基于分布式渲染的多投影几何校正

摘要

分布式渲染始终是近几年计算机图形学领域和计算机体系结构领域研究的热点，从基于共享存储模式的超级图形工作站到分布式计算机集群，有关这一课题的探索始终是众多研究和开发人员的兴趣所在。目前，在基于计算机集群的分布式渲染这一领域已有了极大地发展，从渲染算法的理论验证，到产品级别的开源渲染框架发布，基于计算机集群的分布式渲染体系已经初具规模，并且依旧在不断地完善。这无疑将为复杂场景渲染，如大规模粒子系统、照片级真实感渲染、计算机动画等领域的实时优化带来新的解决方案，同时也将进一步推动整个计算机图形学领域的进一步发展。本项目中利用到的分布式渲染框架 Chromium 是由 Stanford 大学计算机图形工作小组基于 WireGL 开发的后续版本，它具有着系统稳定、可扩展性强、同 OpenGL 应用程序兼容性强等诸多优点，是研究分布式渲染环境的一款极为优秀的平台。它实现了分布式渲染系统中最为关键的指令派发、网络数据传递管理和渲染同步的功能。本文将对 Chromium 在项目中的具体应用加以介绍，包括：利用 Chromium 框架的分布式渲染功能将几何校正的运算分派到计算机集群中的各个节点中，通过 Chromium 提供的 Tilesort SPU 中实现的同步机制对显示输出进行同步以及利用该分布式渲染框架的动态链接库形式的模块加载开发模式（SPU）对几何校正算法进行集成。目前的实验平台是搭建在一托四的计算机集群平台上，即一台运行应用程序的客户节点和四台进行分布式渲染的服务节点。所幸的是该框架已经充分考虑到了分布式渲染系统的可扩展性，在同一个局域网内只需要通过基于 Python 脚本的 Mothership 进行节点配置，就可以将分布式框架部署在支持统一通信协议的计算机集群平台上；而在跨网段的情况下，可以通过在不同网段内部署 Daughtership 来完成网段间的配置，因而，理论上该分布式渲染框架的渲染节点可以无限扩充，这也为项目的后续扩展提供了极大地便捷。

任意投影表面的投影一致是多投影系统（当然也包括单投影系统）的研究热点。投影表面不会总是如理想的那般平直，原柱、天花板、窗帘等都是可以利用的投影表面。另外，即使投影面是平直的幕布，但非人为的抖动（如船上、火车上等）都会使投影面发生形变，哪怕是很小的。这些都会影响用户的视觉感受，因此需要任意投影表面的技术去解决。为此研究人员结合不同的应用，提出过各种解决方案，如基于结构光的实现、基于单应性矩阵的实现和基于嵌入结构光的实现等等。本项目中采用的方案是软件学院数字实验室之前的研究成果，单投影环境下的基于单应性矩阵几何配准技术。其算法核心原理是对于投影平面是平面的情景，我们可以建立摄像机视角和投影仪视角的一个映射（Homography），在文献[2]和文献[3]中介绍了平面和任意表面情况下的几何矫正方法。通过棋盘格图案进行摄像机定标[4]，其黑白交错的图案性质是其可以被容易独立分辨的重要性质，从而可以用来作为检测图案。OpenCV 函数库中的 cvFindChessboardCorners 是一个广泛运用的搜索棋盘格角点的函数，但在复杂光照和材质条件下，该方法通常失效。我们参考了文献[5]提出的一个基于[6]的算法的方法，运用的是棋盘格的拓扑特性。其主要思想是运用 Delaunay 三角系来连接角点。在本项目中，我们将基于单应性矩阵的几何校正算法集成于 Chromium 分布式渲染框架的 SPU 模块中，通过计算机集群节点对高分辨率的图像进行分布式平台下的几何校正，从而实现复杂

环境下的多投影系统。此外，由于所使用的分布式系统具有很好的可扩展性，因此可以支持较大面积的复杂投影环境。

本文还将介绍在项目中所采用的多种优化手段，这其中包括了基于提高 Cache 命中率思想的局部访问性原理、GPGPU 并行优化、多线程优化和网络传输优化。其中，GPGPU 是一种利用处理图形任务的图形处理器来计算原本由中央处理器处理的通用计算任务。这些通用计算常常与图形处理没有任何关系。由于现代图形处理器强大的并行处理能力和可编程流水线，令流处理器可以处理非图形数据。特别在面对单指令流多数据流（SIMD），且数据处理的运算量远大于数据调度和传输的需要时，通用图形处理器在性能上大大超越了传统的中央处理器应用程序。随着 GPU 统一计算架构和通用计算平台的推出，利用 GPU 进行并行加速的应用案例也越来越多，各种各样的复杂算法都被移植到了 GPU 上并且获得了不错的加速比。相较于以往基于汇编和着色语言的 GPGPU 实现途径，通用计算平台为普通程序员提供了 C 语言风格的通用计算语言，通过其强大的编译平台，向开发者提供了更多有关 GPU 硬件的细节，令程序员可以最大限度的优化自己的程序。在本项目中，由于复杂环境的修正算法十分耗时，包括几何校正、光度补偿、投影拼接、光度一致性等等，如此繁杂的处理流程无疑大大降低了画面的渲染效率，尤其是对于那些实时性要求高的应用程序，如电影播放、远程桌面、游戏应用以及虚拟交互等，较低的帧率将会严重影响用户体验，而这一点也是我们在选定项目课题之初便着重考虑的问题。而在项目的进行过程中，我们也尝试了多种优化方案，包括局部的单机单节点运算处理优化以及系统整体性能的网络优化，并通过比较最终选择了加速比最为理想的方案组合，具体的内容将会在正文第四章做详细的介绍。

关键词 分布式渲染 几何校正 GPGPU

Research of Distributed Cluster Rendering System -Multi Projection of Complex Surface Based on Geometrical Warp

Abstract

Recently, distributed rendering has been the focus of computer graphics and computer architecture. The research interests include graphics supercomputer and clusters, which have appealed many researchers and developers. The problem how to make full use of the multi processor and improve the rendering efficiency for complex scene is not only the research goal of distributed rendering but also the meaning of the revolution in the conventional rendering pipeline. Nowadays, research realm of distributed rendering based on clusters has achieved great development, from the theoretical analysis of the rendering algorithm to the industrial level open source rendering framework. Distributed rendering based on clusters has been developed and keeps on improving itself. It is undoubtedly that the achievement will bring novel solution to the real-time optimization of the complex scene rendering, like large scale particle system, photorealistic rendering, computer animation etc. Meanwhile, it will certainly promote the further enhancement of the computer graphics field. Our project is based on the distributed rendering framework, Chromium, which is the successor version of WireGL developed by the computer graphics research group in Stanford University. It is stable, scalable and compatible with the OpenGL application. It is absolutely one of the most advisable distributed rendering platforms for research. Chromium has implemented three basic key function including rendering command distribution, data transmission management and rendering synchronize. Moreover, due to its special designation it supports multi rendering context, which means it can deal with multi rendering target simultaneously, and requires no hardware change. Furthermore, it also supports various kinds of rendering mode, heterogeneous clusters and various kinds of distributed rendering topology. In other words, it supports both sort-first and sort-last models. Clusters composed by different hardware or operating system are also supported only if all the nodes of the clusters can contact with each other using the same protocol like TCP/IP, which makes the scalability of the distributed rendering easy. In addition, Chromium also supports various distributed rendering topology due to its unique SPU chain rendering model. The platform is also easy to be deployed using the Python script.

Arbitrary projection surface consistence is the research focus of the multi project

system. Projection surface will not always be ideal flat, like cylinder, ceiling, curtain and so on. In addition, even if the projection curtain is flat, it is difficult for it to keep the condition all the time due to various uncertain factors like wind which may distort the projection surface. The distortion will impact the visual experience and needs to be avoided using arbitrary surface projection technique. Researchers have proposed different solution based on specific application, like structural light, homograph matrix and embedded structural light. Our project applied homograph matrix warp based on single project, which is one of the research achievement of our digital lab. The principle of the algorithm is that we can build up a map relationship between the camera viewport and the projector viewport if the projection surface is plane. Some paper introduced the geometrical warp of plane and arbitrary surface. The plane case is based on homograph matrix method while the arbitrary is based on structural light method. Chessboard pattern is always used for camera calibration since the black and white crossing pattern is easy to be recognized, which is suitable for pattern detection. Bouguet introduced a novel interactive method for the purpose of chessboard corner detection in paper. The cvFindChessboardCorners function in OpenCV library is widely used to detect the chessboard corners, but it is malfunctioned if the lighting and material is complex. Paper introduced a method based on the algorithm proposed in paper, making use of the topology features of the chessboard, whose primary idea is using the Delaunay triangle set to connect the corners. In our project, we used the homograph matrix method combining the Chromium distributed rendering framework. We warp the high resolution image with the help of the clusters' nodes to implement the multi project system under complex projection surface condition. Moreover, thanks to the scalability of the distributed rendering system, this project works even if the complex projection surface is quite large.

GPGPU is to handle the general computing task which used to be solved by CPU using GPU which is responsible for image rendering. All of these general computing stuffs are not related to image rendering. Thanks to modern GPU powerful parallel processing ability and programmable pipeline, the stream processor can handle non-graphical data. Especially under the SIMD condition and when computing consummation is much larger than data transmission consummation, GPU is superior to CPU in dealing with the application. Since the upcoming of GPU unified computing architecture and general computing platform, parallel optimization using GPU has been popular. And various complicated algorithm has been transplanted to GPU and achieved promising speedup. Compared with former GPGPU implementation using assembly language and shading language, general computing platform provides programmers with C language style general computing language. The powerful compiling platform provides programmers with more specific GPU hardware details, which helps us optimize our programs. In our project, because of the time consuming due to computing for complex environment including geometrical warp, color compensation, projection combination and hue consistence. The complicated processing step will certainly lower the rendering speed, which will have side effect on the user experience especially for those real-time required application like playing movies, remote desktop, games and virtual interaction. We have taken

this point into account as we select this project. We have tried various optimization methods including single node computing optimization and the network optimization. By comparing the speedup of each method we applied the most efficient combination to our project, which will be discussed in detail later in chapter four.

Keywords Cluster Parallel Rendering, Warp, GPGPU

目录

第一章　绪论

增强现实是一项旨在创造真实物理世界同计算机虚拟的数字世界相融合的技术，同时它也是目前计算机图形学和虚拟现实研究领域的一个重要分支。早在 1997 年，来自北卡罗来纳大学的 Ronald Azuma 就曾在他的一篇 Survey 中[1]提出了关于增强现实的通用定义。在这篇文章中，他认为增强现实可以包括三个方面：即将虚拟物于现实结合、实时交互和三维。而增强现实技术按照显示设备可以主要分为三个方向：头戴式、手持式和空间显示技术。随着科学技术的不断进步，目前投影设备的性能较以往已经有了很大的提高，而且同时一些中低端产品的价格也已能为普通民众消费者所接受，这使得基于投影设备的增强现实空间显示技术快速兴起。

相比其他的增强现实实现技术，基于投影的实现方式有着其独有的优势。首先，它提供了一种更为自然地人机工程方法，即通过调节可变的显示分辨率，使得人眼感受更为自然。其次，通过增加投影仪的数量，理论上可以近乎无限地扩展视野范围，从而获得沉浸式的用户体验。这类应用在各种展示博览会中已经得到广泛的应用，目的在于通过这种新型的视觉体验加深参观者对于展品的印象。日常生活中，投影技术对于我们每个人而言也并不陌生：数字课堂上教师所使用的投影仪、环幕影院中的 360 度投影拼接等等，无不彰显着投影技术给人类生活带来的全新变革。而伴随着各种反馈式交互技术的出现，他也被应用于医疗成像，工业制造等领域。因此该技术有着广泛的应用前景，同时也是国内外增强现实技术研究的一个热点，许多研究人员尝试着通过各种手段不断地改进投影技术以及扩展其应用领域，并已取得了许多成效。

尽管投影技术是增强现实的一种有效地实现途径，然而投影设备自身的限制无疑成为了摆在研究和开发人员面前的一道难题，比如投影屏幕要平整，否则发射屏幕的形变会致使投射的图像发生扭曲，从而带来视觉上不良的观感。随着现实增强领域相关研究的发展，投影环境已逐渐摆脱平面的限制，各种基于复杂平面或曲面的投影校正算法相继提出，为该领域的应用转化提供了基础。同时，如何高效实时的完成校正并投射结果，也是在实际应用转化中需要面临的问题。本项目主要研究基于投影设备的空间增强现实技术，主要针对不规则几何形状物体表面，解决投影式空间增强现实中的虚实合成等关键技术问题，并结合分布式集群渲染系统，对提供高效的几何校正投影方式进行了研究。本项目还尝试采用多种优化手段与措施，包括基于 CPU 和 GPU 的并行计算、网络异步传输等，力图实现在实验室硬件条件下的实时分布式渲染。在实际研究的过程中，充分的考虑了对于渲染效率和网络带宽利用方面的优化，并在比较各种优化手段效果后，选择了加速比最高的一种实现方式。此外，由于是基于 Stanford 的开源项目 Chromium 框架，本项目的所有算法实现最终都是以动态链接库的方式集成于框架之中，便于同后续基于该框架研究的集成。最后通过展览展示应用体现其相对传统投影方式的改进。因此项目的主要研究内容可以分为三块：1、基于单应性矩阵的几何校正算法移植实现：主要是针对实验室原有的单投影设备下几何校正算法进行从 Windows 到 Linux 平台的移植，并将其集成到 Chromium 分布式渲染框架内。使得在多投影的应用中，单台 PC 机只需处理其所负责投影部分的几何校正。涉及到了图像分割和偏移参数分配。2、基于 Stanford Chromium 框架的分布式渲染框架搭建：由于项目中的演示是使用四台投影仪，因此通过 Chromium 的配置文件实现一托四（一台 Appfaker 和四台 Render

Server）的网络拓扑，并分别在每台机器上配置了相关的编译和运行环境。3、针对几何校正算法在 Chromium 框架内实现的优化研究：这其中包括了通过访问局部性原理对单线程的几何校正算法进行优化；通过静态线程池对算法进行 CPU 多线程优化；GPU 加速优化（分别尝试了通过 Nvidia Cg 着色器和 Nvidia CUDA 通用计算平台对算法进行 GPU 提速）；以及通过网络数据缓存机制，异步每一帧的图像渲染计算和网络图像传输，同时保留 Chromium 所提供的同步显示机制，使得网络带宽的利用率有所提升。

本项目预期达到的目标有以下几点：1、移植实验室单投影环境下的几何校正算法；2、完成 Chromium 分布式渲染平台的搭建；3、实现 Chromium 框架下的几何校正 SPU；4、对系统渲染性能进行整体优化。以下二至四章将会进一步详细地介绍本项目的三个主要技术内容，包括研究背景、技术实现细节、遗留问题和未来的工作等。而第五章将会介绍项目完成后的实际测试结果以及项目进展过程中遗留的一些问题和设想的解决方案。

第二章　几何校正

2.1 几何校正的研究现状

在针对不规则的几何表面的虚实合成技术研究中，我们采用了基于单应性矩阵的几何校正方法，各个部分国内外相关工作如下。

2.1.1 基于单应性矩阵方面

对于投影平面是平面的情景，可以建立关于摄像机视角和投影设备视角的一个映射（Homography）。在 Michael Brown 的文章[2]和 Oliver Bimber 的文章[3]分别介绍了平面和任意表面情况下的几何校正方法。其中 Michael 的方法是基于单应性矩阵的方法，而 Oliver 采用的则是基于结构光的方法。Microsoft Research 的 Zhengyou Zhang 提供了通过棋盘格图案进行摄像机标定的方案[4]，其黑白交错的图案性质是其可以被容易独立分辨的重要性质，从而可以用来作为检测图案。Bouguet 在他的文章中提出了一种新的交互式的方法来检测棋盘格的角点。在计算机视觉领域的著名开源软件包 OpenCV 的函数库中，cvFindChessboardCorners 是一个被广泛使用的搜索棋盘格角点的函数，但在复杂光照和材质条件下，该方法通常失效。Chang Shu 在他的文章中[5]提出了一种基于文献[6]的算法的方法，运用的是棋盘格的拓扑特性。其主要的思想是运用 Delaunay 三角系来连接角点。

2.1.2 基于结构光方面

结构光测距是一种既利用图像又利用可控光源的测距技术，其基本思想是利用照明光源中的几何信息帮助提取景物中的几何信息。根据光学投射源所投射的光束模式的不同，结构光模式可以分为点结构光模式、线结构光模式、多线结构光模式以及网格结构光模式。线结构光模式复杂度低、信息量大，应用最为广泛。70 年代初，Will 和 Pennington 首先提出利用结构光单点法进行三维测量， 国内基于结构光三维重构技术研究工作都是在大学和研究所内进行的。天津大学首先研制了线结构光轮廓传感器。北京航空航天大学研究了基于 RBF（Radial Basis Function，径向基函数）神经网络的结构光三维视觉检测方法，该方法不需要考虑视觉模型误差、光学调整误差等因素对视觉检测系统测量精度的影响，因而能够有效地克服常规建模方法的不足，保证了检测系统具有较高的精度。清华大学研制一种基于线结构光的多用途传感器，特别适合于对移动物体和腐蚀性表面的快速、在线、非接触的重构与测量。在国外，结构光三维测量系统的研究更为成熟，已经制造出许多基于结构光的三维测量设备应用于各个领域。20 世纪 90 年代后，研究重点主要集中在光栅结构光方面。如 Richard.Peter 和 Gunther 等人提出的一种用光栅结构光进行投影的三维测量系统。目前，越来越多的研究人员开始采用编码结构光的方法和彩色结构光的方法实现三维重构。国外研究人员在这方面不但做了大量的理论研究，而且，许多研究成果已应用于生产实际之中，法国的 J.M.Lequellec 和 F.Lerasle 研制的一种结构光传感器，已用于汽车舱的三维重建。美国公司研制生产的一种彩色结构光传感器，简称 CSL，已用于表面凹痕检测，高度测量，焊料粘贴，厚度测量等等。

2.1.3 基于嵌入结构光方面

嵌入结构光技术是一种通过编码技术将将结构光图案嵌入原始投影图像中的合成技

术。通过这种技术，合成图像中的结构光图案信息是难以被人眼所发觉的，但是却可以被计算机单独解析提取出来。这一特点凸显了结构光“隐藏性”的优势。嵌入结构光系的发展历程是从 90 年代开始的。1998 年 Raskar 就提出一种思想[7]，就是通过对 DLP 投影仪的编程控制,将结构光图像嵌入原始图像中，从而让人眼无法发现结构光图像。Raskar 还为此设计了一个特殊的数字光控投影仪，这个投影仪可以达到 1000Hz 的刷新频率。该方法用于获取物体或者场景的深度信息，不过由于当时硬件上的限制，Raskar 只是推导证明了这个想法的理论可行性。Bimber 在 2007 提出了一种新的时间编码技术[8]。这种方法对设备配置要求有所降低。它可以通过事先选好的一组不可见的结构光图案嵌入在投影画面中，借此来获得进行几何矫正和光度补偿所必需的信息。目前的许多应用在交互性上对结构光测量方法有着更高的要求。用户一方面希望可以更加快速地获得重建好的三维信息，另一方面还要求能够去除结构光打在场景表面所带来的颜色干扰。如果能够解决这两个问题，那么这种新的结构光技术在游戏、制造工业、医疗等领域实用价值会更大。

2.2 单应性矩阵

2.2.1 单应性矩阵介绍

首先需要解释一下什么是单应性矩阵。

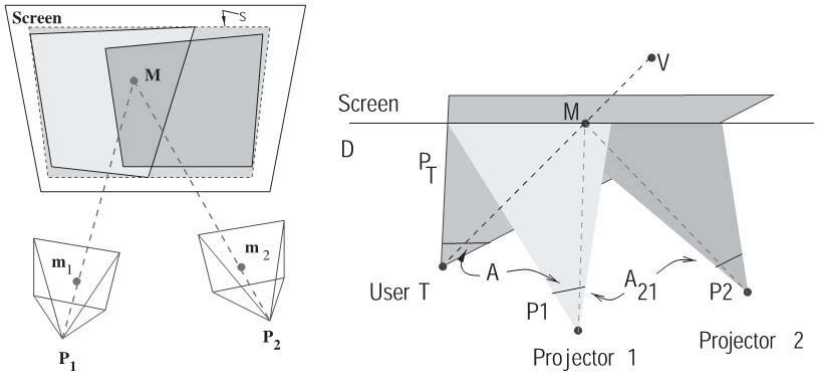


图 2-1 单应性矩阵几何原理

单应性矩阵（homography matrix）表示了不同坐标系下点在同一平面上的投影之间的映射关系，可以将一个透视空间像素坐标映射到另一个透视空间中。以图 4-1 为例解释单应性矩阵的作用。定义 3×3 单应矩阵 H

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2-1}$$

可以建立 M 点在两个投影空间 P_1 和 P_2 间的坐标转换，满足

$$m_2 \cong H_{3\times3} m_1$$

$$\begin{bmatrix} x_{m_2} \\ y_{m_2} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{m_1} \\ y_{m_1} \\ 1 \end{bmatrix} \tag{2-2}$$

其中 m_1 是空间 P_1 坐标系下的点，齐次坐标 $\begin{bmatrix} x_{m_1} & y_{m_1} & 1 \end{bmatrix}^T$，$m_2$ 是空间 P_2 坐标系下的

点，齐次坐标 $\begin{bmatrix} x_{m_2} & y_{m_2} & 1 \end{bmatrix}^T$；$m_1, m_2$ 在平面 S 上的投影都是点 M。

矩阵归一化使得 $h_{33}=1$，这样我们有 8 个参数需要计算，这只要通过大于等于四组点的对应关系，就可解方程组得到矩阵 H。一般使用多于四个点的对应关系，用最小二乘法计算以减少误差和非线性关系。

2.2.2 单应性矩阵算法思想

在计算机视觉中，空间中某一平面上的点在两个不同视平面下得到的图像可以通过 Homography 建立联系，并且这两个视角的位置是基于原实物平面的一个旋转。首先通过投影仪将棋盘格图片投到投影平面上，其次在一个将投影仪基于原平面做一个小旋转的位置用摄像机拍摄图片，下面就是将获得图片进行角点检测，并与原棋盘格的焦点建立匹配关系，随后基于这些匹配实现几何矫正。

上述思想中需要解释的部分如下，首先为什么我们建立了两幅图像中的角点之间的匹配关系后就能够实现几何矫正？原因在于，我们将棋盘格投影到一个非平面的背景上，但是事实上这个背景可以看作是多个小的平面近似构成的，而这些所谓的小的平面，就可以看做是由棋盘格中某四个相邻的顶点形成的小正方形来确定的。又因为，在投影仪中投出的棋盘格图片，在投影仪的角度看来永远是如同投影到平面上一样的，故在投影仪角度的棋盘格角点的位置我们相当于是已知的。因此，当我们检测出了从摄像机图像中获得的棋盘格图像中的角点，并将它们与原棋盘格建立好了匹配关系后，这一个个小平面上的点在这两个视角下的位置之间就建立了一个关系，即 Homography。由此，利用这一系列 Homography，如果我们想在摄像机视角下获得一个正确的如同投影到平面上的图案，我们就知道如何矫正投影仪中的源图案。

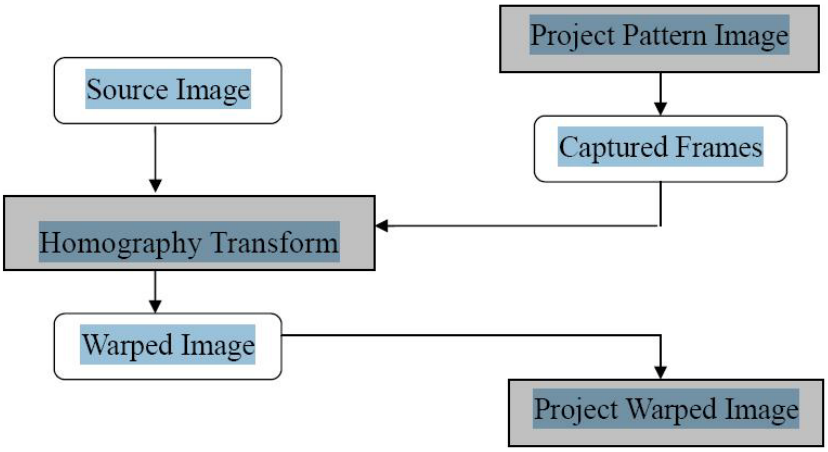


图 2-2 单应性矩阵算法流程

上述算法的基本思想中涉及到了两项基本技术，即棋盘格角点检测与棋盘格角点匹配。

2.2.2.1 角点检测

首先通过计算机读入获得的棋盘格图片，并将其转化为灰度模式，这样在下面的工作中就只需通过对灰度的判断来从图片中提取角点的坐标。考虑棋盘格角点投影的一些不变量，我们选取某点一定邻域内的黑白信息加以判断，其思想如下，在标准棋盘格中，选取以某一个角点为中心的正方形邻域，可以发现，领域正方形可由外及内分为若干层，每一层展开成一维的线性结构时都是黑色和白色区域间隔出现的，即使是投到扭曲平面上我们所获得的棋盘格图片依然如此，因此通过这样一个不变的性质，我们可以将扭曲图像中的

角点检测出来。又由于实际操作过程中受光线等其他因素影响，因此我们的算法还引入了一些参数，如黑白像素差、误差像素个数上限等等，来加以对不同实验环境的控制。

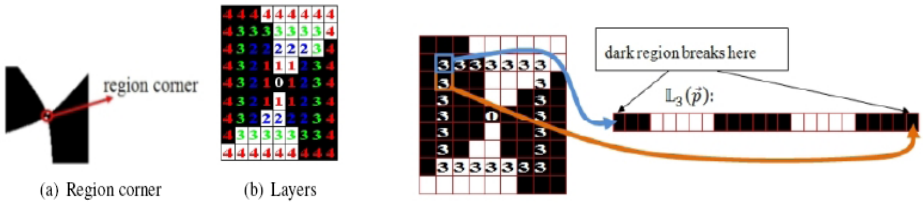


图 2-3 角点检测算法原理

图 2-4 角点检测实验效果图

2.2.2.2 角点匹配

校准采用统一坐标系的图像变换算法来自然的完成拼接。每个投影仪依次投影单张棋盘格图像，并由相机依次捕捉。

在相机视角 V_c 中，每个投影仪覆盖一个扭曲了的投影表面 S_{c_j}，投影的棋盘格内角点构成四边形网格，每个网格区域认为是一个平面，模拟表面形变。通过投影棋盘格 pattern 图 $S_{p_j}=\{Q^i_{p_j} \mid Q^i_{p_j}$ 是棋盘格每一小格$\}$到投影表面 S_{c_j}，S_{c_j} 被划分为四边形面片网格$\{Q^i_{c_j}\}$，对应棋盘格 pattern 图四边形网格$\{Q^i_{p_j}\}$。我们提出一种棋盘格查找算法查找所有棋盘格内角点，并建立对应关系。两平面之间的几何对应关系可由 homography 矩阵表示。对于每一对$<Q^i_{c_j}, Q^i_{p_j}>$，我们应用上述性质。为每一对$<Q^i_{c_j}, Q^i_{p_j}>$，通过四个角点的位置对应建立方程，计算出两个矩阵 $H^i_{c2p_j}$ 和 $H^i_{p2c_j}$，如图 4-2 所示，用于 $Q^i_{c_j}$ 和 $Q^i_{p_j}$ 内部对应点之间的位置换算。图 4-3 显示了单个投影仪棋盘格的四边形网格示例。设从用户观察坐标系 S_v 到相机坐标系中第 j 个投影仪覆盖区域 S_{c_j} 的 homography 矩阵为 H_{v2c_j}，在 S_v 中，用户所看到的一个像素点 p_{v_j}，通过公式 4-3 转换到投影仪坐标系 S_{p_j} 中的一点 $p^i_{p_j}$

$$p^i_{p_j} = H^i_{c2p_j} \cdot H_{v2c_j} \cdot p_{v_j}, p^i_{p_j} \in Q^i_{p_j} \quad (2\text{-}3)$$

反之，从投影仪坐标系 S_{p_j} 到用户观察坐标系 S_v 的变换使用公式 4-4

$$p_{v_j} = H_{c2v_j} \cdot H^i_{p2c_j} \cdot p^i_{p_j}, p^i_{p_j} \in Q^i_{p_j} \quad (2\text{-}4)$$

在多投影系统中，相邻的 S_{c_j}，例如 S_{c_j} 和 $S_{c_{j+1}}$ 会重叠来避免几何上的缝隙。所以 p_{v_j} 可能会映射到多个具有相同像素值的 $p^i_{p_j}$ 上，这就会给重叠区域带来高亮的效果。所以需要一个光度校正过程来达到光度一致性。这里 H_{v2c_j} 可以通过跟踪设备或手动标定。简单起见，我们假设用户和相机在同一个位置，这样 H_{v2c_j} 成为单位阵而可以被忽略，利用公式变换源图像，我们就可以得到每个投影仪需要投影的图像。由于上述 homography 矩阵需由四个点的对应关系才可建立方程计算，所以只有棋盘格内角点所构成的四边形面片网格内的像素点才能通过乘以 homography 矩阵建立准确的几何映射。所以每个投影仪的显示区域限制在内角点所构成的四边形面片网格内部。这样虽然缩小了一定的显示范围，但换来了准确的几何映射，多投影拼接也可以很自然的解决显示画面小的问题。而且，投影仪边界区域的亮度要明显低于内部，舍弃边界部分有利于之后的光度校正。

上述方法对于平面及复杂几何面同样适用。通过改变棋盘格的方块数量就改变了投影表面网格的分割。预先生成好不同方块数量的棋盘格，校准的时候直接采用。这个过程也可以在校准前软件生成，这样可以根据校准结果产生更多的棋盘格图案，增加灵活性。但这样做校准前准备时间会过长。而且由于棋盘格连续校准的过程中会一直用到，所以我们还是采用预先准备好的图案，尽可能的提供多种数量级的图案。

初始使用较少方格数量的棋盘格图案（我们的试验中，对柱面等二次曲面， 5×7 方格是适用的）。校准过程中，平面与复杂几何面有以下几点不同：

- 计算相机与投影仪间的映射关系。复杂几何面的情况，不再是一个单应矩阵能表示的。我们还是投影棋盘格图案，用相机捕捉投影表面，复杂投影表面给投影棋盘格造成的形变给出了投影表面的几何信息。以每一格的四个定点建立投影仪与相机间的单应矩阵，每格投影仪得到了一个单应性矩阵组。

- 计算投影坐标系到全局参考坐标系的映射关系。复杂几何面的情况下需要建立像素的点点映射。首先用上一步得到的一系列矩阵，计算出每个方块投影仪坐标系到全局参考坐标系的变换矩阵。然后对投影图像上的每个像素，根据其所属的方格，应用该格的投影仪与参考坐标系的单应性矩阵，最终得到投影坐标系到全局参考坐标系的点点映射关系。

当校准结束后，如果配准误差较大，采用更细分的棋盘格继续校准。这个过程可以通过计算相机捕捉到校准后的投影效果与理想位置的投影效果计算误差来判断，可以自动实现。

2.2.2.3 几何校正

经过上述匹配过程后，我们就可以通过两个视角下的棋盘格小平面建立相应的Homography，此处的映射矩阵是采用 OpenCv 自带的函数 cvFindHomography 来实现的。这样，投影出的棋盘格区域的所有点都完成了和原棋盘格图像的对应，对于我们需要显示的图片，只要设定好其在输出平面上的位置，就可以找到每一像素在原投影棋盘格中的位置，我们就可以将图片的像素信息写入到投影仪投影的图片当中,实现最后的几何校正。通过前面的算法介绍我们可以知道，我们建立的映射是在投影仪平面和摄像机平面之间的，因此最后实现的结果也只有在摄像机所在位置观察是正确的，而在其他位置观察，则不能看到几何矫正的结果。

（a） （b）

图 2-5 几何校正效果图（图 a 是未经校准的投射结果图像；图 b 是经过几何校准的投射结果图像）

以下是项目中校正前与校正后实验结果的对比图

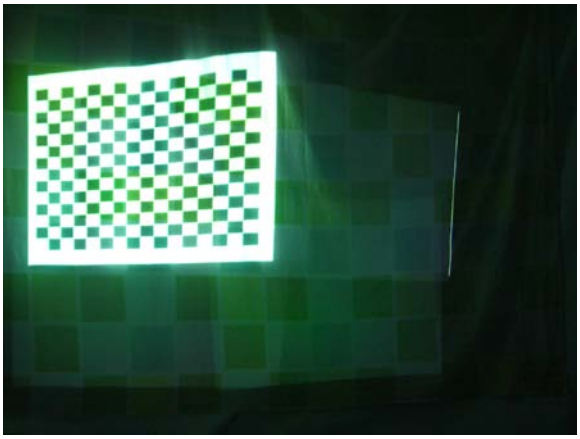
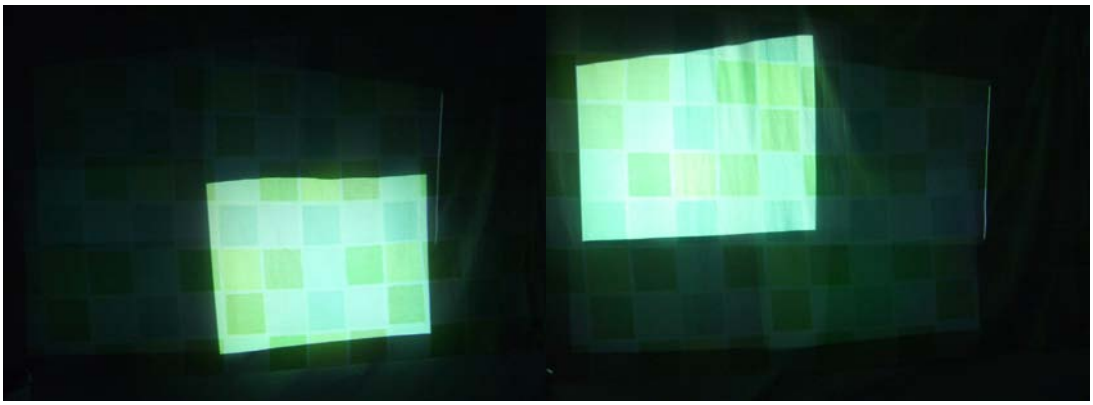
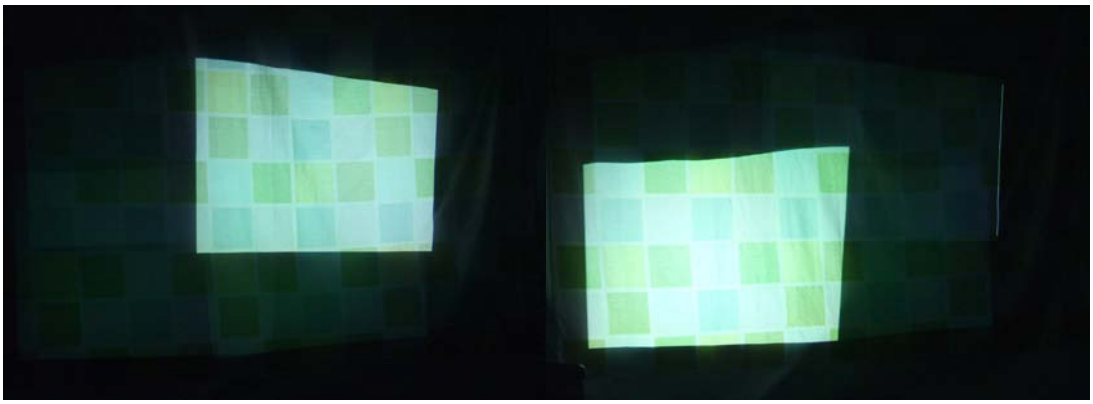


图 2-6 用于几何校正的棋盘格投影图



（a）　　（b）



（c）　　（d）

图 2-7 未经几何校正的四张投影结果（图 a、b、c、d 分别是多投影系统中四台投影仪未经几何校准的投射结果图像）

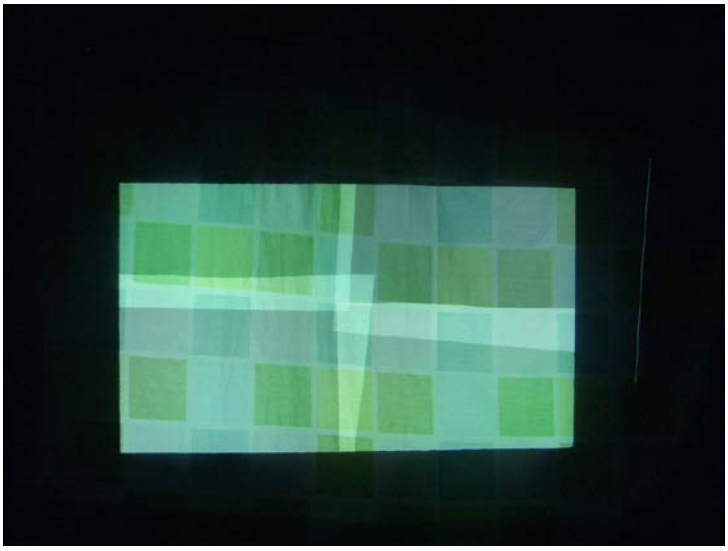


图 2-8 经过几何校正的投影结果

2.3 本章小结

本章主要介绍了几何校准技术的研究现状和方法，并且重点介绍了基于单应性矩阵方法的几何校准技术，并给出了基本的算法流程。

第三章　分布式渲染框架

3.1 分布式渲染的研究现状

3.1.1 分布式渲染的发展

传统意义上的多屏分布式渲染系统环境主要是通过高性能的超级图形计算工作站驱动的，著名的有 SGI（Silicon Graphics, Inc.）公司的 Onyx 系统。该系统具有以下突出的性能特点：共享内存式多处理机制、多重同步图形渲染管线，同时该系统还提供了一个稳定并且可扩展性强的开发平台，主要针对高性能的虚拟现实和可视化模拟等应用的开发。然而不幸的是，所有这些突出的性能却都是以高昂的开销为代价换取的。因此，类似于这种多屏分布式渲染投影环境系统的客户对象只局限于少部分的财团或公司。

近几年，随着高性能、处理功能日趋完善的个人电脑显示卡的迅猛发展，其价格也在不断下降，并且已经达到了普通用户所能够接受的程度，尤其是一些中低端的产品型号如今已经可以用极为低廉的开销得到。同时，由于个人电脑显示卡的不断发展和进步，通过个人电脑组成的计算机集群来替代以往的超级图形计算工作站也成为了可能。如此一来，可以将原本高昂的成本大幅度地削减。然而，随之而来的问题却是如何将原先的分布式渲染体系框架移植到以普通个人电脑组成的集群上，因为共享内存式系统与计算机集群在程序模型上有着很大的不同。在共享内存式的图形系统中，程序员不必担心不同处理器间的数据共享以及不同图形引擎的渲染信息派发等问题；而在计算机集群系统中，程序员却不得不考虑并花费大量的精力去解决上述问题。然而幸运的是，随着 MPI、OpenPBS 的出现，程序员可以逐渐的从这些琐碎的事物中摆脱出来，而专心于在计算机集群平台上的应用开发，而将计算一致性等问题交给这些分布式消息接口来负责处理。

有关基于计算机集群方式的多屏分布式渲染的架构设计与开发近几年已逐渐成为研究的热门与焦点，但该领域尚处于探索研究阶段，因此大家在构建分布式渲染系统的标准体系上并未达成共识。然而，依旧有许多的开发者愿意将他们的现有应用移植到计算机集群平台上或直接在平台上开发新的应用。另一方面，尽管有关平台构建的研究才兴起没多久，但相继已有各种不同的架构体系被提出，有些还是开源软件。

3.1.2 基于计算机集群的分布式渲染介绍

通俗地讲，基于计算机集群的分布式渲染是指通过网络将多台计算机相连接并进行渲染工作，例如基于光线追踪或者辐射度模型的分布式非真实感体渲染、基于图形应用程序编程接口（OpenGL 或者 DirectX）的交互式渲染。在此，我们有必要对相关的术语做一下说明，以免产生歧义。如同在 X 窗口系统中的定义，分布式渲染系统习惯上将运行着应用程序的计算机称为客户端，而负责在本地进行渲染的计算机称为服务端。

大多数近期有关于基于计算机集群分布式渲染的研究工作都将研究重点聚焦在对计算机间分布式多边形渲染不同算法的研究。Molnar 认为[9]，根据图元（primitive）在从模型空间变换到屏幕空间的过程中出现的位置，可以将这些算法分为三类，包括有：1、前序（sort-first）2、中序（sort-middle）3、后序（sort-last）

3.1.2.1 前序算法

在前序算法中，显示的内容被分割为离散不相交的分块（tile），然后将这些分块分配给计算机集群中的每一个渲染节点，而每一个渲染节点都会接收到一个或多个这样的分块，并负责对位于分块内的图元进行渲染。为了实现这一点，图元通常会经过一个预转换（pretransform）的过程以确定图元在屏幕空间内覆盖的区域，然后根据包含该图元的分块将图元发送给接受这些分块的渲染节点。当发送图元到指定的渲染服务器时，网络的带宽消耗可能会很高，而通过利用帧与帧之间的连续性，可以有效地减少网络传输量。由于图元分派的影响，前序算法需要面临负载均衡的问题。Rudrajit Samanta 提出了通过动态调整分块的方法来提高负载均衡的效果[10]。当计算机集群中的渲染节点不断增加时，前序算法也显现出了可扩展性不强的劣势。此外，每个在分块边界上的图元都至少要被分配到两台渲染服务器上并被分别渲染，当图元的数量不断上升时，这种情况发生的可能性也就越大，无疑会对系统的整体性能有所影响。Samanta 通过综合前序和后序算法解决了这个问题[11]。

3.1.2.2 中序算法

在中序算法中，几何图元会先被发送到各个处理器（每个图元只会被发送到一个处理器）进行几何变换阶段的处理，当几何图元被变换到屏幕空间后，图元又会被发送到另外一个处理器进行渲染。类似于前序算法，屏幕空间会被分割成不同的分块，但每个处理器只需要负责分块内图元的光栅化操作。这种算法需要将光栅化操作和渲染管线分离，以便于图元的重新分发。目前，这种算法的实现还依赖于特定的硬件，例如 SGI 公司的 InfiniteReality 引擎。

3.1.2.3 后序算法

后序算法中，每个图元都会被各自发送到计算机集群中的一个渲染节点进行渲染，当所有的图元都被渲染后，这些节点将会将各个渲染的结构进行组合并得到最终的图像。这通常都会需要很大的带宽，因为每个节点都需要将渲染后的整个输出图像发送到一个组装节点进行图像拼装组合。

3.2 分布式渲染系统实现模型

本项目的目的是设计并实现分布式多投影系统，而对于这类分块显示系统而言，前序算法方案无疑是最为有效、便捷、直观的方案。显示内容被分割成不同的分块，而每个分块都将会被一个计算机集群中的渲染节点负责处理。相对而言其他的方案则在处理步骤和带宽消耗上显得低效许多。或许是由于这个缘故，主流的分块分布式渲染软件系统的设计都是采用前序算法的方案，而他们的主要区别在于计算机集群节点间的数据分发方式。Yuqun Chen 最早着手对这一问题的研究[12][13]，他提出了两种通用的架构模型：1、客户--服务模型（client-server）2、主人--佣人模型（master-slave）

3.2.1 客户—服务模型

在客户服务模型中，应用程序运行在客户节点上，客户节点负责生成几何图元信息并派发到各个渲染服务器上。如图 1a 所示。我们可以将该模型分为两种渲染模式：1、立即模式（immediate mode）2、保留模式（retained mode）。其中，在立即模式下，客户节点将会在每一帧时都通过网络发送几何图元；而在保留模式下，每个渲染服务器都会在本地储存已经接收到的几何图元以便于复用，而客户节点只需要将图元几何上的变化发送给服务器，这种方法通常是通过一种场景图（scene graph）来实现的。

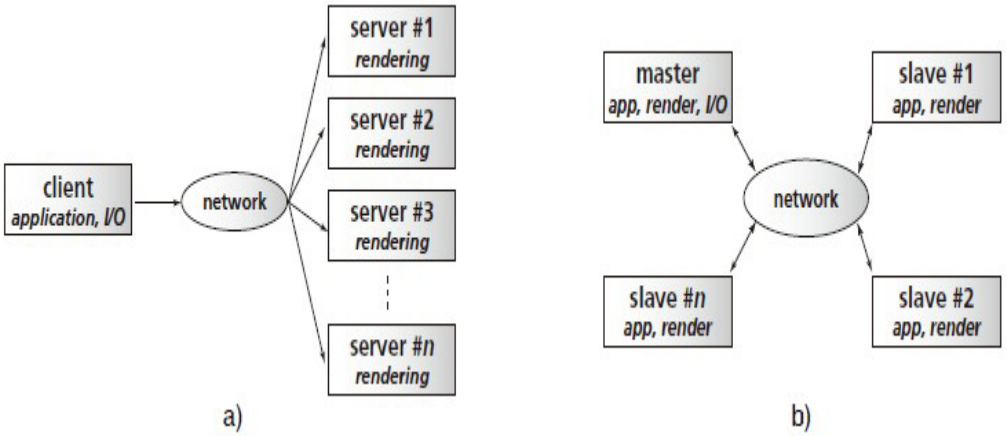


图 3-1 两种架构模型（图 a 是客户-服务模型示意图；图 b 是主人-佣人模型示意图）

3.2.2 主人—佣人模型

在主人佣人模型中，应用程序运行在每个计算机集群节点上，并且各个节点上运行的应用程序必须保持同步，以确保各个应用实例间的一致性。而主人节点此时便负责处理协调所有的用户交互和所有其他节点间的状态变化的同步，如图 1b 所示。

3.2.3 两种模型的比较

主人佣人模型所需要的带宽很少，用户交互和状态改变的信息是相对比较零散并且易于通过网络传输的。但这种方法却未必透明，因为任何影响程序运行的因素都必须作为系统的输入。计时器、随机数、系统调用或者任何可能影响程序运行的变量都必须在节点间进行派发和同步。而客户服务模型对于程序员而言则相对透明得多，在程序员看来，应用程序就如同是运行在一台个人电脑上而不必去关心底层的协调实现，系统框架会负责处理其余的部分。对于这两种模型，目前都有了各自相对成熟的实现版本。基于客户服务模型的有 Aura（Broadcast）、Syzygy（Scene Graph）、Parallel iWalk 等，基于主人佣人模型的则有 Aura（Multiple Copies）、Syzygy（Master-Slave）。

3.3 分布式渲染系统框架

目前图形学社区里相继已有为数不少的分布式渲染系统框架发布，在这之中不乏性能出色的版本，如 OpenSG、VRJuggler、Chromium[14]等，他们都有着一些共同的特点，包括支持多个操作系统平台和异构计算机集群、部署方便、在图形学领域中被广泛地使用、开源等等。本项目中，我们采用的是 Stanford 大学开发的 Chromium 框架，下面将会对该框架做一简要的介绍。

3.3.1 Chromium介绍

Chromium 是 WireGL[15][16]的后继版本，他提供了在计算机集群平台上对于 OpenGL 应用程序的支持。Chromium 用自己的库替换系统的 OpenGL 库，并直接对 OpenGL 的图形指令流进行操作。Chromium 还提供了流处理单元（Stream Processing Unit）机制，以下简称 SPU。每个 SPU 都将一系列的图形指令流作为自己的输入，并对这些输入的指令进行一些操作，然后再将他们往后传递。多个 SPU 还可以像链条一般串联在一起对图形指令流进行复合式的操作。一些框架内置且最为基本的 SPU 包括渲染 SPU（render SPU）、包装 SPU（pack SPU）以及打印 SPU（print SPU）等等，其中渲染 SPU 负责将指令流传递给本地系统实现，包装 SPU 则是通过将指令流打包并发送给计算机集群中的服务器，而打印 SPU 顾名思义则是将指令流以指定的格式输出（终端或文件）。通过替换系统的 OpenGL 库,Chromium 理论

上可以运行任何 OpenGL 的应用程序。

Chromium 的 SPU 是通过模块化的方式实现的，这就意味着 SPU 可以通过任意的方式进行组合。通过修改配置文件可以指定他们之间的组合方式而不需要再进行重新编译。由于该框架着眼于基于普通家用电脑组成的计算机集群，因此该框架并没有考虑图形流水线中不同阶段的通信。也正因为如此，基于中序算法架构的应用将不被兼容。

Chromium 通过分块排序 SPU（tilesort SPU）支持分屏显示分布式渲染。用户可以自定义矩形分块大小，也可以让分块间有所重叠，甚至矩形分块可以不平行于屏幕空间的坐标轴。这意味着用户可以通过传统的几何变换将矩形分块切边或者指定梯形的分块。

在渲染几何图元的应用中，Chromium 会对每个顶点进行预转换（pretransform）的步骤并替每个几何图元维护一个屏幕空间内的包围盒。当指令发送缓冲区被填满后，Chromium 会根据各个包围和所覆盖的分块决定该包围盒所包围的几何图元应该被发送到哪些分块所对应的渲染服务器。最坏的情况下，几何图元的包围盒占据了每个分块，即该几何图元将会被发送给每一台渲染服务器。由于 Chromium 每一帧都会发送几何图元给渲染服务器，即它采用的是立即模式，这无疑将会带了极大地网络传输的开销。另一方面，为了对传输数据量进行优化，Chromium 的指令发送机制基于一个假设：几何图元如果在时间上相邻那么在空间上也相邻，这一假设将会在渲染等值面、三角网格时带来一定的优化。

3.3.2 Chromium项目应用

Chromium 分布式渲染框架支持多种计算机集群拓扑结构，同时也对前序（sort first）和后序（sort last）提供了支持，由于本项目的实际应用情况，我们采用的是前序算法结构，计算机集群由一台运行应用程序的客户节点同四台渲染服务器节点组成。分布式渲染框架部署情况如图所示：

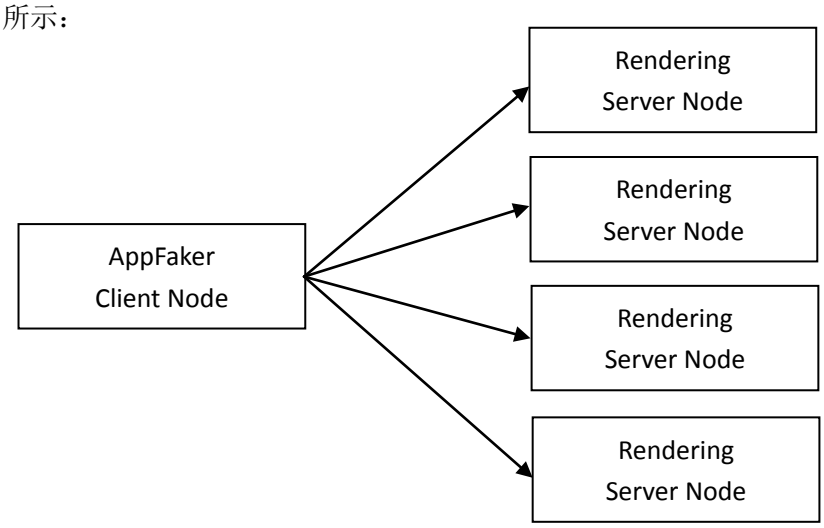
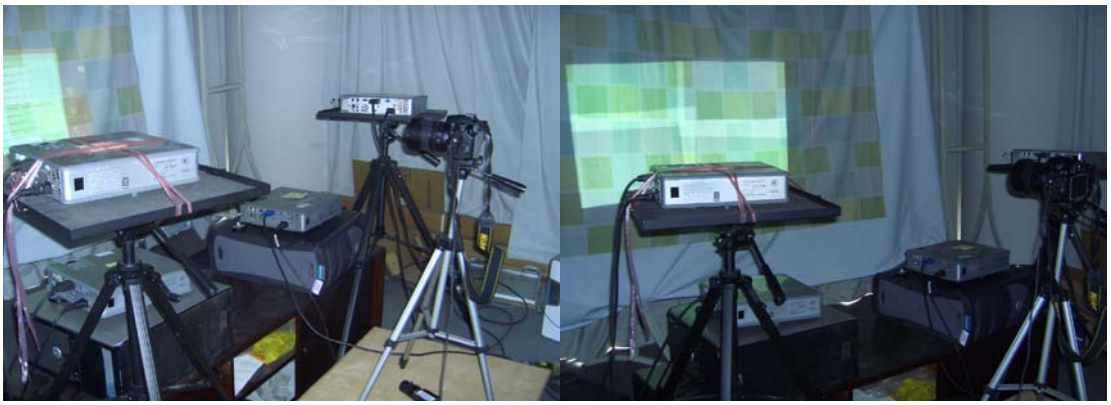


图 3-2 分布式渲染框架部署图



（a）　　（b）

（c）

图 3-3 用于多投影拼接的四台投影仪（图 a、b、c 是实验场景的实物图）

图 3-4 用于获取校正信息的摄像机

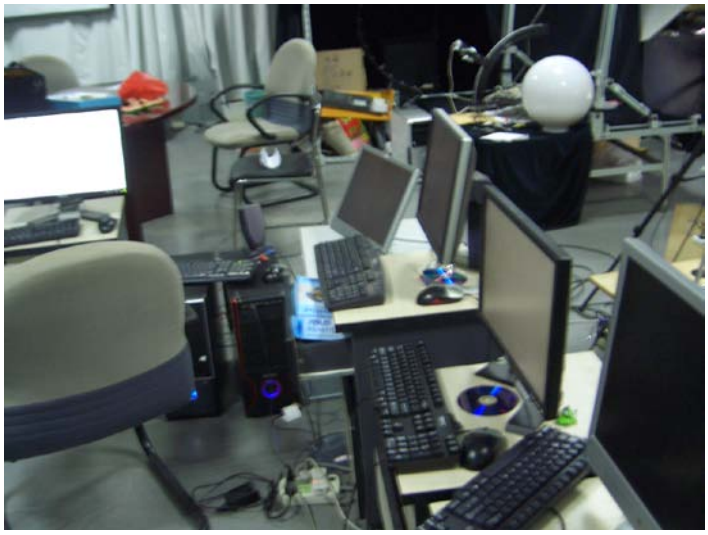


图 3-5 由五台普通台式 PC 组成的集群

3.4 本章小结

本章主要介绍了分布式渲染的研究现状及其实现的方式与途径，并且着重介绍了在本项目中所使用的分布式渲染框架 Chromium。

第四章　优化手段

4.1 分布式渲染系统优化综述

本项目设计的初衷是希望能够构建一个可以在复杂投影背景下进行多投影拼接的实时分布式渲染系统，尽管这种表述显得十分冗长，但简单而言就是希望通过上一章所介绍的分布式渲染框架 Chromium 集成有关几何校正、光度补偿、投影拼接和光度一致性的算法，并且希望通过各种可能且代价可以承受的方式进行整体性能的优化，从而达到实时性的要求。我负责了有关几何校正部分的算法移植，因此我对单个计算机集群节点上的算法优化也都是针对该算法的。优化的方案包括了：1、CPU 版本算法的单线程访问局部性（Locality of Reference）优化[17]；2、CPU 版本算法的静态线程池多线程优化；3、GPU 算法移植着色器语言（Shader Language）实现版本优化[18]；4、GPU 算法移植通用计算平台实现版本优化[19]。

此外，除了对于单个节点的本地算法优化之外，还尝试了对 Chromium 网络传输机制的优化。由于 Chromium 的分块排序 SPU（tilesort SPU）采用了同步机制同步计算机集群中各个节点对于 SwapBuffer 这一条 OpenGL 指令的执行，即各个节点刷新显示设备的显示缓冲区应该保持一致性。这一方面保持了系统的正确性，但同时也造成了系统性能的部分缺失。下文将会对各种实施的优化方案加以详细的介绍，由于并未将所有的优化方案都应用于最后的版本，因此下文也会给出各种优化方案针对本项目的具体优化效果，并加以个人的分析。

4.2 访问局部性优化

4.2.1 访问局部性介绍

所谓访问局部性又称为局部性原理，最初是指在计算机中运行着的程序在一段较短的时间内访问某一数据地址的次数。如今，他的含义已经扩展为时间层次（temporal）、空间层次（spatial）、等距层次（equidistant）以及分支层次（branch）。下面对同本项目相关的时间层次、空间层次分别加以简单的说明：

4.2.1.1 时间层次

时间层次的定义是指一个指定的内存地址在短时间内会被多次的访问，那么在两次相邻的访问间就存在着时间上的近似关系（proximity）。在这种情况下，通过某种特殊的高效存储设备对访问的数据进行缓存将能够有效地提高访问的速度。时间层次其实是空间层次的一个特殊情况，即当空间层次中所访问的位置都是同一个的时候即为时间层次。

4.2.1.2 空间层次

空间层次是指当某一段连续的的数据地址会在短时间内被陆续地访问，那么在这些连续的数据地址间就存在着空间上的近似关系。在这种情况下，有必要评估应该一次性地缓存多大的连续地址空间以提高后面的访问速度。

4.2.2 访问局部性的意义

访问局部性通常是出于以下三点的考虑：1、可预测性：当然，这只是提高计算机程序

运行时可预测行为的一种途径，通过提高预测性，可以延长处理器的串行执行周期，提高处理器的执行效率。2、计算机程序结构：同时，由于程序创建的过程中，相关的数据总是被存放在相邻的位置，因此可以利用空间层次的访问局部性对数据访问加以优化。3、线性的数据组织结构：由于程序中对于大规模的数据总是以数组的线性的数据结构存放，因此也可以进行针对性的访问优化。已有研究人员曾做过相关的研究，即程序访问局部性的强弱对于Cache 命中率有着极大地影响。

4.2.3 访问局部性在项目中的应用

在本项目中，访问局部性主要体现在对全局数据的局部缓存。由于每当 OpenGL 的指令发送到计算机集群渲染服务器节点时都会调用几何校正算法，而该算法需要访问一张全局的 P2I 映射表以实现源图像到投射图像的转变以及一些相关的配置参数等，因此使用局部变量对全局变量以及结构体内变量的缓存可以起到连续化访问数据的作用，从而提高 Cache 命中率，加快数据访问的速度。而实际的优化效果也是十分明显的，移植后的算法经过优化后，单机渲染的帧率从原先的 8fps 提高至 20fps。

4.3 线程池多线程优化

4.3.1 POSIX Threads介绍

本项目的多线程优化方案是基于 POSIX Threads，或 Pthreads，它是一个符合 POSIX 规范的线程技术。该实现在类 Unix 的 POSIX 系统版本上都可以得到，如 FreeBSD、NetBSD、GNU/Linux、MacOS 等。它定义了一套符合 C 语言规范的变量类型、函数、常量，并且包括了线程管理、互斥量、条件变量、锁同步等机制，这些功能在网络传输异步优化的实现中也被用到，具体实现内容会在后文提到。同时，POSIX 信号量（semaphore）也可以同 POSIX thread 协同工作，但并不属于其中的一部分。

4.3.2 线程池（Thread Pool）的选用

在本项目中，几何校正算法的核心部分是通过 P2I 映射文件将源图像中每个像素变换到投影图像的位置，该映射文件是通过预先对投影反射背景的几何形变进行计算并建立照相机空间到投影空间的关联而生成的。由于算法预先考虑到了并行优化的方案，因此算法的实现是遍历投影图像的每一个像素位置然后根据该位置和映射文件再找到源图像中对应位置的像素，并将其值写入投影图像中。可见算法如果采用并行优化后将不会出现写冲突，尽管可能造成读冲突，但由于在几何校正阶段的处理并不会对源图像的像素数据进行修改，因此也不会因为读到脏数据而产生歧义。这一点对于并行优化，包括 CPU 和 GPU 版本的优化都有一个好处，即不需要对数据加锁，从而不会产生由于并行优化而导致的额外性能开销。

在实际的实现中，多线程优化的手段主要是通过为每一个线程指定一个线程 ID，然后根据线程的数量 N 将任务进行划分，在本项目中即将原先的投影图像划分成 N 个部分，每个线程只需要根据自己的线程 ID 负责处理同该 ID 相关联部分的投影图像。此外，考虑到每一帧的图像的处理过程都会调用几何校正算法，因此如果每帧处理时都临时创建线程并在处理完成后销毁线程的话，显然可行性不强，会对渲染的实时性有着极大地影响。这时自然会想到线程池技术，即在系统运行伊始进行初始化配置的过程中就建立一个线程池。所谓线程池即是指通过一个线程管理器维护一定数量的线程数，这些线程在没有任务需要执行的时候便会悬挂起来，但并不会被销毁，而当有任务到来需要处理时，线程管理器先不用急于创建新的线程，而是在线程池中寻找是否有空闲悬挂起来的线程。如果有，则直接将任务分配给他。如此一来可以在一定程度上极大地减少由于频繁开创线程和销毁线程而造成的巨额的开销代价。线程池技术的抽象概念可以参见下图所示：

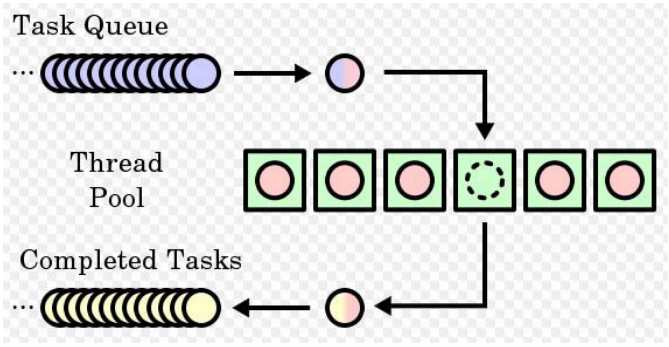


图 4-1 线程池技术的抽象概念

考虑到本项目的实际特点，即数据规模在整个系统运行期间比较稳定，变化不大，即处理的图像数据大小不变，因此线程池的实现是采用静态线程池的方案。所谓静态线程池是指线程的数量是在初始化时指定好的，不会随着程序运行而动态地调整。这样做可以避免繁杂的线程管理和由于线程数量切换而造成的显示帧率抖动，毕竟平滑的播放模式可以给观众带来更好的视觉体验。

4.4 GPU算法移植优化

GPU（Graphics Processing Unit）中文译为图形处理单元，他是一种专门用于将数据转换填充为二维图像像素数据的硬件设施[20]。现代图形处理单元可以高效的进行计算机图形方面的计算，并且他们拥有着高度平行化的结构可以帮助他们在进行一些复杂的代数运算时比传统意义上的 CPU 高效得多。如今，普通的个人电脑都已配备有图形处理单元，他们可以位于显示卡上也可以集成于主板间。随着统一化架构的概念被提出和应用，以及图形处理单元对于通用计算能力的增强，越来越多研究人员尝试将大规模的计算任务移植到 GPU 上，并且多数都获得了不错的加速比。与此同时，随之而来的是各个图形硬件生产商相继推出自己的 GPU 通用计算平台，吸引更多的研究人员和程序员将目光投向 GPU。

4.4.1 GPGPU概念介绍

GPU 已逐渐成为现今性价比最高的计算设备，越来越多的研究人员和开发人员开始对如何将其巨大的计算潜力用于同于计算的目的产生浓厚的兴趣，于是 GPGPU（General Purpose computing on the GPU）的概念应运而生。随着近年来 GPU 的迅猛发展，如今图形显卡已可以提供极高的访存带宽和惊人的计算能力，并且仍在以高速不断增长，其增长速率甚至已超过了摩尔定律，之所以出现这样的局面正是由于 GPU 和 CPU 两者之间不同的设计架构所造成的。此外，现代 GPU 的发展之所以如此蓬勃很重要的一点是由于它的可编程，由于最初的 GPU 是不可编程的，极大地限制了它的应用范围，固定的图形管线也使它缺乏灵活性。然而，随着 GPU 体系结构的革新与发展，从顶点可编程，到片段可编程，再到如今的统一架构，GPU 经历了从面向图形应用到面向通用应用的过程。而伴随着硬件发展，它的开发平台也在朝着通用化的方向发展，从最早的汇编着色语言，到 C 语言风格的着色语言，再到如今的通用计算平台，随着新的平台推出，GPGPU 的门槛也在不断降低。如今在 GPU 上开发程序已不再需要图形学的背景了，这无疑将会吸引更多的程序员们走进 GPGPU 的世界，而这也必将促进 GPU 自身的进一步发展。

4.4.2 Nvidia Cg着色语言优化

着色语言是一种载入着色器中的特殊程序语言，不同于普通的程序语言，它有着特殊的数据类型，如颜色、法向量等等。由于为了应用于不同的显卡着色器，多种不同的着色语言相继被发布，如 HLSL、GLSL、Cg 等等。考虑到本项目的开发是在 GNU/Linux 系统下，并

且由于 Chromium 分布式渲染框架最新的版本是在 OpenGL1.5 的版本上开发出来的，而 GLSL 是直到 OpenGL2.0 的时候才被支持的。因此，考虑到平台支持性和系统兼容性的因素，Cg 成了本项目唯一的选择，由于它平台支持性好（已知的有 Windows、Linux 和 Mac 的版本）并且同时支持 OpenGL 和 D3D。

4.4.2.1 Cg着色语言介绍

Cg 代表用于图形的 C 语言（C for graphics），Cg 语言本身也是基于 C 语言的。它的设计初衷是为了给图形硬件提供对渲染物体形状、外观和运动的可编程控制能力。从广义上讲，这种类型的语言被称为光照着色语言，但 Cg 可以做光照着色以外的很多事情。例如，Cg 程序可以实现物理模拟、混合和其他的非光照任务。Cg 和其他光照着色语言与传统的编程语言不同，它是基于数据流模型的。在这样的模型里，计算的发生是为了响应流经序列处理过程的数据。Cg 程序运行在渲染一幅图像时被处理的顶点和片段上，顶点和片段信息从一边流入，经过程序里的变换后从另一边流出，而具体的变换实现则是由程序员指定的。

4.4.2.2 Cg着色语言版本实现

在实现的过程中，由于着色语言自身的限制，必须将算法表述为图形学的形式，从而利用着色器进行计算。这其中有些概念需要先加以简要的说明：

1、离屏渲染（Offline Rendering）：传统的渲染方式中，数据流经图形渲染管线后会被写入系统默认的帧缓冲中并刷新到显示设备上。而离屏渲染的方式是指将图形渲染管线的输出重定向到自定义的一块缓冲区中，如此以来，输出的结果数据将不会被刷新到显示设备，而应用程序则可以通过图形语言（本项目使用的是 OpenGL）提供的方法对缓冲区进行访问。

2、帧缓冲对象（Frame Buffer Object）：帧缓冲对象简称 FBO，它扩展了 OpenGL 对于离线渲染的支持，包括渲染到纹理（Rendering to Texture，RTT）。

接着本文将对 Cg 实现的关键步骤加以说明：

1、 创建纹理：纹理用于存放数据，包括输入的数组类型参数、计算输出的结果数组。在本项目中，输入的数组参数主要有两种，一种是全局的 P2I 映射数组，一种是每一帧需要处理的源图像数组；而输出的结果数组则存放了几何校正后的投影图像。
2、 创建 FBO：使用 OpenGL 的离线渲染功能。
3、 绑定：将纹理和 FBO 通过指定的挂载点绑定在一起。
4、 设置投影变换矩阵：为了能够使纹理中的每项数据经过图形渲染管线的插值后能够与片段着色器中的每一个片段按照索引值一一对应，需要将投影模式设置为正交投影。
5、 Cg 设置：主要是运行 Cg 程序的一些配置步骤和参数配置等等，可以参考相关的 Cg 教程。
6、 触发着色器：即执行几何校正运算，如同一般的 GPGPU 编程方式一样，几何校正算法是通过片段着色实现的，而将纹理数据和片段一一对应则是通过绘制一张贴有该纹理，大小为显示窗口的大小。
7、 读取帧缓冲：将结果数组由显存读入内存中。

过程如图 4-2。

在本项目中，由于每一帧的图像校正都需要调用该算法，因此，对于一些全局的常量应该尽可能的减少其在内存和显存间的传送。因此，本项目在实现时只在系统初始时完成对 Cg 程序运行的配置以及 Cg 全局参数的传送，而每一帧计算的过程中只需要传送源图像数据到显存，从而减少了因传送重复数据而造成的不必要的时间消耗。

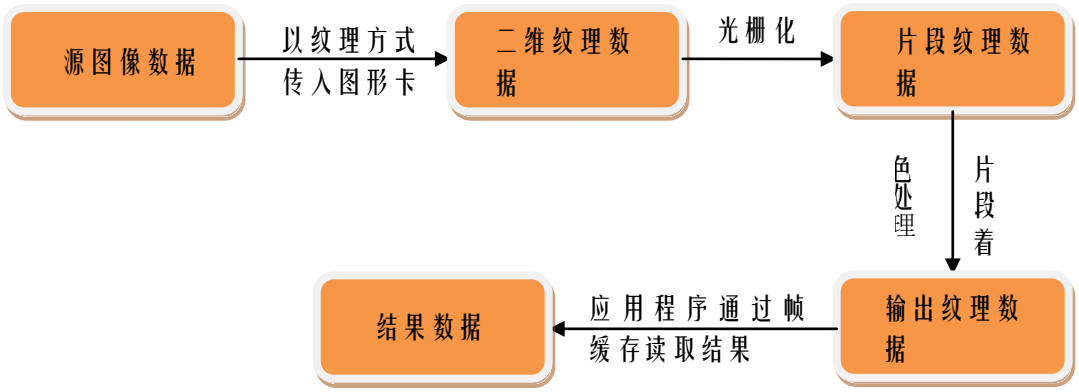


图 4-2 着色语言实现过程简图

4.4.3 CUDA通用计算平台优化

4.4.3.1 CUDA通用计算平台介绍

CUDA 是 Nvidia 的 GPGPU 模型，它使用 C 语言为基础，可以直接以 C 语言的形式，写出在显示芯片上执行的程序，而不需要去学习特定的显示芯片的指令或是特殊的结构。Nvidia 的新一代 GPU，包括 GeForce8 系列及往后的显示芯片都支持 CUDA，并且还提供了支持 Windows、GNU/Linux 的版本。不同于以往通过着色语言的 GPGPU 实现方式，CUDA 通用计算平台给了开发人员更多操作硬件的灵活性，例如通过指定每个 SM（Stream Multiprocessor）的执行线程数进行负载均衡调整，通过指定数据存放的位置（片内或片外）提高对频繁访问数据的读写速度，通过访问地址对齐提高对片外数据的读写速度。因此，CUDA 相对着色语言更适合用于通用计算。

在 CUDA 的架构下，一个程序分为两个部份：host 端和 device 端。Host 端是指在 CPU 上执行的部份，而 device 端则是在显示芯片上执行的部份。Device 端的程序又称为 "kernel"。通常 host 端程序会将数据准备好后，复制到显卡的内存中，再由显示芯片执行 device 端程序，完成后再由 host 端程序将结果从显存中取回。如图所示

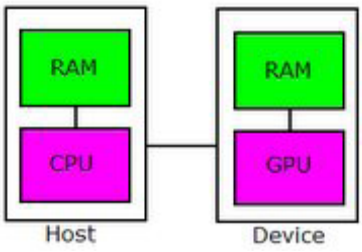


图 4-3 CUDA 主从设备示意图

4.4.3.2 CUDA通用计算平台实现

算法的具体实现和 Cg 版本的片段着色实现十分相似，唯一的区别是片段着色程序是通过片段的索引值查找纹理中的数据，而 CUDA 程序中则是通过线程 ID 以及块 ID 计算出查找数据的索引值。其余的实现同片段着色程序的实现类似。

4.5 Chromium网络传输异步优化

优化的最后部分是有关于系统整体性能的优化。通过之前的各种优化手段，计算机集群中单个节点的处理速度已经非常之快，在实验室的实际测试环境下，几何校正算法部分已经可以达到 90+fps，即使是运算更为复杂的光度补偿的处理速度经过 GPU 通用计算平台加速后也已经达到了 40+fps。然而，当我们对整个分布式渲染系统进行测试的时候（1 台运行应用程序的客户端节点和 4 台渲染服务器），实际的运行结果却只有 15fps 左右，因此不得不对 Chromium 的整体运行机制进行重新的评估，并找出制约系统渲染效率的瓶颈所在。

通过阅读有关 Chromium 的论文和 Chromium 的分块排序 SPU（tilesort SPU）的源代码，了解到 Chromium 的实现是基于立即模式的，即每一帧都会发送数据给渲染服务器，这相对于保留模式，其对网络带宽的需求更大，并且当每一帧的发送数据量很大时将会极大地影响到系统的整体渲染效率。在本项目中，由于需要对高分辨率的图像进行处理，因此每一帧都会向计算机集群中所有的渲染服务器发送分块后的图像，此外还分块之间还会保留与部分的重叠区域，以供后续的投影拼接使用。但即使忽略这些冗余数据量，每一帧中都需要发送一幅高分辨的图像，以 1280*1024 的分辨率为例，在实验室局域网条件下（千兆网），图像的传输速率最多也只能达到每秒 34 帧。而考虑到实际传输的数据量还要更大并且网络带宽的实际利用率无法达到百分之百（信号良好的局域网实际测得的利用率最高接近于百分之九十），因此网络传输实际上是限制分布式渲染系统渲染效率进一步提高的制约因素。已知可以根本解决的问题的方案还是需要更换硬件配置，比如使用万兆网络替代现在的千兆网络或者使用高速的专用网络，国外一些实验室的分布式渲染平台往往使用高性能的专用网络以达到渲染系统实时性的需求。但考虑目前无论是万兆网络还是高速专用网络，所需的硬件配置价格较高（网卡约几千元，而交换机则需要上万元），因此本项目中并没有采用这一方案。但在项目后续的工作中应该会购置新的硬件配置，以从根本解决网络瓶颈的问题。

本项目中，在硬件制约的条件下希望能够通过软件层对网络尽可能地进行优化。在仔细研究了 Chromium 分布式渲染系统网络传输的机制之后，发现的确还存有提升的空间，下面将对改进优化的方案进行具体的阐述：

Chromium 采用的是立即模式，即每一帧都会向渲染服务器发送渲染指令，然而由于不同的渲染服务器每一帧的处理速度无法保持绝对的一致，因此 Chromium 提供了多种同步机制对渲染输出进行同步。这一措施保证了系统渲染的正确性，同时也提供了对异构计算机集群的支持，但另一方面却导致了对应用程序的阻塞，即在渲染服务器计算的同时，网络处于闲置的状态，只有当所有的渲染服务器渲染完一帧图像并且同步后，运行应用程序的客户节点才会从阻塞中被唤醒继续运行并发送下一帧需要处理的源图像数据。为了能够在渲染服务器进行渲染计算的同时使网络不闲置，需要将网络数据传输和渲染计算有串行关系转为并行关系，也可以称之为异步化。从而当渲染服务器计算一帧源图像的同时，下一帧的源图像已经开始向该渲染服务器传送，并且渲染服务器会在本地开辟一块缓存用于存放已经抵达但还来不及处理的源图像数据。当渲染服务器处理完当前帧的源图像后便可以直接从本地的缓存中读取数据而不必再等待网络数据传输。此外，Chromium 的同步机制得到了保留，因此保证了网络数据传输同渲染计算异步化后显示的结果不会出现不一致的现象。

在具体的实现中，为了保持对于分布式渲染系统其它模块的透明，网络传输的优化也是通过 Chromium 的 SPU 机制实现的，即 AsynServer SPU 和 AsynClient SPU，其中 AsynServer SPU 是位于渲染服务节点的 SPU，而且必须位于该节点 SPU 链条中的第一个，而 AsynClient SPU 则是位于应用程序客户节点上，并且必须位于分块排序 SPU（tilesort SPU）之前。下面具体说明两个 SPU 各自的用途、实现和两者之间的关联。

1、 AsynClient SPU：该 SPU 主要实现了通过多线程方式对同步指令 SwapBuffer 的调用次数

进行记录。具体的实现是当应用程序调用同步指令 SwapBuffer 后，AsynClient SPU 中负责处理该 OpenGL 指令的函数会通过对一个计数器执行加一操作对该指令的延迟次数进行纪录，然后函数会返回即意味着该同步指令不在受同步信号的阻塞。而真正负责对渲染服务器节点进行同步指令 SwapBuffer 调用的是由另外一个线程完成的，它创建于分布式渲染系统初始化时，并且会根据当前剩有的 SwapBuffer 调用次数，对渲染服务器节点进行显示刷新同步。项目中还通过使用互斥量（mutex）避免两个线程对于计数器可能造成的读写冲突。如此一来，对于渲染服务节点的显示刷新同步将不会阻塞应用程序的运行，从而应用程序可以无阻塞地调用 DrawPixel 函数并将源图像数据不停顿的发送给渲染服务器节点。

2、 AsynServer SPU：该 SPU 主要实现了对网络数据的本地缓存以及同步 SwapBuffer 指令调用后重定向源图像数据为本地缓存数据栈中最早的图像数据。该实现主要是针对同 AsynClient 间的协调通信，以达到对后续 SPU 模块的透明化，如此可以在不必修改算法的前提下，将原本串行的网络数据传输同渲染计算并行化。而且，对于今后在该分布式渲染平台上开发的新的模块也可以在不必考虑异步化实现细节的情况下，编写自己的模块。

路程如图 4-4

4.6 优化结果分析

表 4-1 优化结果分析对照表

优化方案	优化效果
访问局部性	将显示帧率从 8fps 提升至 20fps，而在实验室设备上的处理速度达到 90fps。
Nvidia Cg 着色语言	通过修改通道数，保持 8fps，由于优化效果不明显，并未移植到实验室的计算机集群平台上。
CUDA 通用计算平台	在实验室设备上的处理速度由 90fps 提升至 93fps。
静态线程池	将显示帧率从 20fps 提升至 43fps，尚未移植到实验室设备上。

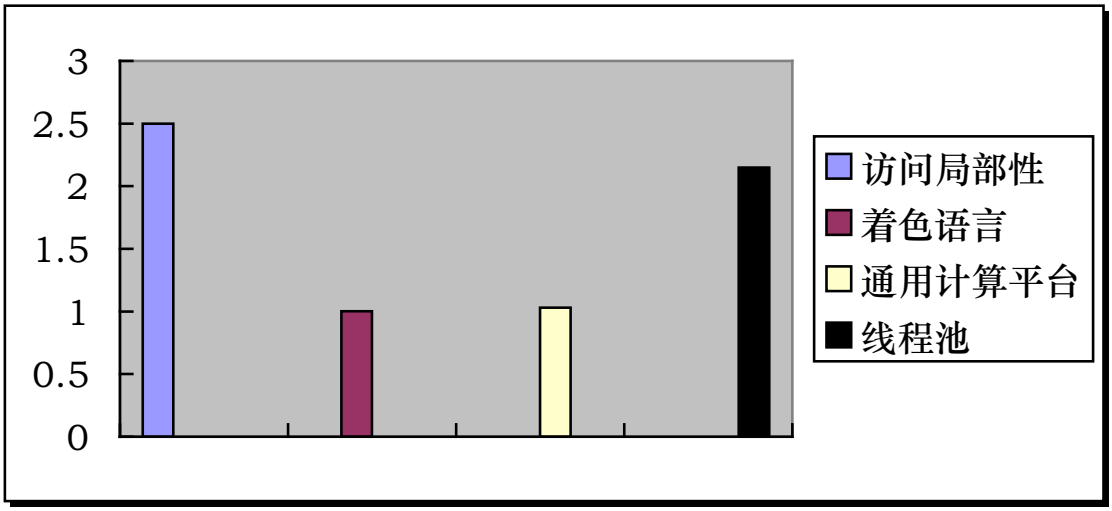


图 4-6 优化效果对比图表

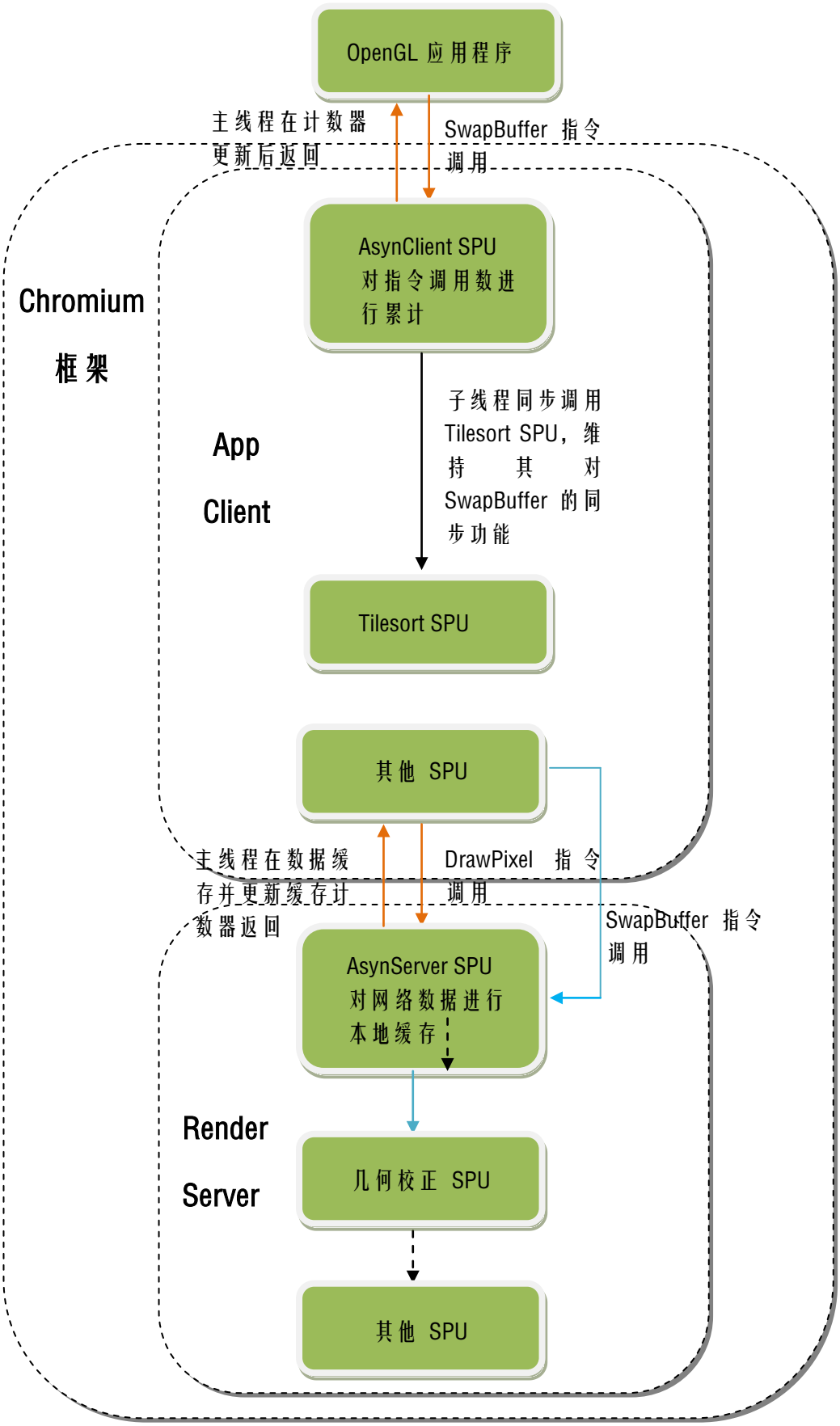


图 4-4 网络优化过程简图

通过实际实现验证比较后，最终采用的单机优化方案有访问局部性和静态线程池，之所以没有采用 GPU 优化的方案并不是 GPU 加速本身存在任何问题，主要是由于算法自身的特性所导致。由于几何校正算法针对每个像素的处理极为简单，即从源图像中取得数据并赋给输出图像对应位置的像素。尽管 GPU 的流处理器数量可观可以对程序进行并行化，但一方面其执行频率却低于 CPU。此外，由于每一帧都需要从内存向显存传送大量的源图像数据，尽管 GPU 同显存间的带宽远高于 CPU 同内存，但从内存到显存的带宽却低得多，因此这成为了制约其速度提升的主要瓶颈。最后，由于源图像的数据量大，并且每条单独的执行线程对其访问是随机的（依赖于映射文件中描述的映射关系），因此也没有办法对齐进行 GPU 片内高速存储的缓存优化。基于以上这些算法固有的特性，几何校正算法移植到 GPU 上的效果并不理想，这一方面也验证了并非所有的程序都是适合于基于现在计算机体系的并行化。因此，最终采用了 CPU 版本的并行化，依靠多线程和多核处理器。相对于 GPU，这种方案省去了从内存到显存的时间消耗，由此，它的加速效果也最为明显。

4.7 本章小结

本章主要介绍了在项目中所尝试的各种优化方案与手段并对每一种优化方案给予具体的介绍，同时对优化的结果进行了比较并加以分析。

第五章　结论与展望

本项目通过使用分布式渲染框架与学习几何校正算法原理，实现了多投影在复杂环境背景下的应用，这其中包括了对 Stanford Chromium 渲染框架的搭建、几何校正算法的 Linux 移植以及算法同分布式渲染框架的整合，使得该算法在计算机集群平台上运行稳定。同时，还尝试了多种手段对分布式渲染系统效率的提升，包括 Cache 优化、多线程、GPU 移植以及影响系统整体性能的网络传输优化。经过对各个优化方案效果的比较和分析，对本项目的优化制定了效果最为明显的优化方案。同时，尽管 GPU 版本的优化并未采用，但由于在实现时考虑到了后续应用通过 GPU 加速的可能，已经将其封装成接口定义清晰的库文件，便于今后在这个平台上的应用开发。

本项目的主要成果如下：

- 完成分布式渲染框架的搭建，通过阅读 Chromium 框架的相关论文，并且学习了 Python 脚本语言，完成了在实验室局域网环境下的分布式渲染系统部署和配置。
- 完成了几何校正算法的移植，通过阅读实验室学长在这方面所完成的工作报告，熟悉并了解了单应性矩阵方法和棋盘格提取角点方法。并将该算法移植到 Linux 环境下，同 Chromium 框架集成。
- 完成了几何校正 GPU 算法移植，学习了 Nvidia Cg 编程和 CUDA 通用计算编程，学习了 OpenGL 的离线渲染方法，了解了 OpenGL 实现离线渲染的各种机制。并且分别实现了基于 Nvidia 着色语言版本的算法移植和基于 CUDA 通用计算平台的算法移植。
- 完成了网络传输优化的设想方案，在了解了 GPU 通过 Multi-thread 掩盖访存 latency 的机制之后，几何 Chromium 自身的 Lazy Update 同步机制，提出本地 Cache 网络数据的方案并通过在 Render Server 和 App Client 两端各插入一个 SPU 实现这一目的。
- 完成了对 CPU 版本几何校正的多线程并行优化，使用 POSIX thread 实现了几何校正的静态线程池优化方案。
- 整合优化方案，完成最终的 Demo 制作。

尽管对于分布式渲染系统单个节点上的运行速度已有了显著的提升，但网络部分的优化却并没有实质性的改观，原因在相关的章节已经做了详细的阐述。因此，后续的优化工作将着眼于网络部分，选择的方案有对分布式渲染框架的重构，由于采用了立即模式，并且现在的数据传输是点对点，因此可以考虑数据广播以及保留模式，这种方案有着经济性的好处，但同时也具有一定的风险，一者开发难度较大，再有优化效果未必明显。还有一种方案就是对网络硬件配置进行提升，无论通过从硬件层次的提速，如替换千兆网为万兆网，还是使用高速专用网，如 myrinet，都需要更换硬件配置，这种方案尽管需要的费用是相对昂贵的，但是考虑到能够换取直接的性能提升还是值得的。因此，各种方案还应该进一步视实验室的实际需要与项目发展而定。

参考文献

[1] Ronald T. Azuma: A Survey of Augmented Reality, Teleoperators and Virtual Environments, August 1997

[2] BROWN, M., MAJUMDER, A., AND YANG, R. 2005. Camerabased calibration techniques for seamless multiprojector displays.IEEE Transactions on Visualization and Computer Graphics 11, 2, 193–206.

[3] BIMBER, O., AND RASKAR, R. 2006. Modern approaches to augmented reality. In ACM Transactions on Graphics : ACM SIGGRAPH 2006 Courses, ACM Press, New York, NY, USA, 1.

[4] ZHANG, Z. Y. 2000. A flexible new technique for camera calibration.IEEE Trans. Pattern Analysis and Machine Intelligence 22,11 (Nov.), 1330–1334.

[5] SHU, C., BRUNTON, A., AND FIALA, M. 2003. Automatic grid finding in calibration patterns using delaunay triangulation.Tech. rep., National Research Council of Canada, Aug.

[6] WATSON, D. F. 1981. Computing the n-dimensional delaunaytessellation with application to voronoi polytopes. The Computer Journal 24, 2, 167.

[7] Raskar, R., Welch, G. Cutts, M., Lake, A., Stesin, L., Fuchs, H.: The Office of the Future:A Unified Approach to Image Based Modeling and Spatially Immersive Displays. In:Proc. of SIGGRAPH'98 (1998).

[8] BIMBER O,GRUNDHÖER A., SEEGER M., HÄNTSCH F.: Dynamic Adaptation of Projected ImperceptibleCodes, Proc. of IEEE International Symposium on Mixed and Augmented Reality (2007).

[9] Steven Molnar, Michael Cox, David Ellsworth, Henry Fuchs: A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Application (1994).

[10] R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh. Load Balancing for Multi-Projector Rendering Systems. Eurographics/SIGGRAPH workshop on Graphics hardware, 107-116, 1999.

[11] R. Samanta, T. Funkhouser, K. Li, and J. Singh. Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs. Eurographics/SIGGRAPH workshop on Graphics hardware, 99-108, 2000.

[12] H. Chen, Y. Chen, A. Finkelstein, T. Funkhouser, K. Li, Z. Liu, R.Samanta, and G. Wallace. Data Distribution Strategies for High Resolution Displays. Computers and Graphics Vol. 25, 811-818,2001.

[13] Y. Chen, H. Chen, D. W. Clark, Z. Liu, G. Wallace, and K. Li. Software Environments For Cluster-based Display Systems. IEEE Symposium on Cluster Computing and the Grid, 202-210, 2001.

[14] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner,and J. T. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. SIGGRAPH, 693-702,2002.

[15] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan. Distributed Rendering for Scalable Displays. IEEE Supercomputing, 2000.

[16] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A Scalable Graphics System for Cluster. SIGGRAPH,129-140, 2001.
[17] 王恒娜: 浅析访问局部性原理在 Cache 存储系统中的作用, Journal of Anhui University Natural Science Edition, January 2005
[18] Randima Fernando, Mark J. Kilgard: The Cg Tutorial-The Definitive Guide to Programmable Real-Time Graphics
[19] 张舒, 褚艳利: GPU 高性能运算之 CUDA, 中国水利水电出版社
[20] Minh Tri Do Dinh: GPUs - Graphics Processing Units, Vertiefungsseminar Architektur von Prozessoren, SS 2008, Institute of Computer Science, University of Innsbruck July 7, 2008

谢辞

本论文的完成，得益于上海交通大学软件学院数字艺术实验室的杨旭波老师的指导。从选题的确定、论文资料的收集、论文框架的确定、开题报告准备及论文初稿与定稿中，杨老师都倾注了的大量心血，在此表示由衷的感谢！

在这里，还要特别感谢实验室项天远学长和黄嘉浩同学，谢谢你们在项目遇到困难时给予我的支持与建议，也感谢你们同我一起为这个项目而付诸的努力。

最后，我要感谢我的父母，如果不是他们给予我经济、生活等各个方面的支持，我不可能走到今天，我会为你们继续努力的生活学习，用我的成绩来报答你们的养育之恩。