计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(12):200−206 [doi: 10.15888/j.cnki.csa.006113]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 LOD 的海量地形数据并行渲染技术①

王青云[1,2], 罗　泽[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 计算机网络信息中心, 北京 100190)

摘　要: 随着地球空间信息技术的发展, 建立具有海量空间数据的大规模虚拟地形场景越来越重要. 然而, 面对海量的地形数据, 如何简化地形, 提升绘制与渲染效率, 是地形渲染的关键. 本文对 LOD 地形渲染技术、大规模数据集的分析与处理、并行计算等相关技术进行了研究, 提出了基于 LOD 的海量地形数据并行渲染技术. 该技术首先使用 LOD 四叉树简化地形, 其次结合多核 CPU 并行计算的方法提升效率, 最后结合大规模数据调度策略, 实现了海量地形数据的并行渲染, 并分析对比了非并行和并行情况下的实验结果. 本文所取得的理论与技术方面的成果可为大规模场景渲染提供新的技术思路.

关键词: LOD 技术; 并行计算; 海量数据; 地形渲染

引用格式: 王青云,罗泽.基于 LOD 的海量地形数据并行渲染技术.计算机系统应用,2017,26(12):200–206. http://www.c-s-a.org.cn/1003-3254/6113.html

Parallel Rendering of Massive Terrain Data Based on LOD

WANG Qing-Yun[1,2], LUO Ze[2]

[1](University of Cinese Academy of Sciences, Beijing 100049, China)
[2](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: With the development of geo spatial information technology, it is more important to build large scale virtual terrain scene with massive spatial data. However, in the face of massive terrain data, how to simplify terrain, improve rendering and rendering efficiency, is the key to the terrain rendering. After the research of the terrain rendering technology of LOD, the analysis and processing of large scale data sets, parallel computing and other related technologies, the parallel rendering technology of massive terrain data based on LOD is proposed. At first, quadtree LOD is used to simplified terrain, secondly it is combined with multi-core CPU parallel computing method to enhance efficiency, then it is combined with the large data scheduling strategy, finally it realizes the parallel rendering of massive terrain data, and it analyzes the non-parallel and parallel experiments under the same conditions. The theoretical and technical achievements in the research can provide a new idea for large scale scene rendering.

Key words: LOD technology; parallel computing; massive data; terrain render

1　引言

1.1　研究背景与意义

随着地球空间信息技术的发展, 建立具有海量空间数据的大规模虚拟地形场景越来越重要. 地形渲染是场景渲染的核心部分, 一直是图形学领域的热点问题之一. 它是视频游戏、地理信息系统 (GIS)、虚拟现实 (VR) 系统中重要的组成部分[1]. 随着遥感技术的发展, 大范围高分辨率的遥感数据已经可以被获取, 包含上千万多边形的场景变得越来越常见, 远远超过图形硬件的绘制能力. 如此大容量的数据要在应用中发挥

① 收稿时间: 2017-03-15; 修改时间: 2017-03-31; 采用时间: 2017-04-10

实际作用, 必须要有高效、快速的大规模地形渲染系统的支持. 因此, 如何简化地形, 提升地形绘制与渲染效率, 是实现一个大规模的地形渲染系统的关键.

提升地形绘制与渲染效率, 主要从两个方面着手: 一方面是合理地组织地形数据, 采用合理的数据模型和数据调度策略, 在不影响视觉效果的前提下, 减少需要渲染的三角形的数据量; 另外一方面是采用加速渲染算法, 提高单位时间地形数据的处理量[2].

1.2 国内外研究现状

减少需要渲染的地形数据方面, 主要使用 LOD (Level of detail) 多层次细节技术实现, LOD 技术是目前大规模的地形场景渲染的研究重点之一. 地形 LOD 技术是在不影响画面视觉效果的前提条件下, 根据地形的不同复杂程度和人眼观察地形的特点, 对地形的不同区域采取不同细节的描述和绘制. 通过逐次简化景物的表面细节来减少场景的几何复杂性, 从而提高绘制算法的效率. LOD 方法能够灵活地调度资源, 处理数据, 既减少了运算量, 又不会降低图像的显示效果[2-4].

目前 LOD 算法的主要实现方式有:静态层次细节 LOD 技术和动态层次细节 LOD 技术. 静态 LOD 技术是在进行地形渲染前, 采用 Top-down 方式对地形进行化简, 从最高精度开始, 从地形数据中逐层地去掉地形的顶点数据, 每层数据对应一个分辨率. 动态 LOD 技术是用连续变化的分辨率根据当前视点位置的实时动态来绘制层次细节模型. 如表 1 是动态 LOD 与静态 LOD 的区别.

表 1 动态 LOD 与静态 LOD 的区别

	动态LOD	静态LOD
占用CPU时间	在运行时候动态生成不同分辨率的地形模型, 大量占用CPU时间	在预处理阶段生成不同层次的模型, 渲染时不需要实时生成地形模型, 选择合适的分辨率即可, 节省了计算资源
占用存储空间	运行时动态生成, 不占用多余的存储空间	在预处理阶段, 生成的每一个不同细节层次的地形模型都需要存储, 占用大量空间
显示质量	模型分辨率改变时, 视觉变化微小, 自然过渡	实时显示时, 不同分辨率的地形模型之间切换可能会有明显的抖动感
灵活性	根据视点变换灵活切换视图模型分辨率	可选的分辨率有限, 不太灵活

动态 LOD 技术的优点是更符合人眼观察的特点, 保证了地形渲染的连续性和一致性, 同时也有效减少了地形数据的冗余. 缺点是在渲染过程中需要实时计算、绘制不同分辨率的地形模型, 过程相对复杂, 会占用一定的计算资源. 本文将主要研究基于四叉树的动态 LOD 地形算法与其并行化实现.

加速渲染算法方面, 主要是通过软硬件加速算法实现的, 如: 数据存储访问优化技术、GPU 加速绘制技术及并行渲染技术等, 其中并行渲染技术是软件加速方法的重要组成部分之一.

随着并行计算的发展以及多核 CPU 的出现, 并行化成为了提高算法效率的重要手段之一. 在计算机图形学中, 根据不同的分类标准, 并行渲染也有不同的分类体系. 根据数据调度和功能实现的方式分类, 并行渲染算法可以分为数据并行算法和功能并行算法两种. 根据图元归属判断发生的方式和时机, Molnar 等于 1994 年将并行图形渲染系统划分为 Sort-first、Sort-middle 和 Sort-last 三种体系结构. 根据实现平台分类, 可分为高性能计算机、计算机集群和多核微机三种[3].

微机是个人用户使用最多的三维浏览客户端, 随着个人用户对三维图形逼真度要求的提高和微机硬件的升级, 微机平台上的并行渲染将成为研究的热点之一. 目前基于多核微机平台的并行渲染研究还处于起步阶段, 尚未出现令人满意的并行渲染系统.

1.3 本文的主要内容

目前, 对单机渲染系统采用流水线并行技术的应用较少, 少部分实现了部分并行化效果. 三维地形渲染方面对于多核 CPU 并行计算挖掘不足, 尚未能做到真正意义上的并行渲染[4,5]. 因此, 基于多核 CPU 的并行计算对于提高算法效率具有非常重要的意义.

本文针对这一现状, 提出了一套完善的并行化处理算法, 较为有效地实现了在个人计算机上的地形渲染并行化. 本文在研究 DEM 数据组织及数据调度的基础上, 实现基于缓存区间的大规模数据调度算法; 在地形绘制阶段, 本文提出四叉树 LOD 算法, 并基于多核 CPU 产生多线程并行处理地形数据, 实现了地形 LOD 并行绘制算法, 并测试性能. 本文所取得的理论与技术方面的成果可为大规模场景渲染提供新的思路.

本文的主要内容如下: 第一节主要是引言部分, 介绍了大规模 LOD 地形渲染技术的研究背景意义与国

内外研究现状; 第二节主要是基于四叉树 LOD 技术的并行化算法; 第三节主要是大规模 DEM 数据调度算法, 提出了基于缓存区的数据调度及数据实时更新机制; 第四节主要是本文的实验设计、性能分析等. 最后一节内容主要是结论与展望.

2 LOD 地形算法的并行化

2.1 四叉树 LOD 算法

基于四叉树的 LOD 方法是采用四叉树结构存储 DEM 数据, 在进行地形渲染时, 首先自顶向下遍历四叉树, 再实时计算节点可见性, 即判断节点是否需要四叉分割.

在四叉树结构表示地形模型的过程中, 每一个地形块都可以由四叉树中的节点表示, 每个节点都对应一个地形区域. 四叉树的逻辑结构如图 1 所示. 当一个节点被判断为可见时, 会被细分为 8 个三角形网格, 并继续往下一层判断其子节点的可见性. 当一个节点不可见时, 该节点细分为两个三角形网格, 不再继续细分[6,7].

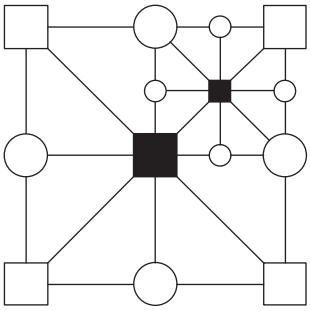

图 1 四叉树的逻辑结构

在生成四叉树, 需要不断地将节点细分, 那么如何决定一个节点是否需要分割就是一个关键的问题. 节点评价标准决定了在某一时刻, 该节点是否可见, 也就是该节点是否需要细分. 这种标准通常有两个决定性的因素. 其中之一是视点, 离视点较近的部分分辨率较高, 细节较多, 反之, 离视点较远细节较少. 另外一个影响因素是地形本身的特征. 比如起伏不大的地表面分辨率较低, 很少的细节就能很好地表现出来, 而不平整的地表面则需要更多的细节来表现它的地理特征[8-10].

综合考虑视距和地形特征, 设观察者所处位置为 $V(V_x,V_y,V_z)$, 地形顶点坐标 $P(P_x,P_y,P_z)$. 则按照如下公式决定某个节点是否被激活:

$$L_1 = max(|P_x - V_x|, |P_y - V_y|, |P_z - V_z|) \quad (1)$$

$$Enabled = error * \text{Threshold} < L_1 \quad (2)$$

其中 L_1 为视距的近似值, *error* 为节点的顶点误差, Threshold 为细节阈值常量, 细节阈值常量是人为设置的一个常量参数, 它是 LOD 模型的分辨率的控制参数. Threshold 越大, 模型分辨率越高, 反之分辨率越低. Enabled 为节点激活标识, 当 Enabled 为 1 时, 表示该节点需要激活. 当 Enabled 不为 1 时, 则屏蔽该点. 对四叉树的每一个节点进行激活标志计算, 得出节点可见性, 这个过程称为顶点测试.

为了减小计算量, 在实时计算节点可见性的过程中, 一般不会对每个节点进行顶点测试, 而是对地形块先进行盒测试. 类似于顶点测试, 以地形块为单位判断该块的可见性, 再决定是否需要分割.

2.2 生产者消费者模型

在进行四叉树裁剪的过程中, 系统一方面要根据用户控制下的视点, 不断的遍历基于当前视点的四叉树; 另一方面, 要对四叉树的每一个节点进行节点激活判断, 在本文中, 四叉树的深度最深可达到 19 层, 总节点数目多达 $(2^{19}+1)*(2^{19}+1)$, 系统需要实时遍历节点, 并计算每个节点的可见性, 对系统的计算性能有很高的要求, 因此, 本文采用多线程并行化方法, 以充分利用系统的计算资源.

本系统采用生产者消费者模型, 生产者消费者模型是一个并行化处理中常用的模型, 如图 2. 它拥有一个队列容器, 并借此来解决生产者和消费者的单线程转换为多线程的并行化问题. 本文使用一个线程 (生产者) 用于遍历四叉树节点, 和多个线程 (消费者) 用于计算四叉树中的节点可见性.

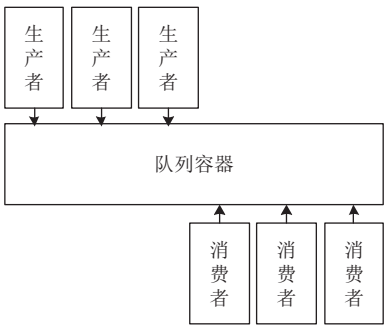

图 2 生产者消费者模型

生产者的具体职责是, 将一份大量数据进行拆分,

并产生多份待处理的数据. 而消费者的功能是逐份处理产生的多份待处理数据. 二者之间不直接传递数据, 而是通过一个数据队列来缓存待处理的数据. 所以, 当生产者完成待处理数据的生产之后, 不用等待消费者处理, 而是直接将待处理数据放入缓存队列, 然后继续生产下一份待处理数据. 消费者无需直接接触海量数据, 也不需要与生产者交换数据, 而是从缓存队列里直接取出待处理数据进行计算.

2.3 并行化算法

为了实现四叉树 LOD 算法的并行化, 可以建立一个节点队列, 用于储存四叉树节点的指针. 然后使用生产者线程, 通过广度遍历算法不断实时生成节点队列. 再使用多个消费者线程, 在满足节点父子顺序的条件下, 实时处理队列中的节点 (即判断点的可见性), 即可实现四叉树节点可见性更新的并行化.

生产者线程的主要任务是根据随着视点变换, 从四叉树根节点开始, 通过广度遍历算法不断实时生成节点队列. 本文中生产者通过一个先进先出队列存储四叉树的节点队列. 获取每个节点的激活标志, 如果节点被激活, 则该节点进行四叉分割, 将四个子节点加入节点队列; 反之, 将该节点从节点队列弹出. 重复以上操作, 直到节点队列为空为止.

消费者线程的主要任务是根据生产者生成的节点队列, 计算队列中每个节点的可见性. 本文采用多个消费者, 通过一个先进先出队列, 存储节点队列中每一个点的可见性, 更新节点的激活标志. 直到节点队列为空.

使用生产者-消费者模型进行 LOD 算法的并行化的关键在于如何解决多线程遍历多叉树的数据同步问题. 本文采用 mutex 互斥量来维持生产者与消费者线程之间的同步机制. 互斥量是一种表现互斥现象的数据结构, 来保证共享数据操作的完整性. 本算法使用了两类互斥锁: 四叉树节点队列锁 mutex1 和四叉树中的节点锁 mutex2, 分别用于来保证四叉树节点队列和四叉树节点的数据操作完整性.

本系统中, 结合生产者消费者模型的多线程四叉树 LOD 并行算法流程图如图 3 所示. 其流程如下:

1) 每当视点变换时, 缓存数据更新, 四叉树根节点对应的地形数据随之更新, 四叉树也随之更新.

2) 并行四叉树 LOD 算法, 激活多个线程 (包含生产者、消费者).

3) 从根节点开始, 首先生产者线程将根节点加入节点队列, 消费者从队列中取出根节点, 更新激活标志为可见.

4) 生产者判断当前节点队列是否为空, 读取当前节点队列中最后一个节点的可见性, 如果为可见, 产生四个子节点, 并加入节点队列. 如果不可见, 将该节点弹出队列.

5) 消费者判断当前节点队列是否为空, 从节点队列中取出节点, 并计算节点可见性, 更新节点激活标志.

重复 4)、5) 直到节点队列为空. 更新视图模型并进行绘制.

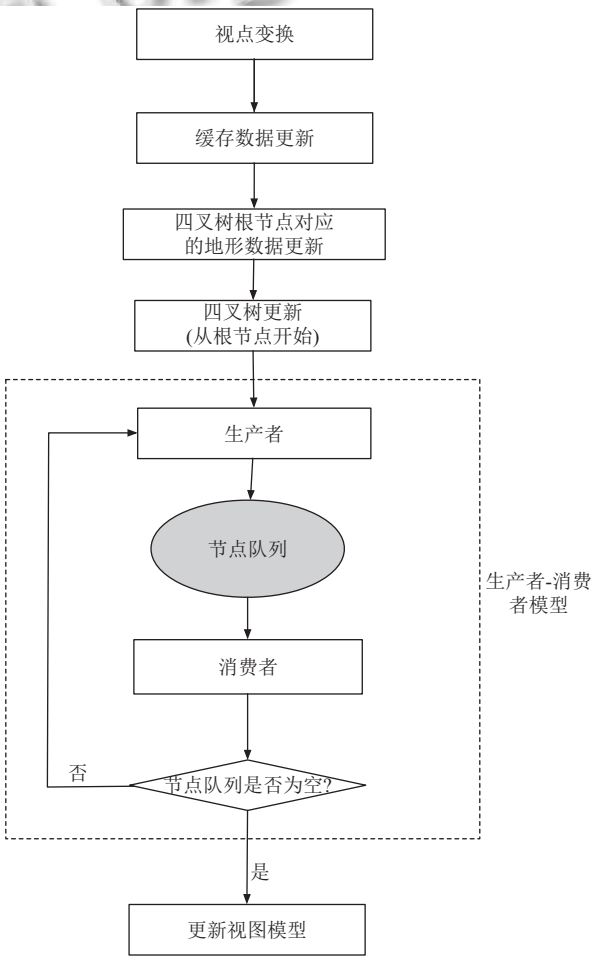

图 3 LOD 并行化算法流程图

3 大规模 DEM 数据调度

3.1 GLSDEM 数据集

全球地形数字高程模型 (GLSDEM) 是 NASA 和 USGS 组织公开的全球地形地貌空间数据, 由多个数据集的综合处理而成, 它实现了覆盖全球大部分地区的

地形记录. GLSDEM 数据集的地形分辨率为 3 弧秒 (90 m), 地理坐标基于 WGS-84 标准. GLSDEM 数据集以 GeoTIFF 格式储存. GLSDEM 数据集包含了 19234 个 GeoTIFF 文件. 每一个 GeoTIFF 文件大小为 2.75 MB, 总数据量为 51.8 GB.

文件读取使用 GDAL 开源 C++代码库后, 读取文件后可以得到每个 GeoTIFF 文件的地形分辨率, 1201*1201, 其覆盖范围为球面上边长为 1 弧度的正方形. 根据 GLSDEM 数据集的数据结构可将三维空间坐标转换为全球坐标.

3.2 大规模数据集调度算法

本文使用的地形数据集数据量巨大 (大小为 51.8 G), 采用了数据分块的组织方式, 以 GeoTIFF 文件为单位, 采用数组的数据结构存储地形数据. 由于内存限制, 普通电脑无法一次性将所有数据加载至内存供 CPU 使用. 此外, 一次性加载所有地形数据, 由于硬盘传输速度慢, 会导致程序读取全部地形数据时间过长, 启动速度极慢. 因此, 为了实现实时读取硬盘中大规模地形数据以及大量地形数据向三维空间的映射, 本文设计了内存缓冲区和实时数据更新机制.

3.2.1 内存缓冲区

内存缓冲区用于缓存硬盘中的地形数据. 由于视点变换的需要, 渲染完一帧以后需要重新加载新的一帧所需要的地形数据, 新的地形数据便被读取并加载至内存. 在渲染过程中, 首先将地形数据加载到缓存区, 一旦视点发生变化, 内存数据需要更新, 都是将缓存区作为数据源, 而不是直接从外存读取数据.

本文设置了 300 个文件单位的缓存区, 约 825 mb 的缓存容量. 当视野发生变化时, 载入视野范围内的数据, 删除两倍视野范围以外的数据. 即保留了一倍视野范围的缓冲. 本文的实现方式是首先在内存中分配一块固定大小区域作为缓存区, 然后通过 GDAL 库的读取波段数据接口函数 RasterIO() 从外存中读取数据并存入缓存.

3.2.2 数据更新机制

在数据更新方面, 本文采用的是根据视点变换, 实时动态渲染地形数据块. 视点的变换会导致视景体和缓存区的变化, 使得内存数据和缓存数据发生变化.

本文将视点的变换归纳为旋转、平移、缩放这三类. 本文通过调用 OpenGL 的视图矩阵类 ViewMatrix 类, 它会根据视点的位置、朝向等变化, 实时更新视图模型. 当鼠标或者键盘移动, 视点发生变化, 首先调用 Glut 库的消息处理函数, 监听窗口大小改变、键盘和鼠标事件. 将这些用户事件对应到不同类型的视点变换操作, 接着根据视点变化类型, 调用前面提到的的三种不同视点变换函数. 最后, 根据视点位置的移动距离, 方向, 视角大小等计算出需要更新的视野范围, 实时更新模型视图矩阵, 然后调用 glutSwapBuffers() 方法将缓存数据加载到内存并进行渲染输出.

3.2.3 数据调度算法

本文采用数组的数据结构存储地形数据, 并通过固定长度为 300 的一个先进先出队列 history, 控制缓存区的大小. data 是所有的缓存文件数据, data[*]是一个文件数据, data[*][*]是一个 DEM 文件中的一个点.

视点的运动会导致缓存数据更新、视图模型的更新. 需要采用一定的调度算法来保证内存数据和缓存数据的稳定. 本文采用如图 4 所述调度算法.

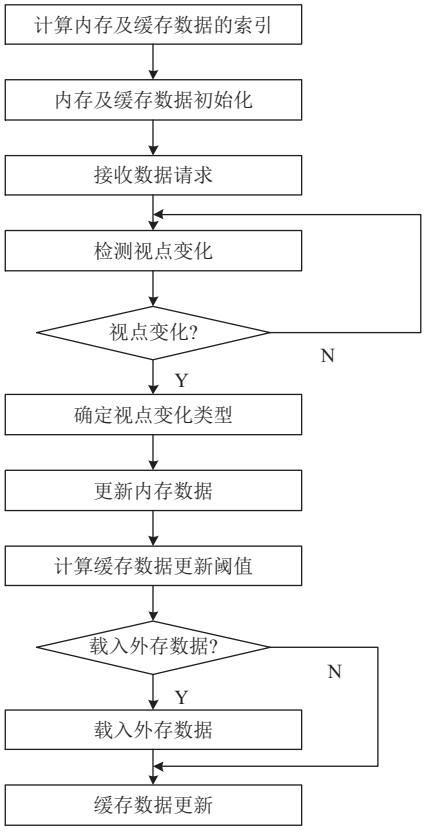

图 4 数据调度算法流程图

1) 获取当前视点位置, 得出内存和缓存区需要加载的数据索引 (地址).

2) 采用数组 data[*]存储缓存区地形数据, 并通过

固定长度的先进先出队列 history, 记录被加载到缓存区的数据文件以控制缓存区的大小.

3) 当视点发生变化时, 判断是否需要更新内存数据.

4) 如果需要更新的内存数据在缓存区内, 直接从缓存区读取数据即可; 如果需要更新的内存数据不在缓存区内, 需要先计算缓存数据更新范围, 清除缓存区中无用数据并载入外存数据至缓存区; 当视点发生变化时, 载入视点可视范围内的数据, 删除两倍视野范围以外的数据. 即保留了一倍视野范围的缓存区. 重复 3) 和 4).

4 试验与总结

4.1 实验设计

本文使用了 C++作为开发语言, 以 OpenGL 作为三维渲染引擎, 在 Xcode 的环境下, 使用 GLUT、GDAL 库, 实现了基于四叉树的 LOD 地形实时动态显示系统. 本系统运行在双核 CPU 的微机上, 采用四线程, 硬件配置如表 2.

表 2 开发环境配置

硬件	配置参数
处理器	1.7 GHz Intel Core i7
内存	8 GB 1600 MHz DDR3
显卡	Intel HD Graphics 5000 1536 MB
外存	SSD 256 G

4.2 结果对比

如图 5、6、7 所示, 对应现实空间坐标点为北纬 23 度, 东经 121 度, 实际空间地理位置为台湾的不同视点渲染效果图.

4.3 性能分析

本文实现了基于四叉树的 LOD 地形实时动态显示系统, 分别采用并行渲染的方式与非并行渲染的方式以进行性能比较 (参见表 3). 可以看出, 采用并行渲染的帧率明显高于非并行渲染帧率, 内存占用量基本保持在 825 M 左右 (300 个缓存地形数据块文件), 说明多核 CPU 并行渲染可以有效的提高三维 DEM 渲染的性能, 并且内存数据也稳定在合适的规模;采用并行渲染的帧率远远高于非并行渲染方式, 说明并行的效果显著, 达到了提高渲染性能的目标.

5 结语

本文利用多核 CPU 微机的并行计算能力, 在大规模 DEM 三维地形数据渲染过程中, 采用并行计算的方式, 提高了系统的渲染效率. 本文所包含的工作主要分为以下几个部分: 首先, 实现了基于四叉树的 LOD 地形渲染算法, 基于此, 结合并行计算方法, 采用生产者消费者模型, 实现了 LOD 算法的并行化. 除此之外, 在研究并行技术的同时, 研究了大规模地形渲染时的海量数据的组织与调度方法, 通过设置一定区域的内存缓冲区以及数据更新机制实现了大规模数据的加载与处理.

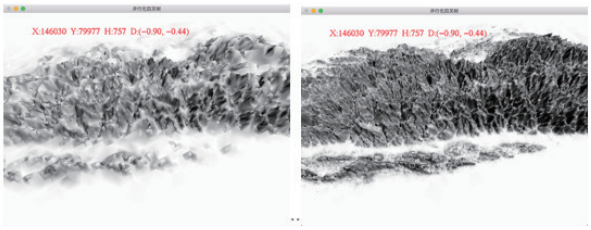

图 5 不同细节阈值对比图

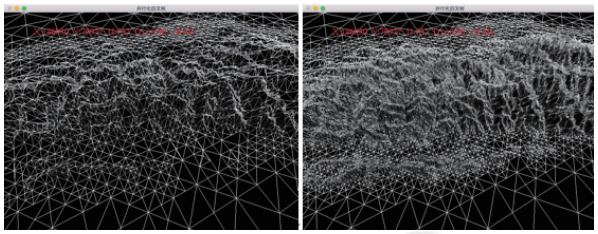

图 6 不同细节阈值对比图 (线框模式)

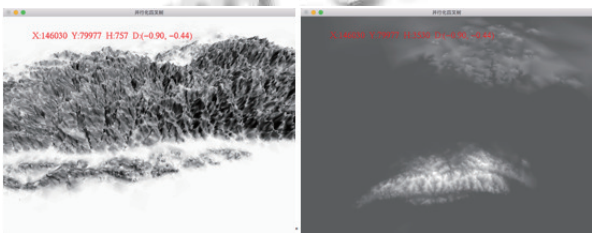

图 7 不同高度对比图

表 3 地形渲染实验结果性能对比

数据量(GB)	并行渲染帧率(帧/秒)	非并行渲染帧率(帧/秒)
1	67	30
8	59	26
16	55	19
30	53	21
52	52	23

地形渲染是大规模场景渲染的核心部分, 随着微机硬件的发展, 在微机上实现大规模三维地图渲染已经具有一定的硬件基础. 本文基于多核 CPU, 实现了大

规模地形数据渲染的实验, 所取得的理论与技术方面的成果可作为大规模场景渲染的理论基础和技术基础.

同时, 关于大规模场景渲染, 还有很多问题值得我们进一步研究与探讨. 比如 DEM 三维可视化的仿真效果, 如何更好的模拟真实的地理环境, 在渲染地形数据的同时, 也渲染出光照、纹理、阴影、不同的地貌特征等; 比如结合 GPU 和本文使用的多核 CPU 共同实现三维地形渲染, 从而实现更高效率的并行计算, 这些都是需要进一步开展的研究内容.

参考文献

1 陈路. 3D 游戏引擎技术—大规模场景实时图形渲染的研究与实现[硕士学位论文]. 成都: 电子科技大学, 2005.

2 陈景广, 佘江峰, 宋晓群, 等. 基于多核 CPU 的大规模 DEM 并行三维渲染. 武汉大学学报•信息科学版, 2013, 38(5): 618–621.

3 Zhai R, Lu K, Pan WG, *et al*. GPU-based real-time terrain rendering: Design and implementation. Neurocomputing, 2016, 171: 1–8. [doi: 10.1016/j.neucom.2014.08.108]

4 王冲. 大规模地形的快速几何绘制与实时纹理映射技术研究[硕士学位论文]. 天津: 中国民航大学, 2009.

5 刘晓平, 凌实, 余烨, 等. 面向大规模地形 LOD 模型的并行简化算法. 工程图学学报, 2010, 31(5): 16–21.

6 周发亮. 基于四叉树的 LOD 技术在地形渲染中的应用. 计算机仿真, 2007, 24(1): 188–191.

7 Wu J, Yang YF, Gong SR, *et al*. A new quadtree-based terrain LOD algorithm. Journal of Software, 2010, 5(7): 769–776.

8 Yu WL, Zhang LM, Zhang BQ, *et al*. Large-scale LOD adaptive terrain rendering. Applied Mechanics and Materials, 2012, 220-223: 2450–2453. [doi: 10.4028/www.scientific.net/AMM.220-223]

9 任宏萍, 靳彪. 基于 4 叉树的 LOD 地形实时渲染技术. 华中科技大学学报 (自然科学版), 2011, 39(2): 6–10.

10 赵庆. 大规模地形数据调度与绘制技术研究与实现[硕士学位论文]. 成都: 电子科技大学, 2011.

11 郑笈, 李思昆, 陆筱霞. 大规模场景绘制的存储数据调度组织研究. 节能环保 和谐发展——2007 中国科协年会论文集 (一). 武汉, 中国. 2007. 6.

12 Yang C, Dai SY, Wu LD, *et al*. Smoothly rendering of large-scale vector data on virtual globe. Applied Mechanics and Materials, 2014, 631-632: 516–520. [doi: 10.4028/www.scientific.net/AMM.631-632]