPool 的上限大小。

4.4.3 图形处理器计算加速介绍

图形处理器 GPU，通常也被成为显示核心或者视觉处理器，是一种专门在个人电脑和一些移动设备，比如平板电脑、智能型手机等上进行图像运算的微处理器。经过 NVIDIA 公司几代的设计，图形处理器不仅使显卡核心减少了对中央处理器的依赖，而且反而能够分担很多由 CPU 来进行计算的工作，特别是在进行三维图形处理、矩阵计算等的时候，加速效果十分显著。

2007 年 6 月，NVIDIA 公司推出了一个计算设备架构 CUDA(Compute Unified Device Architecture)，它是一种将 GPU 作为并行计算设备的软硬件体系。在 NVIDIA 公司的强力推进下，CUDA 的版本不断更新，目前已经到了 6.0 版本，增加了许多计算的特性以及对新的 GPU 核心的支持。而与以往的 GPU 相比，支持 CUDA 的 GPU 在架构上也有了显著的改进，特别是 NVIDIA 核心与 AMD 核心的显卡区别，后者只能依赖于 OpenCL 进行并行计算。

4.4.4 模型重建加速的 CUDA 核函数

CUDA 的优势在于矩阵计算，而点云数据的构造以及左右视图的生成均涉及大量的矩阵计算，因此本系统采用自定义 CUDA 核函数的方法来对点云生成与左右视图生成进行加速。运行在 GPU 上的 CUDA 并行计算函数被称为 kernel（内核函数）。一个 kernel 函数并不是一个完整的可执行程序，而是整个 CUDA 程序中的

一个可以被并行化执行的步骤[31]。

CUDA 将 GPU 的计算模型设计为三层结构，即 Grid、Block、Thread 三层结构。Kernel 以线程网格（Grid）的形式组织，每个线程网络由若干个线程块（block）组成，而每个线程块又由若干个线程（thread）组成。实质上，kernel 是以 block 为单位执行的。

根据 CUDA 计算的模型，将第二章中视图生成的计算进行加速。具体的加速函数为 generatePointCloud 与 generateViewData。

重建点云数据的过程中，需要根据深度数据(u,v,d)与彩色数据(u,v,r,g,b)构造点云数据（x,y,z,r,g,b,d）。由于图像大小为640×480。因此我们将 Grid 分为64×48个 block。再将每个 Block 分解成10×10的 Thread。最终每个 Thread 中仅仅针对一个像素点进行点云重构的过程。而整幅图像中的640×480个像素点之间几乎都是并行计算。对于每一个 Thread 中进行计算的图像下标求解方法为：

$$\begin{cases} index = bid.x*10 + tid.x + (bid.y*10 + tid.y)*gid.x*10 \\ row = index/640; \\ col = index - row*640; \end{cases} \tag{4-2}$$

为了对比 GPU 加速前后视图显示速率的差距，我们统计了在不进行模型修复的前提下，左右视图显示的帧率，如表格 4.2 所示。进行 GPU 加速后，用于视图转换计算的时间可以忽略不计，视频的显示速率接近于设备产生数据的原始帧率。

表 4.2 GPU 加速前后显示帧率对比

是否加速	否	是
显示帧率	15FPS	30FPS

4.4.5 利用核函数参数预加载进行修复加速

在公式(3-6)中，h_1、h_2、h_3分别代表了距离、深度、彩色三个因素对参考点选取的影响因子。在实际的计算中可以发现，s_1的值基本是固定的，可以预先进行计算放到一个哈希表中，这样每次在修复时只需要一个索引就可以得到参数的值，相当于用空间换时间，提高了计算的效率。

$$disHash[i][j] = e^{-\frac{i^2+j^2}{2r^2}} \tag{4-3}$$

在公式(4-3)中，$disHash$是对因子 s_1 的一个预计算。i、j 是参考点到待修补点在行与列上坐标的偏移，r 代表搜索矩阵的半径。

同时，本系统的修复算法模块对第三章中结构导向的修复算法进行了简化，将彩色信息简化为灰度信息。从三通道的彩色信息差值下降为单通道的灰度差值，数据的范围也降为255^2。因此可以将灰度差值信息也进行一个预计算。如公式(4-4)所示:

$$grayHash[i][j]=e^{-\frac{(i-j)^2}{2*255^2}} \tag{4-4}$$

4.5 系统实现效果

4.5.1 系统主界面

如图 4.9 所示，是远程视频会议系统的初始化配置界面，共分为设置区、操作区、监控区三个部分。设置区包括图 4.9 所示的 Process Setting 和 Network Setting 两部分。其中 Process Setting 包括的设置选项有计算类型(ComputeType)、检测类型(Detection)、修补策略(Compen)、显示源(ViewSource)。计算类型包括 GPU 和 CPU 两种，特指左右视图生成部分，是采用 GPU 计算，还是采用 CPU 计算。检测类型是指是否在运行过程中进行人体部位检测。显示源可选网络或者本地，如果选择网络，则显示部分将显示从网络传输过来的立体数据，如果选择本地，显示部分将显示本地的立体视频数据。Update 按钮表示是否更新深度背景，Inpaint 按钮表示是否对生成的左右视图的彩色数据进行修补。

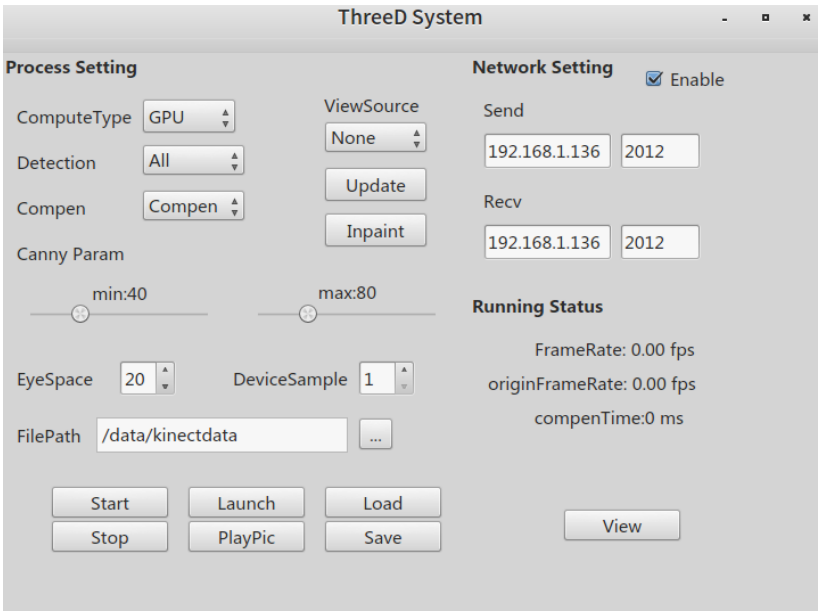


图 4.9 系统主界面设置截图

操作区的主要按钮包括 Start、Launch、View 三个功能。Start 是指开始程序的运行，会启动网络模块、图像修补模块、显示模块。Launch 是启动设备，会开启设备驱动模块。View 按钮会启动一个显示实例，将显示模块中渲染好的图像进行显示，为了方便显示与对比，显示的实例可以开多个，每一个实例可以指定一种显示类型。

4.5.2 各类型视图界面介绍

视图界面是指立体视频制作过程中各个步骤出现的不同类型的视图，与主界面中 View 视图的各个类型相对应。包括原始深度、原始可见光图像、修复后深度、OpenGL 的三维模型、左右视图。

(a) 原始深度图　　(b) 原始彩色图

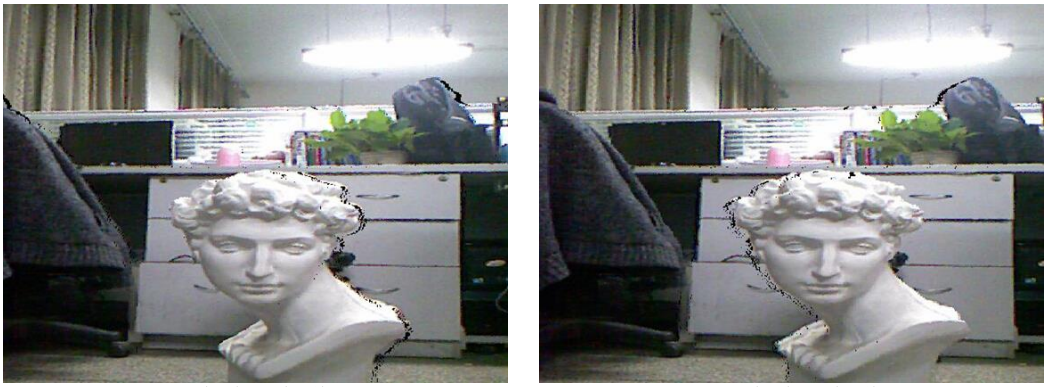
图 4.10 Kinect 原始数据视图

如图 4.10 所示，分别是 Kinect 采集到的原始深度图和可见光图像。其中图 4.11(a)是经过配准后的深度图，其深度数据与可见光图像中的彩色数据是一一对应的。画面中包括常见的会议室背景，并使用一个石膏像作为使用者进行演示。

(a) 修复后深度图　　(b) OpenGL 模型截图

图 4.11 修复后的深度图与重建的模型截图

图 4.11(b)是将可见光数据与深度数据进行点云重建后得到的三维场景模型的截图，在生这个模型后，需要设置两个虚拟摄像机，来对模型进行虚拟摄像，得到左右视角的可见光图像，如图 4.12 所示，图 4.12(b)是左视图图像，模拟人的左眼观看场景的效果，图 4.12(a)是右视图图像，模拟的是人的右眼观看场景的效果。需要特别说明的是，这个左右视图并不是直接从 OpenGL 的模型中拍摄出来的。



(a) 右视角虚拟摄像图　　(b) 左视角虚拟摄像图

图 4.12 从模型虚拟摄像得到的左右视图

从图 4.12 可以发现，生成的左右视图由于视角的变换，会产生彩色信息的缺失。在图 4.12(a)所示的右视图中，由于视角向右平移，因此新的视角会看到原先视角所看不到的物体表面信息，也就是石膏像右边黑色的点所表示的区域；同理，4.12(b)图中是左视角视图，缺失的物体表面彩色信息就在石膏像的左边。

按照 3.4 小节的方法进行修复后的结果如图 4.13 所示，从视觉效果上看，原图中存在的黑点被很好的修复，基本还原了周围的纹理特征并且具有连续性，修复的结果提高了视图的视觉效果。

(a) 右视角修复图　　(b) 左视角修复图

图 4.13 修复后的左右视图

然而仅仅对左右视图进行修复是不够的，还需要把左右视图编码为 3D 电视可以播放的格式，具体的做法也比较直接，就是分别将左右视图在横向进行缩放，即

保持高度不便，宽度为原来的二分之一，然后将缩放后的两张图像拼接成一张图像。这样拼接后的图像大小尺寸上与原先的图像一致，但在内容上，左半边为左视图缩放后的图像，右半边为右视图缩放后的图像。

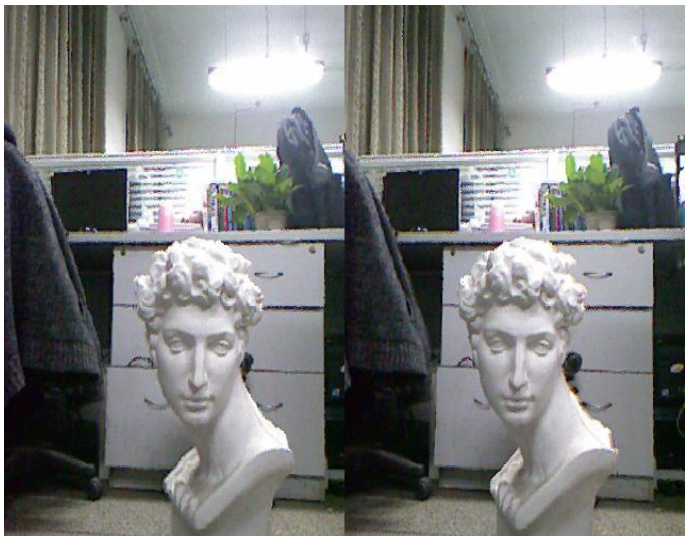


图 4.14 左右视图输出界面

如图 4.14 所示界面，主要是将系统中生成的左右视图进行拼接，方便连接到 3D 电视后进行显示。软件运行时，将这个显示界面全屏，然后投放到 3D 电视上，开启左右分屏模式，使用者带上 3D 眼镜，即可得到具有临场感的立体视觉[32]。

4.6 本章小结

本章从立体视频会议系统的需求分析出发，介绍了开发系统所需要的开发环境以及运行环境、选用原因。并详细介绍了设备获取模块、高速缓冲模块、数据修复模块、立体显示模块的具体实现思路。对于系统中影响实现效果的计算部分专门进行了加速研究，采用了 CPU 多线程和 GPU 中 CUDA 自定义核函数的方法来分别对深度修复和立体显示进行了加速。

第五章 总结与展望

5.1 本文工作总结

本文对立体视频中的深度数据和彩色数据的修复方法进行了研究，首先介绍了使用Kinect数据产生立体视频的原理，然后分析了Kinect数据缺陷的产生原因，继而从对立体视频影响较大的深度数据的完整性出发，在研究主流图像修复算法的基础上，提出了一种融合结构信息进行深度修复的方法。由于立体视频最终的呈现效果还是左右分屏的彩色视频，因此本文又分析了左右视图产生信息缺失原因并采取了快匹配的修复方法进行修复。

融合结构信息进行深度数据修复主要参考了传统的图像修复中非局部算法的思想，以及双边滤波的计算思路。修补的框架中引入了颜色信息、距离信息还有深度信息本身。融合结构信息主要体现在待修补点周围参考点的选择上，不同于传统非局部思想中采取邻域矩形区域内的点，本方法是在 8 个方向上间隔取点。而取点的个数以及最大范围都有明确的限定。在参考点选择完成后，又为每个参考因素设置了不同的参考因子。并从深度信息缺失产生的原因入手，将深度值更大的点赋予更高的权重。在综合了各个参考点不同比例的权重之后得到一个平均加权预测深度。最后对修补过的深度数据进行一次中值滤波使得深度图像更为平滑。

本文另一方面的工作就是设计并实现了一套立体视频会议系统。这是建立在深度数据修复的基础上，首先进行整体的需求分析，随后是对项目中采用的技术、模块设计思路的一个具体介绍，包含 QT 库以及开发、编译环境，以及对系统中关键处的几个设计，比如设备获取模块，缓冲区模块，显示模块。本文同时采用了 CPU 和 GPU 加速方法对系统中可能存在密集计算的地方进行了加速，并详细介绍了多线程加速原理以及视图变换中 GPU 核函数具体编写思路。

5.2 未来工作展望

本文探讨了对 Kinect 的深度数据进行结构导向的深度修复方法，以及对重构出来模型的左右视图进行彩色数据修复，后续在立体视频会议中应用修复算法并进行立体展示，但是在提高计算速度的时候降低了修复的准确性，没有使算法的精确性和速度得到很好的权衡。

下一步考虑用深度学习的方法进行深度数据的修复，这种方法虽然可能在学

习的过程中更加耗费时间，但是一旦学习完成，识别的速度将会很快。对于学习样本的选择，可以采取完整的深度图和彩色图数据，从中人为地擦出部分物体边缘数据，并选取边缘周围点的方法构造大量的数据集进行学习。

在第四章的立体视频会议的实现中，只是考虑到了立体视频数据的传输，没有加入相应的音频传输，而 Kinect 本身也支持音频数据的获取，因此这也是一个后续可以改进的部分。对于场景的立体数据，也可以考虑使用多个摄像头进行采集，然后进行拼接的方法。

参考文献

[1] 鲁业频,李素平. 立体视频技术的发展现状综述[J].电视技术,2012,12:28-31.

[2] Zhang Zhengyou. Microsoft Kinect Sensor and Its Effect[J]. IEEE Multimed, 2012, 19(2):4–10

[3] Gavriel J. Iddan, Giora Yahav. Three-dimensional imaging in the studio and elsewhere[C]. Three-Dimensional Image Capture and Applications IV. San Jose, CA. 2001, vol. 4298:48–55.

[4] Daniel Scharstein, Richard Szeliski. High-accuracy stereo depth maps using structured light.[C] Computer Vision and Pattern Recognition, 2003, vol.1:195.

[5] Andrew K. C. Wong, Peiyi Niu, Xiang He. Fast acquisition of dense depth data by a new structured light scheme[J]. Comput. Vis. Image Underst. Jun 2005, 98(3):398–422.

[6] Wei-Chen Chiu, Ulf Blanke, Mario Fritz. Improving the Kinect by Cross-Modal Stereo[C]. British Machine Vision Association. 2011, 116.1–116.10.

[7] Sergey Matyunin, Dmitriy Vatolin, Yury Berdnikov, Michail Smirnov. Temporal filtering for depth maps generated by kinect depth camera[C]. 3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2011, pp. 1–4.

[8] Sung-Yeol Kim, Ji-Ho Cho, Andreas Koschan, Mongi A. Abidi. Spatial and Temporal Enhancement of Depth Images Captured by a Time-of-Flight Depth Sensor[C]. IEEE, 2010, 2358–2361.

[9] J. Zhu, L. Wang, R. Yang, J. Davis. Fusion of time-of-flight depth and stereo for high accuracy depth maps[C]. Computer Vision and Pattern Recognition, 2008, pp. 1–8.

[10] M. Bertalmio, G. Sapiro, V. Caselles, C. Ballester. Image Inpainting[C]. Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 2000, pp. 417–424.

[11] J. Shen, T. Chan. Mathematical Models for Local Nontexture Inpaintings[J]. SIAM J. Appl. Math, Jan 2002, 62(3):1019–1043.

[12] M. Bertalmio, L. Vese, G. Sapiro, S. Osher. Simultaneous structure and texture image inpainting[J]. IEEE Trans. Image Process., Aug. 2003, 12(8):882–889.

[13] A. Criminisi, P. Perez, K. Toyama. Region Filling and Object Removal by Exemplar-Based Image Inpainting[J]. IEEE Trans. Image Process., Sep. 2004, 13(9):1200–1212.

[14] 葛霁光，靳波，葛耀峥，郭晓晖. 立体视觉深度认知过程研究[N]. 浙江大学学报, 2005, 02:2–7.

[15] 罗桂娥. 双目立体视觉深度感知与三维重建若干问题研究[D]. 中南大学, 博士学位论文，2012.

[16] 王永，孙可，孙士祥. 3D 显示技术的现状及发展[J]. 现代显示, 2012, 12: 26–29.

[17] 范哲. 基于 Kinect 的三维重建[D]. 西安电子科技大学, 硕士学位论文, 2013.

[18] 吴福朝. 计算机视觉中的数学方法[M]. 计算机视觉中的数学方法, 科学出版社, 2008.

[19] 常波. 立体视频舒适度客观评价算法研究[D]. 西安电子科技大学, 硕士学位论文, 2014.

[20] Richard Hartley, Andrew Zisserman. Multiview Geometry in computer vision[J]. Robotica Cambridge UK, Mar. 2005, 23(02):271–271.

[21] Z. Zhang. A flexible new technique for camera calibration[J]. IEEE Trans. Pattern Anal. Mach. Intell., , Nov. 2000, 22(11):1330–1334.

[22] A. Buades, B. Coll, J.-M. Morel. A non-local algorithm for image denoising[J]. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, 2:60–65.

[23] A. Bugeau, M. Bertalmio, V. Caselles, G. Sapiro. A Comprehensive Framework for Image Inpainting[J]. IEEE Trans. Image Process., Oct. 2010, 19(10):2634–2645.

[24] A. Telea. An Image Inpainting Technique Based on the Fast Marching Method[J]. J. Graph. Tools, Jan. 2004, 9(1):23–34.

[25] L. I. Rudin, S. Osher, E. Fatemi. Nonlinear total variation based noise removal algorithms[J]. Phys. Nonlinear Phenom., Nov. 1992, 60:259–268.

[26] A. Wedel, T. Pock, C. Zach, H. Bischof, D. Cremers. An Improved Algorithm for TV-L 1 Optical Flow[C]. Statistical and Geometrical Approaches to Visual Motion Analysis, Eds. Springer Berlin Heidelberg, 2009, pp. 23–45.

[27] K. Khoshelham and S. O. Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications[J]. Sensors, Feb. 2012, 12(2):1437–1454.

[28] 崔大为. 窦文华, 计永兴. 三维视频压缩研究进展[J]. 计算机工程与科学, 2011,5:63–68.

[29] 李艳民. 基于 Qt 跨平台的人机交互界面的研究和应用[D]. 重庆大学,硕士学位论文, 2007.

[30] 眭俊华, 刘慧娜, 王建鑫, 秦庆旺. 多核多线程技术综述[J]. 计算机应用, 2013 S1:239-242.

[31] Wei Wu, Fengbin Qi, Wangquan He, Shanshan Wang. CUDA's Mapped Memory to Support I/O Functions on GPU [J]. Tsinghua Science and Technology,2013, 06:588-598 .

[32] 崔大为. 三维动态显示技术研究[D]. 国防科学技术大学,硕士学位论文, 2010.

致谢

转眼就要离开西安，这七年承载了我的喜怒哀乐、成长蜕变。在西电两年半的研究生生涯中，无论是在学习、生活还是在人生的规划中，我都成熟稳重了许多。毕业临别之际，我要特别感谢所有帮助、关注过我的人们。

特别感谢我的指导老师齐飞副教授两年多来对我的培养与关怀，齐老师治学严谨、诲人不倦，丰富的学识与睿智的思考常常给我带来思维上的启迪，而扎实的编程功底与丰富的 Linux 系统知识更是我孜孜追求的目标。齐老师在深度修复与计算加速等方面不仅给我充满挑战性的课题目标，同时也给予了研究建议与理论支撑，还有良好的计算机硬件与设备支持，在求学态度、人生目标等方面上也使我受益良多。同样感谢的还有我的导师武筱林教授，他在多个领域获得的成就，一直是鞭策我积极进取的不竭动力。

感谢航天电子信息研究所的石光明教授，感谢石老师对我的关注与培养。感谢谢雪梅副教授、张犁副教授、李甫副教授、刘丹华副教授、赵光辉副教授、董伟生副教授、王晓甜老师、林杰老师、牛毅老师、吴金建老师等，感谢各位老师长期以来对我的指导和帮助。

感谢实验室中同级的小伙伴赵亚龙、任京波、吴石重、高山、王艳涛等，与大家共同科研与玩乐的几年毕生难忘，相信将来会是一段美好的回忆。感谢樊春晓师兄的言传身教。尤其感谢夏辰师姐，与师姐合作研究的课题，幸得成果，而师姐的科研精神与专注程度也令我十分钦佩。感谢师弟黄原成，在交接工作过程中，帮我完成了论文中相关资料的整理。感谢何为，与我默契配合，完成了项目中立体视频编解码与网络部分。

感谢师兄杭建、曲丁、刘鑫、陈晓伟等。我们曾一同为了梦想而并肩作战，那段时光将是我一生的财富。

特别感谢我的父母，给予我选择的自由，也教我为人处世。同时也感谢一路陪我走来的女友陈颖，你们将是我继续前行的动力。

作者简介

1.基本情况

男，山东招远人，1990 年 2 月出生，西安电子科技大学电子工程学院电路与系统专业 2012 级硕士研究生。

2.教育背景

2008.08～2012.07	就读于西安电子科技大学电子工程学院信息对抗技术专业，获工学学士学位
2012.08～	西安电子科技大学电子工程学院电路与系统专业硕士研究生

3.攻读硕士学位期间的研究成果

3.1 发表的学术论文

[1] Fei Qi, Junyu Han, Pengjin Wang, Guangming Shi, and Fu Li. Structure guided fusion for depth map inpainting, Pattern Recognition Letters, doi:10.1016/j.patrec.2012.06.003.

[2] Chen Xia , Fei Qi, Guangming Shi, and Pengjin Wang. Nonlocal center–surround reconstruction-based bottom-up saliency estimation, Pattern Recognition. doi:10.1016/j.patcog.2014.10.007

[3] Chen Xia, Pengjin Wang, Fei Qi, and Guangming Shi, Nonlocal center–surround reconstruction-based bottom-up saliency estimation, Image Processing (ICIP), 2013 20th IEEE International Conference , pp. 206–210.

3.2 发明专利和科研情况：

[1] 国家重点自然基金项目，高精度非接触式体感仪的研制的相关项目，立体视频会议系统。

[2] 华为近场定位声波项目，2014 年 04 月至今，负责近场定位中网络模块的开发。