一种基于 OGRE 图形引擎的实时分布式渲染系统

孙益辉,陈福民,王海峰
SUN Yi-hui,CHEN Fu-min,WANG Hai-feng

同济大学 计算中心,上海 200092
Computing Centre,Tongji University,Shanghai 200092,China
E-mail:jonahsyhn@yahoo.cn

SUN Yi-hui,CHEN Fu-min,WANG Hai-feng.Real-time distributed rendering system based on OGRE.Computer Engineering and Applications,2008,44(31):102-103.

Abstract: Real-time and interactive virtual reality has been widely applied.Real-time distributed rendering system can solve the time bottleneck and improve real-time performance and output resolution.With research and analysis of distributed rendering theory,a real-time distributed rendering framework based on Sort-First and synchronization mechanism are designed according to design pattern.At the same time,the system using OGRE graphics engine is implemented in practice.

Key words: distributed-rendering;Sort-First;synchronization;Object-oriented Graphics Rendering Engine(OGRE)

摘 要:虚拟现实的实时交互得到了越来越广泛地应用,实时分布式渲染有效地解决了普通 PC 机渲染的时间瓶颈问题,提高了系统实时渲染性能和输出分辨率。在对分布式渲染原理研究分析的基础上,依据设计模式的思想,设计了一种基于 Sort-First 结构的实时分布式渲染框架和同步机制,并在实践中应用 OGRE 图形引擎实现了该系统。

关键词:分布式渲染;Sort-First;同步;OGRE
DOI:10.3778/j.issn.1002-8331.2008.31.029 **文章编号**:1002-8331(2008)31-0102-02 **文献标识码**:A **中图分类号**:TP317.4

目前,虚拟现实技术应用越来越广泛,场景规模在不断增大的同时也变得更复杂,对系统硬件提出了更高的要求,用普通 PC 单机实现实时交互渲染比较困难,而大型图形工作站成本较为昂贵,难于普及和扩展。在这种情形下,实时分布式渲染是一种较好的替代方法,它将渲染任务分配到各客户机,由多台渲染客户机协同完成每帧图像的渲染。OGRE (Object-oriented Graphics Rendering Engine)是用 C++语言开发的跨平台开源图形引擎,它对底层系统库(Direct3D 及 OpenGL)细节进行了有效的抽象,提供了基于现实世界对象的接口和其它类[1],在该图形渲染引擎上可以较快地开发各种应用程序。系统设计的目标是扩展 OGRE 单机渲染模式,将渲染、网络、物理、声音、输入等引擎构架成一个通用的分布式渲染平台,为应用程序开发提供接口和支持。

1 并行渲染框架设计

实时分布式渲染系统采用基于 C/S 架构,渲染服务器和客户机通过局域网相互连接,并用 RakNet 网络引擎通信。RakNet 是 RakkarSoft 公司所开发的基于 UDP 协议软件包, 为应用程序提供高效、有序、可靠的传输服务。

1.1 Sort-First 体系结构

根据 Molnar 等理论, 按几何图元归属判断时机将并行绘制系统分为 Sort-First、Sort-Middle 和 Sort-Last 三类[2]。在绘制流水线几何处理之前判断的并行图形绘制系统属,它在几何处理阶段决定图元对象在屏幕上的对应位置,每个参与并行绘制的流水线仅负责屏幕的一部分区域。任务划分按策略分为动态和静态两种方法,扩展基于 OGRE 的实时分布式渲染系统设计采用 Sort-First 的静态策略, 客户机仅渲染事先分配的一块大小均匀区域,当屏幕是 2 分屏时,每个渲染客户机只负责渲染 1/2 块屏幕区域,渲染结果经服务器同步处理后输出。

1.2 分布式渲染分屏

OGRE 引擎 3D 场景渲染处理中,相机(Camera)一般采用视棱锥(Frustum)透视投影,位于远近两个裁剪面之间的视棱锥才进行投影变换和参与渲染。实时分布式渲染分屏将总场景视棱锥根据分屏要求沿相机射线方向进行划分, 如图 1 所示 1×2 分屏,把原场景视棱锥分割成 2 个非正棱锥,分别分配给 2

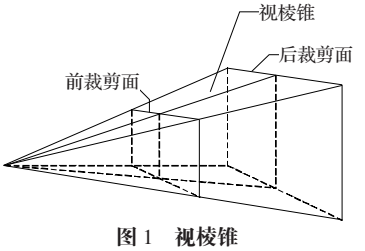

图 1 视棱锥

基金项目:上海市重大科技攻关项目(No.051111026)。
作者简介:孙益辉(1975-),男,博士生,主要研究方向:多媒体、分布式渲染、虚拟现实;陈福民(1944-),男,研究员,博士生导师,主要研究方向:虚拟现实和多媒体;王海峰(1982-),男,硕士生,主要研究方向:分布式渲染和多媒体。
收稿日期:2007-12-10 **修回日期**:2008-03-03

个不同渲染客户机，每个客户机场景的相机位置、方向、前后裁剪面坐标等保持一致但互相错开，然后用所分配的视棱锥进行透视投影和渲染。3D 中用一个 4 阶投影矩阵调整、控制分屏和投影中心位置，4 阶投影矩阵形如：

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}$$

其中 m_{00} 和 m_{11} 分别对场景的 x 与 y 方向进行缩放，m_{02} 及 m_{12} 则调整各分屏投影中心位置，它们赋予不同的参数，就可以获取不同的场景位置[3]，如 m_{00}=2，m_{11}=2，此时将场景放大为原来的 4 倍，即 4 分屏，如果渲染客户机的投影矩阵设置 m_{02}=1，m_{12}=1，则它将输出场景右上部分。

1.3　层次设计与抽象工厂模式

应用程序设计独立于底层引擎是本系统设计的一个重要目标。通过这种隔离式设计，有利于上层代码在不同平台的移植；同时，便于代码维护和升级。系统设计将 OGRE 与其它图形渲染引擎的共有属性和方法加以抽象，即在 OGRE 图形引擎上再进行封装，组成通用接口层。系统上层是独立的应用程序，它调用由引擎抽象而来的通用平台接口，接口层下面则是各种具体引擎，如本系统所采用的 OGRE，它调用 OpenGL 或 Direct3D 函数具体绘制。如图 2 所示，层次设计中使用了设计模式中的抽象工厂（Abstract Factory）模式，它提供了用来创建一系列相关或相互依赖对象的接口[4]，OGRE 引擎在其自身实现中也使用了此设计模式，体现在它的 Plug-in 机制。一方面把与引擎相关部分和无关部分实现隔离，如底层渲染系统既可以使用 OpenGL，又可以使用 DirectX；另一方面便于系统功能的扩充，如其场景管理既可用二叉空间分割树（BSP）方式，又可用八叉树（Octree）[5]。通过这种层次设计，可方便地替换 OGRE 等底层引擎，平台构建极具灵活性。

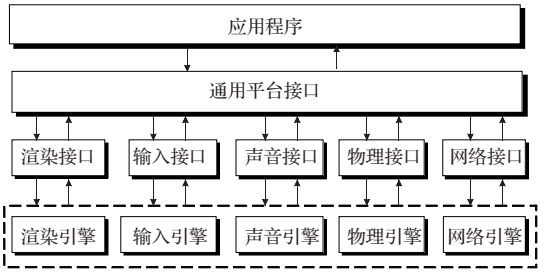

图 2　系统层次结构

2　系统实现

2.1　分布式渲染流程

实时分布式渲染系统启动时根据配置创建一个合适的场景管理器，然后再自动创建一个根节点对象，在这个根节点下面可以创建多个场景节点，每个场景节点下又可创建其子节点，3D 渲染实体就被附加到不同的场景节点上，组成一棵场景树。场景节点内部保存一个可移动对象列表，实体继承可移动对象列表，把实体附在场景节点上；同时，实体内部包含子实体列表，它继承于 Renderable，场景更新时，首先调用场景节点 addToRenderQueue 方法，这个方法内部迭代可移动对象列表的每一项，调用可移动对象的更新渲染队列抽象方法，从可移动对象继承的类均实现此方法。如实体，在实体内部迭代子实体列表每一项，将每个子实体送入渲染队列中去，完成渲染队列的更新工作。

2.2　同步控制

基于 Sort-First 静态任务划分不可避免地涉及负载均衡问题，随着场景变化，容易出现几何图元分配过于集中，导致其中的几个图形流水线过载，影响渲染的实时性；其次，当参与并行绘制的几何处理器数目过多或在负载严重失衡时，可能导致大量的处理器资源空闲[6]。渲染客户机之间的负载不平衡情况导致它们完成各自渲染任务所花时间不同，在各配置相同的条件下，负载大的渲染客户机比负载小的所花时间要长，各客户机在渲染后输出时不同步导致画面不完整或撕裂，不能满足应用的需求。解决这个问题，必须设计一个使各渲染客户机同步输出的机制。图 3 是本系统所设计并实现的一种帧同步算法，渲染服务器首先通过 UDP 网络协议组播需要同步的数据，如各种实体的位置、方向、角度等信息，渲染客户机收到相关数据后开始渲染，但在翻转之前即交换前后端缓存区时发送一个标识给渲染服务器，表示当前帧渲染任务是否已经完成。当渲染服务器收到所有渲染客户机的反馈标识或超时后，渲染服务器发送下一帧数据，指示各渲染客户机翻转并开始新一帧渲染，保证每个渲染客户机输出时严格帧同步，渲染速度快的客户机总是在等待速度慢的，在这种情形下，效率最差或负载最重的渲染客户机决定系统的整体性能[7]。

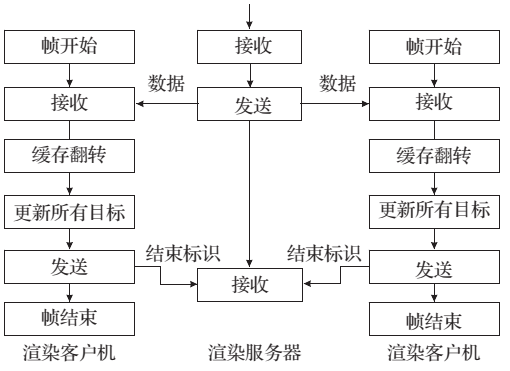

图 3　同步处理

2.3　信息处理分类

在实时分布式渲染过程中，渲染服务器与染客户机通过不同的信息流相互合作与协调，共同完成渲染任务。系统在实现中按照逻辑和流向，将信息分为三类：一是输入信息，从用户流向渲染服务器。服务器接受用户输入，以实现系统与用户交互，如用户转动相机视角从不同的角度观察场景；二是命令和数据信息，从渲染服务器流向客户机。系统初始化时，渲染客户机对将要渲染的场景和实体等一无所知，它的所有信息全部来源于服务器，从渲染服务器到客户机有不同的命令和数据类型，系统实现中设计了三种命令：创建、更新和销毁。当渲染服务器发送一条创建命令需渲染客户机创建某实体时，则在数据包中分别填入命令类型、实体名字及属性等，渲染客户机根据事先协议顺序读取数据包内容后开始渲染；三是反馈信息，从渲染客户机流向服务器。服务器收集反馈信息，指示客户机开始渲染或翻转，以进行同步处理或差错控制。

3　实验结果及分析

实时分布式渲染系统通过设置视棱锥参数实现任意分屏，本系统在测试中使用二分屏，即将整个场景分别由两台渲染客

（下转 111 页）