关于集群渲染（cluster rendering）的研究

摘要：本次读书报告主要阅读的文献是关于集群渲染（cluster rendering）的一些文章。报告主要分为四个部分：首先简单的介绍了渲染技术及其发展，其次介绍了集群渲染的国内外研究现状，然后阐述了集群渲染的架构和调度算法，最后我进行了总结并提出了一些我自己的看法。

一、渲染技术及发展

渲染技术是一种计算机图像生成技术，它针对已构造的几何场景模型，结合几何对象表面的色彩、纹理及材料，考虑在各类可见光源环境下，根据光照模型计算生成几何表面的色彩及光亮度，并最终生成具有较高真实效果的场景图[1]。其关键点主要包括场景模型的简化和管理、光照模型、纹理映射、材质设置、渲染算法等。渲染是基于一套完整的程序计算出来的，硬件对它的影响只是一个速度问题，而不会改变渲染的结果，影响结果的是看它是基于什么程序渲染的。因此在既定的程序下，通过提高硬件的性能对渲染的速度帮助很大。近年来，游戏、动画和电影事业的蓬勃发展，对渲染速度提出了更高的要求。在较大规模的动画制作项目过程中，特别是三维动画和电影节目的制作，由于复杂的场景和特效，着色渲染需要大量的时间。渲染速度过慢成为了影视特特效和动漫制作的瓶颈。因此，不断提高渲染速度是影视和动漫等创作过程中追求的目标。根据我的理解，我将渲染技术的发展分为 4 个阶段：基于 CPU 的渲染、基于 GPU 的渲染、网络渲染以及集群渲染（cluster rendering）。

基于 CPU 渲染的渲染是最初的阶段，也是最慢的。以 4K 分辨率进行渲染为例，假设每帧渲染耗费 1 小时，1 分钟动画(60s× 24frames/s)需要 60 天来进行渲染，1 部 120 分钟的影片需要 20 年的时间来完成渲染工作[2]。在时效性上远不符合影视创作和动漫制作的要求。

GPU 自 1999 年首先由 nVidia 公司提出来后，其发展的速度是 CPU 更迭速度的 3 倍多。GPU 的运算速度比 CPU 要快，主要是由于 GPU 是为图形实时渲染而量身定制的。具有超长流水线和并行计算的优势[3]。因此，采用 GPU 方式的渲染在速率上有一定的提高。

在网络环境下，由于网络节点的独立 CPU 和 GPU 渲染能力十分有限，因此

采用网络渲染可充分发挥网络资源优势、利用渲染节点的计算能力来进一步提升渲染效率。文献[4]中利用了 Muster 渲染管理软件，使用 5 台 PC 机对 3D max 场景文件进行了网络渲染，取得了较高的渲染加速比值。

集群渲染（cluster rendering）是利用集群计算机的优势，通过网络分发进行的并行渲染，充分利用集群网络中的计算机硬件资源，将复杂的场景通过大量计算，生成预览图像或最终动画图像，以供效果调整审定或后期制作合成之用。整个过程是自动化和智能化的，它代表了制作技术的发展方向。

二、集群渲染的国内外研究现状

在国外，集群渲染在国外的影视制作中已得到了广泛的应用，如在美国电影《变形金刚》中，采用了 RenderMan 和 Mental Ray 渲染软件[6]，对创作所有机器人图像进行了集群渲染，此外，国外有一些在线的商业集群渲染平台，如 Anthony Chong 等人提出了基于网格渲染服务的框架，同时为了解决大型数据量的场景文件和图片的传输问题，设计了一个无损三维压缩算法，并申请了专利[6]。

在国内，许多集群渲染都是采用手工或简单的自动分配来完成渲染任务，对于智能化的集群渲染，国内的若干研究机构和应用单位也进行了一些尝试，但限于技术和经验尚未大规模地应用，智能化的集群渲染也成为目前解决三维制作工作中的瓶颈、改善工作流程和提高效益的首选方案。现已建立了并行集群渲染系统的典型应用有：中央电视台并行集群渲染系统、上海市多媒体公共服务平台、北京渲染平台和中国传媒大学动画学院渲染平台等[7]。目前出现的基于网络的在线集群渲染技术，主要是针对业务门户的改进，尚不能充分利用网络中所有分布节点的计算能力。

三、集群渲染的架构和调度算法

3.1 集群渲染的架构

基于集群的动漫渲染调度系统采用三层结构，如图 3.1 和图 3.2 所示，系统总体框架可以分为三层：最底层为渲染资源层，即实际执行渲染任务的服务器，支持多种硬件平台和多种操作系统，而这种异构性对用户完全透明；中间层为集群渲染系统的核心功能层，包括用户管理、作业管理、资源管理和安全管理等功能；最上层为用户与系统交互的门户层，门户层允许授权的用户并提交作业，且

可以查询作业的运行情况以及资源的使用情况。基于集群的动漫渲染调度系统的三层结构中，核心层是动漫渲染调度系统最重要的层，一方面，核心层接受并执行来自门户层的命令，另一方面，核心层在实时监控资源的使用情况的基础上，负责管理资源的分配、渲染作业的分发以及动态的负载均衡。

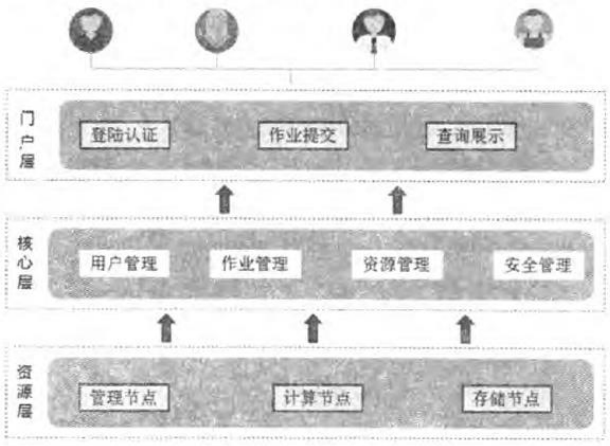

图 3.1 系统总体架构

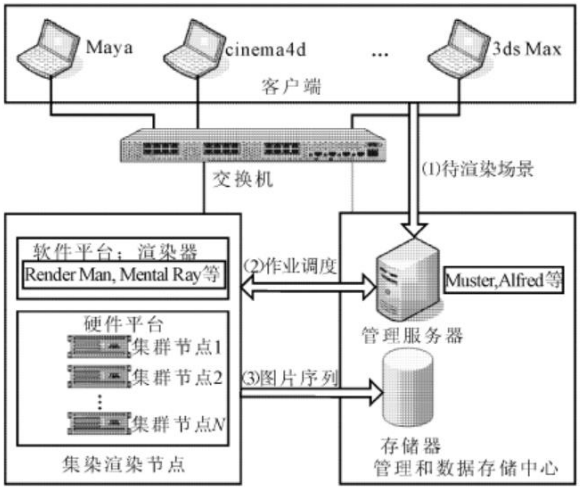

图 3.2 典型的网络集群渲染工作流程

门户层是联系用户和系统的纽带，是用户对整个系统认识的最直观的接口，为了方便用户的使用，大概有两种访问模式：Web 模式和控制台模式。Web 模式下用户可以直接通过因特网访问系统，通过简单的鼠标操作即可完成任务的提交和查询；而控制台模式则为高级用户提供，完全采用命令行方式，用户登陆以后直接运行命令，也可编写脚本来完成复杂的任务。

核心层实现了渲染调度系统的主要功能部件。这些部件主要包括用户管理、作业管理、资源管理和安全管理等。核心功能层屏蔽了下层各种异构渲染资源的异构性，为上层提供透明的信息服务、作业管理等安全的无差异服务。

资源层集群服务器为用户提供包括渲染计算、存储、网络等资源，它是进行资源管理和作业调度的主要对象。资源平台上需要安装代理模块，用于集群渲染作业管理系统控制该资源平台。

作业管理核心层是集群渲染作业管理系统的功能主体，客户端的所有请求都由作业管理核心层响应，核心层将各类请求处理后转交相应的代理模块，代理模块再调用相应目标系统的渲染服务。作业管理的主要任务是如何根据作业的计算需求和集群资源使用的情况，完成各个作业所需计算资源的选择与合理配备，并管理作业的调度和控制执行。实现集群渲染资源的整体优化是基于集群的渲染调度系统作业管理的首要目标。因此，设计一种旨在解决系统中各节点工作负荷不平衡的问题的负载均衡策略是非常重要的。

3.2 集群渲染的调度算法

集群渲染系统的负载均衡策略旨在采用各种手段解决系统中各节点工作负荷不平衡的问题，使得集群渲染系统能够充分地发挥每个工作节点的性能，实现渲染集群系统工作效率的最大化，其中的核心组件是任务分配和节点调度管理。通常，不同的任务分配及调度策略将导致系统牌不同的工作状态，进而影响系统的整体性能及强壮性，寻找适用于集群渲染系统的最优调度策略是非常有现实意义的研究工作。下文将介绍一种基于任务生成时间和任务量判别的负载均衡调度算法，它主要结合了先来先服务算法及短时间优先算法的策略思想，以有效提高渲染系统的整体工作效率，获得更好的系统稳定性。

先来先服务调度算法（FCFS，First Come First Serve）是一种常用的服务调度算法，该算法在系统进程调度和任务调度中均有广泛的应用。集群渲染系统中采用该调度算法时，当有空闲资源可用时，从等待任务队列中选择一个或者多个最先进入该队列的作业，将它们调入任务执行队列，然后分配某个工作节点执行该任务，该任务被不间断执行直至任务完成。FCFS 算法比较有利于渲染工作量较大的任务，而不利于工作量较小的任务，虽然 FCFS 算法的 CPU 利用率较高，但会增加工作量较小渲染任务的等待时间，影响任务响应质量。

短作业优先调度算法（SJF，Shortest Job First），是指对短作业优先调度的算法，它同样广泛用于系统的进程调度和任务调度。SJF 算法在执行每次渲染任务调度时，循环从等待任务队列中选择一个或者若干个估计运行时间最短的作业，将他们调入任务的执行队列进行渲染，直到任务完成。SJF 算法能有效地降低作业的平均等待时间，提高系统吞吐量，降低渲染时间较小任务的等待时间。然而

SJF 算法也存在不容忽视的缺点：（1）该算法对渲染任务量大的不利，如果有一个渲染时间较长的任务进入系统的等待队列，由于调度程序优先调度那些渲染时间较短的任务，将导致时间较长的任务长期不被调用；（2）由于作业的长短只是根据用户提供的估计执行时间而定的，致使该算法不一定能真正做到短作业优先调度。

为了更有效地提高集群渲染系统节点 CPU 的资源利用率，同时兼顾渲染任务的执行等待时间，保证任务的响应即时性，本文提出了一种基于任务生成时间和任务量的负载均衡调度算法。算法主要思想为：为渲染任务等待队列中的每个任务分配一个动态的调度优先级，当系统可执行新任务时，根据任务优先级，先执行优先级高的任务。显然，优先级的定义为上述调度算法的核心，具体定义为：

$$\mathrm{Pri} = \mathrm{r} \cdot \mathrm{f(s)} + (1 - \mathrm{r}) \cdot \mathrm{h(s)} \tag{3-1}$$

其中，f(s)为渲染任务产生时间的优先级函数，借鉴 FCFS 算法调度思想，依据渲染任务的产生时间，定义优先级函数f(s)为：

$$\mathrm{f(s)} = \frac{1}{T_s + 2} \tag{3-2}$$

T_s为等待队列中在任务 s 之前产生的未被调度执行的任务数，显然，越早产生的任务其值越大。此外，考虑T_s有可能为 0，f(s)定义的分母式作加 2 处理平滑。式（3-1）中的定义h(s)为渲染任务的计算任务量相关的优先级函数，同时结合 SJF 算法的思想，将渲染任务量小的任务优先执行。定义任务量优先级函数h(s)为：

$$h(\mathrm{s}) = \frac{1}{F_s + 2} \tag{3-3}$$

F_s为在等待队列中根据渲染任务的帧数和渲染时间，等待队列中任务量大于任务 s 的任务数量，任务量越小优先级越高，同样在$h(\mathrm{s})$中将分母作加 2 考虑。式（3-1）中 r 为权重因子，当 r=1 时，均衡调度算法等同于 FCFS 算法，当 r=0 时，均衡调度算法等同于 SJF 算法。以下是 FCFS 算法、SJF 算法和上述基于任务生成时间和任务量的负载均衡调度算法的 CPU 利用率的比较：

表 3-1 三种调度算法的 CPU 利用率（%）

	FCFS	SJF	负载均衡
第一次	87.58	66.38	77.45

第二次	56.98	50.53	80.57
第三次	69.77	73.38	71.57
第四次	91.35	68.08	59.38
第五次	65.39	69.97	60.07
第六次	91.68	68.74	73.28
第七次	70.36	62.22	73.28
第八次	61.56	72.45	77.16
第九次	61.56	45.95	89.56
第十次	52.84	69.64	61.62
平均	71.27	64.73	72.86

从表 3-1 中可以看出在 CPU 利用率方面，负载均衡调度算法的 CPU 利用率最高，FCFS 算法次之，SJF 算法效率最低。因此，负载均衡调度算法均有很好的性能表现，能更好地满足集群渲染系统的任务调度负载均衡需求。

四、总结

显然，对于现在的电影和动画制作来说，现在的渲染速度还是不够的。本文主要致力于从硬件角度加速渲染，主要研究了集群渲染（cluster rendering）。不过，我觉得集群渲染比较适合于单帧画面的渲染，而连续画面的渲染个人感觉用网络渲染更合适一些。或许可以把网络渲染和集群渲染相结合，这样任务分配时就可以先以帧为单位分配到每个小集群，小集群渲染单帧的时候再根据上述负载均衡算法进行任务调度。另外，我觉得连续画面之间存在一些联系，或许这里存在挖掘的空间，可以使得渲染的工作大量减少。

参考文献

[1] Voos R,Blended Learning:What is it and Where Might It Take Us［J］.Sloan-C View,2003(1):2-5.

[2] 戴敏利.基于 Muster 的集群渲染系统功能扩展研究与开发[D].武汉:华中师范大学,2007.

[3] Krüeger A, Kubisch C, Straub G, et al. Sinus endoscopy--application of advanced GPU volume rendering for virtual endoscopy[J]. IEEE Transactions on

Visualization & Computer Graphics, 2008, 14(6):1491-1498.

[4] Okamoto Y, Oishi T, Ikeuchi K. Image-Based Network Rendering of Large Meshes for Cloud Computing[J]. International Journal of Computer Vision, 2011, 94(1):12-22.

[5] Jiao S, Wang X, Zhou M, et al. Multiple ray cluster rendering for interactive integral imaging system.[J]. Optics Express, 2013, 21(8):10070.

[6] Chong A, Sourin A, Levinski K. Grid- based computer animation rendering[C]// Proceedings of the 4th international conference on computer graphics and interactive techniques. Australasia, Southeast Asia, 2006:39- 47.

[7] 刘伟.WEB 方式集群渲染系统的研究与实现[D].武汉:华中师范大学,2007.

[8] Nikolopoulos D S, Polychronopoulos C D. Adaptive scheduling under memory constraints on non-dedicated computational farms[M]. Elsevier Science Publishers B. V. 2003.

[9] Shoemake K. Animating rotation with quaternion curves[J]. Acm Siggraph Computer Graphics, 1985, 19(3):245-254.