

# 大规模室外动态场景调度机制研究

胡 斌<sup>1,2</sup>, 江 南<sup>1</sup>, 邹志强<sup>3</sup>, 邵 华<sup>1,2</sup>, 王 鹏<sup>4</sup>

(1 中国科学院南京地理与湖泊研究所, 南京 210008; 2 中国科学院研究生院, 北京 100039;

3 南京邮电大学, 南京 210008; 4 南京大学, 南京 210093)

摘要: 论文分析了传统的地形流式处理方法和八叉树场景组织的优缺点, 然后把松散八叉树引入动态场景的组织, 并提出了数据调度流水线的概念, 在此基础上提出了粗粒度页和细粒度片的动态调度方法。该方法是基于外存(Out-Of-Core)的实时调度方法, 用粗粒度页 Page 组织地形, 用细粒度片 Tile 渲染地形, 既避免了频繁读取外存, 又实现了对大范围的地形场景数据的精细调度; 通过把调度过程分散到流水线的各阶段和各帧, 减少了“爆发式”内存和磁盘 I/O 请求, 确保了渲染的平稳性; 利用数据调度流水线的阶段性管理 Page 和 Tile 的生命期, 避免了 Page 和 Tile 不必要的状态转移带来的开销。

关键词: 松散八叉树; 动态场景; 页; 片; 调度流水线

## 1 引言

大规模室外动态场景的实时渲染是 GIS、虚拟现实、游戏、仿真等领域的关键技术, 是近年来国内外众多学者的研究热点之一, 并取得了卓有成效的工作<sup>[1-3]</sup>。大规模室外动态场景的实时渲染有许多技术上的难点, 主要包括海量地形数据的表示和组织、动态场景的管理以及场景的真实感表现等。相关论文描述了一些大规模地形的快速渲染方法, 基本思想是将地形数据分块组织<sup>[4-6]</sup>, 对每一块数据都预先生成几种不同细节层次的网格, 存储在外部存储介质上。运行时刻选择当前可见地形块的适当的细节层次导入内存, 然后进行渲染工作。这种方法的缺陷是需要采用较多的存储空间, 在两种细节层次间切换时会产生视觉上的跳跃效果。本文在充分利用数据调度的流水线的基础上, 采用粗粒度分页和细粒度分片的动态调度机制, 结合视点的动态 LOD 算法, 实现了高质量的渲染效果和实时的渲染速度。考虑普通八叉树在动态场景的管理方面存在不足, 本文引入松散八叉树组织动态场景, 使八叉树在可见性判断、碰撞检测和场景查询等方面的优势扩展到大规模室外动态场景<sup>[7-9]</sup>。

## 2 松散八叉树及其应用分析

Thatcher Ulrich<sup>[10]</sup>提出的松散八叉树的概念主要用于空间分区。在普通的八叉树场景管理中, 整个场景被划分为一棵八叉树, 每个节点代表一个包围体。八叉树的根节点代表整个场景包围体, 包围了整个场景中的几何体。如果只是用于组织大规模静态场景, 使用普通八叉树是很合理的, 它在可见性判断和碰撞检测方面都具有很高的效率, 但是如果场景中还包含很多其他几何体, 尤其是细小的动态几何体, 普通八叉树就难以适用了。

由于八叉树只是四叉树的 3D 扩展, 因此接下来我们用四叉树的方式来说明八叉树的组织。如图 1 所示, 黑色几何体尽管很小, 但它跨在大包围体的分割面上(虚线所示), 按普通八叉树的管理方式, 这个几何体将属于大包围体。如果场景中有很多小几何体, 将形成小几何体高层节点聚集的现象, 如果小几何体是动态移动的, 这种情况会加剧。小几何体高层聚集降低了空间操作的效率, 尤其是使八叉树在可见性判断、碰撞检测和场景查询等方面的优势大打折扣。

收稿日期: 2006-06-14; 修回日期: 2007-01-18.

资助项目: 本文得到中国科学院知识创新项目(KZCX3-SW-331)资助.

作者简介: 胡斌(1975-), 湖南衡阳人, 男, 博士, 毕业于中国科学院南京地理与湖泊研究所, 主要从事虚拟现实与 3S 集成研究。发表论文 10 余篇, E-mail: hb\_hubin@126.com

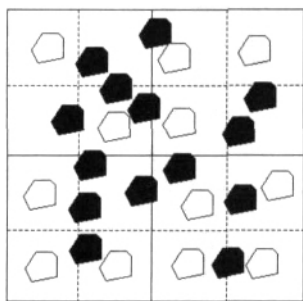


图 1 小几何体高层聚集

Fig.1 Small objects stict to higher nodes

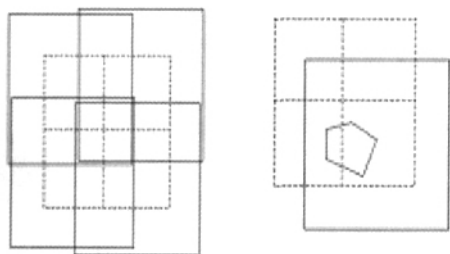


图 2 松散八叉树节点

Fig.2 Loose octree nodes

针对普通八叉树的缺点, 我们采用松散八叉树 (Loose Octree) 来管理大范围动态场景。松散八叉树的节点层次和中心不变, 但是节点包围体的边长更大, 节点相互重叠。如图 2 左图所示, 虚线表示普通八叉树节点, 实线表示松散八叉树节点(为便于区分, 对节点包围体进行了偏移)。松散八叉树节点较之普通八叉树节点有何优势呢? 如图 2 右图所示, 在普通八叉树节点中(虚线所示), 跨在节点分割线上的细小几何体吸附在高层节点中, 而在松散八叉树中(实线所示), 由于节点包围体更大, 同样的几何体属于较低层次的节点, 这样就极大地缓解了小几何体高层聚集问题。

设根节点边长  $W$ , 普通八叉树节点边长为  $L$ , 松散八叉树节点边长为  $L$ , 则  $L = L = W/2^{\text{level}}$

上式中  $\text{level}$  为节点层次, 从 0 开始。设  $R$  为  $\text{level}$  层能容纳的物体最大半径(不论其位置如何), 则

$$R = (L - L) / 2 = (-1) W/2^{\text{level}+1}$$

设  $\text{level}$  层为第一个能容纳半径为  $r$  的几何体的层次, 则

$$r = R = (-1) W/2^{\text{level}+1}$$

$$\text{level} = \log_2((-1)W/r) - 1$$

取  $\text{level} = \text{cell}(\log_2((-1)W/r) - 1)$

一般来说, 取 2 是较好的选择, 一是便于计算, 二是避免同一层中几何体属于多个节点, 上述公式可以简化为  $\text{level} = \text{cell}(\log_2(W/r) - 1)$

设世界坐标原点在根节点中心, 当计算出几何体节点层次  $\text{level}$  后, 可由几何体中心坐标直接求出节点索引

$$\begin{aligned} \text{index}(x,y,z) &= \text{floor}((\text{Geometry}(x,y,z) + W(x,y,z)/2)/L) \\ &= \text{floor}((\text{Geometry}(x,y,z) + W(x,y,z)/2)/(W/2^{\text{level}})) \end{aligned}$$

上式中,  $\text{Geometry}(x,y,z)$  表示几何体中心,  $W(x,y,z)$  表示世界尺寸。

一般来说, 采用松散八叉树组织场景会产生更多的结点, 但由于大多数算法考查结点比考查几何体的计算复杂性要简单得多, 因此松散八叉树能产生更好的效率。以可见性判断为例, 判断一个结点的可见性只需要检测结点包围体的几个顶点, 精确判断几何体的可见性需要检测几何体的所有顶点(最坏情况下), 而几何体一般都是由成千上万的三角形形成, 因此几何体比结点可见性判断要复杂得多。通过适当增加的结点可见性判断从而减少几何体可见性判断的数量, 松散八叉树产生更好的渲染效率。

因此, 对于静态地形场景, 采用八叉树场景组织是合理的方式, 但是对于存在很多动态小几何体的大规模室外场景, 存在小几何体高层结点聚集的现象, 选用松散八叉树是较合适的。

### 3 基于粗粒度页和细粒度片的动态调度机制

大规模室外场景管理最需要解决的是地形的调度。因为地形数据的规模都比较大, 运行时刻不可能将所有数据都导入到内存中, 一般的做法是对地形进行分块。

但是传统的地形分块机制有一些不足之处。

首先, 不能进行精细的剪裁。尽管很多时候视锥体只是覆盖可见区块的一小部分, 但是整个区块的数据仍然需要全部加载, 而不能进行精细的剪裁, 只加载这些区块的部分数据, 如图 3a 所示。

其次, 分块粒度难以正确确定。如果分块粒度太大, 更新潜在可见区时从外存加载数据的时间较长, 如果分块粒度太小, 随着视线的移动, 会频繁更

新潜在可见区,导致频繁从外存加载数据。这两种情况都对帧速率造成冲击,引起帧停顿。

再次,没有充分考虑和利用数据调度的流水线过程。地形分块机制在更新潜在可见区块时往往简单地一次性加载包括纹理数据的所有地形数据,引起“爆发式”内存和磁盘 I/O 请求,从而加剧了帧速率的“颠簸”。

最后,地形分块调度机制要求潜在可见区的地块,无论将来是否可见,都已经加载了所有数据。显然,对于将来不可见的数据块,预先加载所有数据,既浪费了有限的内存,又降低了渲染性能。

依据对传统地形分块调度机制的分析,本文提出了数据调度流水线的概念,并在此基础上对传统的分块的流式处理方法进行改进,提出了一种粗粒度分页和细粒度分片相结合的流式处理方法。该方法和松散八叉树场景组织相结合,既解决了分块粒度问题,又能进行更精细的剪裁(如图 3b),同时对数据的调度过程进行细粒度划分,充分利用数据调度的流水线结构,确保了渲染 FPS 的平稳性。

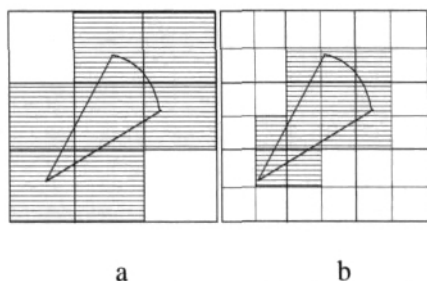


图 3 基于粗粒度分页和细粒度分片的调度机制  
能进行更精细的剪裁

Fig.3 Dynamic dispatching based on coarse-granular pages  
and fine-granular tiles realizes delicate cutting

为了对地形数据调度进行更好的组织,本文提出了一种数据调度流水线的概念。该数据调度流水线对地形的调度过程进行了细分,把地形数据从开始加载到显示在屏幕的过程进行阶段划分,以便于场景系统的剪裁,减少系统负荷。如图 4 所示,在送入最终的渲染管线前,地形数据需要依次经历以下阶段:加载格网数据、加载纹理数据、生成顶点数据流、基于视点形成 LOD、进入渲染管线。

从上述数据调度流水线可以看出,只有实际需要渲染到屏幕的才会经历上述所有阶段,如果场景剪裁系统对地形进行了剪裁,被剪裁掉的地形不会经历上述所有阶段,剪裁可以发生在上述各阶段,

所以如果在某一阶段地形被剪裁掉,后续的各阶段就不会发生,从而避免了多余的数据状态转换过程,优化了场景渲染。

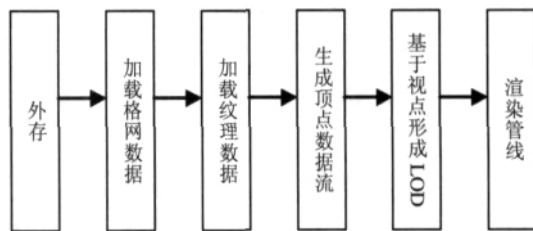


图 4 数据调度流水线

Fig.4 Dispatching pipeline

为了充分利用数据调度的流水线结构,我们提出页和片相结合的动态调度机制(以下用 Page 表示页,用片表示 Tile)。Page 类似地形分块机制中的地形块,预先分割好,但粒度更大,Tile 在运行时通过对可见区中的 Page 进行划分而动态产生,粒度较小,两者都从各自对应的内存池分配内存,用后归还池,确保了内存的高效使用。Page 是磁盘 I/O 单位,同时也是场景八叉树组织单位,Tile 是渲染单位,它的数据只需从 Page 中获取。Page 有多个状态:未加载、格网化、纹理化和就绪。Tile 也有多个状态:未加载、已加载、LOD 化。数据调度流水线各阶段和 Page、Tile 各状态之间的关系如图 5,其中斜框对应 Page 和 Tile 各状态,长方形框对应流水线各阶段。

我们利用四个 Page 队列对 Page 进行调度:Page 格网加载队列、Page 纹理加载队列、Page 就绪队列和 Page 卸载队列。Page 格网加载队列保存未加载格网状态的 Page,等待加载地形格网数据,Page 纹理队列保存格网化状态的 Page,等待加载纹理数据,Page 就绪队列保存就绪状态的 Page,Page 卸载队列保存需要回收到内存池的 Page。

利用三个 Tile 队列对 Tile 进行调度:Tile 加载队列、Tile 渲染队列和 Tile 卸载队列。Tile 加载队列保存未加载状态的 Tile,队列中的 Tile 需要从 Page 中加载数据并生成顶点数据流(包括顶点位置、法线和纹理坐标)的 Tile,而渲染队列保存已加载状态的 Tile,需要 LOD 化,Tile 卸载队列保存需要回收到内存池的 Tile。

在初始化阶段,整个场景组织成松散八叉树结构,松散八叉树的每个叶结点对应一个 Page,此时



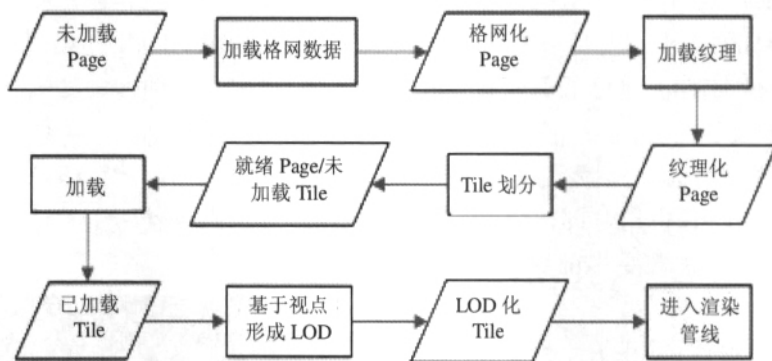


图 5 数据调度流水线和 Page、Tile 各状态间的关系图

Fig.5 Relation between dispatching pipeline and Page and Tile

的 Page 处于未加载状态,并未从外加载数据。

随着视点的移动,重新计算可见区和潜在可见区,根据新的可见区和潜在可见区中 Page 的状态更新各 Page 队列。

当潜在可见区中的 Page (此时的 Page 处于就绪状态) 进入可见区时,对其划分 Tile,形成多个 Tile。此时的 Tile 处于未加载状态,只是建立了 Tile 间以及相邻 Page 包含的 Tile 间的邻接关系,并未从 Page 中获取数据。由于 Page 是粗粒度的,Page 所包含的 Tile 不一定都处于可见区,因此需要根据视点和视野计算落入视锥体内的 Tile,并据此对 Tile 进行视锥体剪裁。

对通过视锥体剪裁的 Tile 再进行遮挡剔除测试,只有通过遮挡剔除测试的 Tile 才是可见的,才需要进行状态转换。

采用自底向上生成 LOD 的算法,在 Tile 加载阶段生成最高分辨率的顶点数据流,然后基于 LOD 准则形成其他 LOD 级别。当 Page 或 Tile 长期未使用(一般是长期不可见)就放入卸载队列等待回收。算法流程如下:

(1) 基于当前视点计算可见区和潜在可见区范围。

(2) 扫描可见区和潜在可见区,根据 Page 状态放入相应队列。

(3) 扫描所有 Tile,根据其可见性放入相应队列。

(4) 按就绪队列>纹理队列>格网队列的优先级获取队列中离视点最近的 Page,更新 Page 状态和相关 Page 队列。

(5) 对通过视锥体测试的 Tile 进行遮挡测试,并更新相应 Tile 队列。

(6) 从加载队列中获取 Tile,生成 Tile 顶点数据流、LOD 评价因子数组。

(7) 从渲染队列获取 Tile,基于 LOD 评价因子生成 LOD 索引数据流,把 Tile 的顶点数据流和 LOD 索引数据流送入渲染管线进行渲染。

(8) 按最近最小使用原则回收未使用的 Page 或 Tile。

算法根据数据调度流水线中 Page 和 Tile 的不同状态把它们放入不同队列,使 Page 和 Tile 的加载过程在各帧尽可能平均分布,避免了单帧加载过多的 Page 或 Tile 导致帧停顿。算法充分利用了帧间相关性,数据调度过程中只有少数页面需要读入。

对于地形来说,虽然使用四叉树已经足以实现地形的组织,但是考虑到在实际的室外动态场景中除了地形,往往还会有其他静态或动态的物体,所以最终采用松散八叉树组织,并结合片页机制调度动态场景。与普通八叉树一样,松散八叉树能进行高效的场景剪裁。利用松散八叉树,能在很短的时间内基于视点计算出可见区(可以把视锥体投影到二维平面简化计算),进而建立潜在可见区。在确定可见区后,对可见区中的 Page 进行细粒度 Tile 划分,对不完全可见的 Page 中的 Tile 进一步计算可见性,从而获得更精细的剪裁效果。因此基于粗粒度页和细粒度片的动态调度机制和八叉树场景管理相结合,极大地提高了整体效率。

粗粒度 Page 和细粒度 Tile 的动态调度机制有如下优点:

(1) 结合数据调度流水线,Page 和 Tile 被细分为许多状态,各状态在流水线各阶段平稳转换,确保了渲染的平稳性。

(2)可以进行精细的视锥体剪裁和遮挡剔除。对于部分可见的 Page, 形成的 Tile 中包括可见和不可见两种, 由于只渲染可见的 Tile, 避免了不可见 Tile 多余的状态转换和加载过程。

(3)Page 中只保存原始数据流, 不需要转换到渲染数据流, 减少了内存和 CPU 占用; 可以对 Page 进行粗粒度划分, 一次从外存加载更多的数据, 避免频繁读取外存。

(4)充分利用帧间相关性, 采用流水线结构, 可以把 Page 和 Tile 的加载过程分散到各帧, 减少了“爆发式”内存和磁盘 I/O 请求引起的帧停顿, 因此适合于基于外存的大规模室外场景实时渲染。

(5) 当按最近最小使用原则回收 Page 和 Tile 时, 它们是渐进式加载, 避免了多余的加载过程和状态转换, 节省了内存和 CPU 开销。

## 4 应用及结论

本文的成果已应用在“气象信息三维模拟系统”中。该研究中选取整个台湾地形作为场景, 首先从等高线图提取规则格网, 得到纵横向点数为  $14264 \times 9812$  的 DEM, 利用三个波段组合的真彩遥感图像作为纹理图 (纵横向点数同样为  $14264 \times 9812$ ), 然后对 DEM 和纹理图像分割, 共形成  $28 \times 20$  块  $513 \times 513$  的 DEM (因为 DEM 要接边, 所以边大小为 2 的幂加 1) 和同样数量的  $512 \times 512$  的纹理图像 (不足一块的 DEM 或纹理块要增补空白的行和列), Tile 的规格为  $65 \times 65$ 。整个场景中除了大地形, 还要模拟飞行器及各种气象条件, 包括云、雾、雨、雷暴和闪电等各种动态情景。我们选取多个特征点进行 Hermite 插值形成固定飞行路径, 以  $1\text{km/s}$  的飞行速度飞行, 飞行高度为  $10\,000\text{m}$ , 在无雨区帧速率稳定在  $60\text{FPS}$  左右, 在雨区帧速率基本稳定在  $40\text{FPS}$ , 急转弯对帧速率影响不大, 在只显示地形并降低飞行速度和高度的情况下, 帧速率在  $200\text{FPS}$  以上。以下是部分效果图, Page 边界间接缝效果良好, 画面平滑流畅。运行的硬件环境: P4 2.8C CPU, ATI X700 显卡, DDR 1G 内存, 软件环境: WindowsXP, Direct3D9。其效果参见图 6, 7。

本文引入松散八叉树对大规模室外动态场景进行组织, 使普通八叉树在可见性判断、碰撞检测和场景查询等方面的优势应用到大规模动态场景。

对传统地形分块机制进行了优化, 提出了数据调度流水线的概念, 并在此基础提出了基于粗粒度分页和细粒度分片的动态调度机制。采用这一机制, 可以对大规模室外动态场景数据进行精细调度, 把调度的过程分散到流水线的各阶段和各帧中, 实现大规模室外动态场景的实时渲染, 同时确保好的渲染质量。

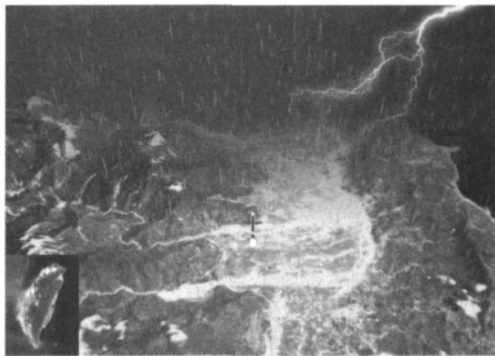


图 6 雷暴区飞行效果

Fig.6 Snapshot in thunderstorm area



图 7 山区飞行效果

Fig.7 Snapshot in mountainous area

## 参考文献

- [1] 赵友兵, 石教英, 周 骥 等. 一种大规模地形的快速漫游算法. 计算机辅助设计与图形学学报, 2002, 14(7): 624~628.
- [2] Pajarola R B. Large scale terrain visualization using the restricted quadtree triangulation. IEEE Visualization '98. North Carolina: IEEE, 1998, 19~26.
- [3] Lindstrom P, Pascucci V. Visualization of large terrains made easy. IEEE Visualization 2001. San Diego, California: IEEE, 2001, 363~370.
- [4] 吴亚东, 刘玉树. 基于连续细节层次的地形动态生成技术. 北京理工大学学报, 2000, 20(5): 602~606.
- [5] Thatcher Ulrich. Rendering Massive Terrains using Chunked Level of Detail Control [EB/OL]. <http://cvs>.

sourceforge.net/viewcvs.py/\* checkout\*/tu - testbed/tu -  
testbed/docs/sig-notes.pdf?rev=HEAD

- [6] Victor L. Spears III terrain level of detail in first person-ground perspective simulations. Master Thesis, Naval Post-graduate School Monterey, California, March 2002.
- [7] Gargantini I. Linear octrees for fast processing of three-dimensional objects. Computer Graphics and Image Process-

ing, 1982, (20):365~374.

- [8] 罗振东, 廖光裕. 计算机图形学原理和方法. 复旦大学出版社, 1993, 179~183.
- [9] 傅由甲. 基于动态八叉树的复杂场景交互式实时漫游. 武汉理工大学学报, 2005, 29(2):309~311.
- [10] Thatcher Ulrich. Spatial partitioning schemes. Game Programming Gems, ISBN: 1584500492

## Research on Dispatch of Dynamic Scene in Large Scope Outdoor Environment

HU Bin, JIANG Nan, ZOU Zhiqiang, SHAO Hua, WANG Peng

( 1 Nanjing Institute of Geography & Limnology, CAS, Nanjing 210008, China;

2 Graduate School of the Chinese Academy of Sciences, Beijing 100039, China;

3 Nanjing University of Posts & Telecommunications, Nanjing 210003, China;

4 Nanjing University, Nanjing 210093, China)

**Abstract:** This paper analyzes traditional streaming-based method and discusses the advantages and disadvantages of octree scene organization. And then the paper introduces loose octree into dynamic scene organization and puts forward the concept of data dispatching pipeline. In addition, according to this concept, the corresponding method of dynamic dispatching based on coarse-granular pages and fine-granular tiles is put forward. It is a real-time dispatching method based on out-of-core terrain which uses coarse-granular pages to organize terrain and fine-granular tiles to embroider terrain, so it avoids reading from out-of-core terrain frequently and realizes delicate dispatch for the data of large scope terrain scene. There are some advantages of our method as following: by analyzing the dispatching process into each stage and frame of pipeline, it decreases the "explosive mode" memory and I/O request and ensuring stationarity; by making use of stages of data dispatching pipeline to manage the life cycle of page and tile, it avoids unnecessary costs of state transferring of page and tile; and by using memory pool to manage the allocation and recycle of memory, it has higher utilization efficiency to memory.

**Key word:** loose octree; dynamic scene; page; tile; dispatching pipeline