高技术通讯 2009 年 第 19 卷 第 8 期:833 ~ 838

doi:10.3772/j.issn.1002-0470.2009.08.011

基于多分辨率数据源镶嵌的大规模地形实时绘制算法①

郎 兵②* ** 方金云* 韩承德*

(*中国科学院计算技术研究所空间信息处理技术实验室 北京 100190)

(**中国科学院研究生院 北京 100049)

摘 要 针对不同分辨率的测量地形数据源,提出了镶嵌四叉树模型,用以实现对多分辨率数据源的镶嵌,以及与细节层次(LOD)算法的融合。设计了基于此模型的批 LOD 算法,给出了相关公式,并设计了一种从 CPU 到 GPU 的渐进 LOD 数据传输方法。分析了镶嵌四叉树的场景浏览模式,设计了基于此模型的双向两级外存数据调度算法,通过水平、高度两个方向的预测和两级数据预取实现数据页的动态更新,实现了三维地形场景的快速构建和实时浏览。实验表明,该算法能够快速处理大规模多分辨率数据源并且能够充分发挥 CPU、GPU 与 I/O 的效率,从而在有限环境下进行快速实时绘制。

关键词 镶嵌四叉树(TQT),细节层次(LOD),渐进 LOD 传输,地形绘制,外存模型

0 引 言

大规模地形数据的实时绘制在数字城市、战场虚拟、三维地理信息系统(GIS)中有着非常重要的应用。随着数字测量技术的飞速发展,大量地形数据都是按照不同分辨率进行采集的,例如美国地质调查局(USGS)全球数字高程模型(DEM)数据就涵盖 7.5 分、15 分、30 分和 1 度四种测量分辨率。而现有地形算法使用的常规四叉树结构[1-3]或者二叉树结构[4-6]都是在单一分辨率数据上动态生成不同简化细节,无法支持多分辨率数据源的镶嵌。

地形数据庞大的数据量远远超过了当前系统的实时处理能力。常用两种方法对系统进行加速,一种是采用地形细节层次(level of detail,LOD)技术简化场景的复杂度,其中传统的实时自适应细分算法[1,2,4]是实时动态地对地形网格进行合并与细分,需要消耗大量的 CPU 时间。随着图形硬件的发展,批 LOD(aggregated LOD)算法[3,5,6]将三角形组成簇或组进行 LOD 选择,减少了 CPU 负载并且充分利用了 GPU。文献[5]提出的 CABBT 算法,文献[6]提出的 BDAM 算法,都是用不规则三角网(triangulated irregular network,TIN)表示三角形簇,把几帧相关的批三角形存储到显存,进行大规模地形绘制,但是动态的分裂与合并操作仍然需要较大的 CPU 开销。另一种是外存(out-of-core)调度算法,通过对数据进行实时的内、外存高效调度,从而能够处理超过系统内存的大规模地形数据。文献[2]提出基于空间填充曲线组织外存(out of core)模型数据,文献[7]使用分块多分辨率存储策略,块内采用空间填充曲线。但是它们均无法对不同分辨率的多数据源进行控制与调度。

针对上述问题,本文提出一种基于多分辨率数据源镶嵌的大规模地形实时绘制算法,提出了镶嵌四叉树模型,实现了对不同分辨率的多数据源的镶嵌。设计了基于镶嵌四叉树的批 LOD 算法,通过渐进 LOD 批传输提高 GPU 的效率,减少 CPU 计算。并基于镶嵌四叉树进行分层增量式数据调度,通过水平、高度两个方向的预测和两级数据预取实现数据的动态更新,从而实现对多源海量地形数据的快速实时绘制。

1 算法描述

为了组织和表示测绘采集的不同分辨率的多源地形数据,同时与细节层次算法进行融合,本文提出了镶嵌四叉树(tessellated quadtree,TQT)模型,并定义了以下几个结构:

数据四叉树(data quadtree,DQT):把不同分辨率的多源地形数据镶嵌成为四叉树结构,树的层次由

① 973 计划(2004CB318202)和 863 计划(2001AA135210,2002AA114020)资助项目。

② 男,1981 年生,博士生;研究方向:3 维 GIS;联系人,E-mail: langb@ict.ac.cn

(收稿日期:2008-06-16)

多源数据的分辨率级数确定。

LOD 四叉树(LOD quadtree, LODQT):对数据四叉树的每个结点,对应镶嵌的基于 LOD 算法生成的四叉树。

镶嵌四叉树:每个结点都镶嵌了 LOD 四叉树的数据四叉树。

1.1 算法框架

整个算法框架如图 1 所示,由对数据的预处理阶段和实时调度与显示阶段组成。在预处理阶段,算法将多源地形数据镶嵌为数据四叉树,对树中结点计算层次因子,并映射嵌套 LOD 四叉树,其数据网格经过视点无关的层次简化后以三角形条带形式分层存储,最终形成镶嵌四叉树模型和相应的索引文件。在实时阶段使用多线程机制,主线程负责对数据进行视锥裁减,计算合适的分辨率数据和细节层次,并预取数据。子线程则实时完成数据的内、外存换页。最后图形处理器(GPU)将视点内的有效数据进行渲染,得到相应的场景。整个过程中,CPU 和 GPU,内存、显存和 I/O 分工合理,使用平衡,构成协调工作的一个整体系统。

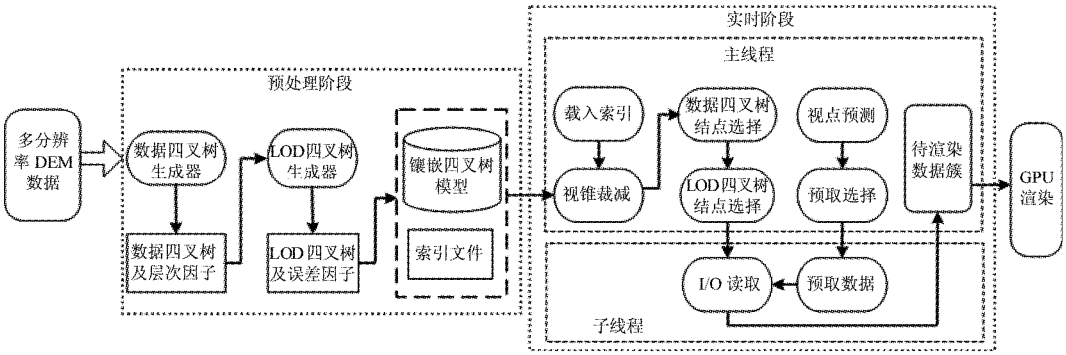


图 1 算法框架图

1.2 镶嵌四叉树数据组织

镶嵌四叉树模型构建首先生成数据四叉树模型,将不同分辨率的多源原始地形数据按照分辨率层次构成数据四叉树中不同层次的结点,为了避免在拼接时结点之间可能出现的裂缝,每个结点四周都增加一个垂直的条带,该条带共享结点的边界数据,以静态三角形的形式存储,与周围结点没有依赖关系。然后对每个结点,计算其对应的 LOD 四叉树模型,构造动态细节层次。

对 LOD 四叉树的网格细节层次使用条件限制四叉树的三角形简化,定义高程数据点为 $h_i = z_i(x_i, y_i)$,树的最高分辨率数据就是相应的数据四叉树结点,表示为 $H_n = \{h_0, \cdots, h_m\}$。为了实现数据从 CPU 到 GPU 的渐进传输,数据的层次关系必须能够满足隐式的内部嵌套,因此在进行层次简化时使用两个约束条件:

条件 1:定义简化的分辨率层次为 $H_n^i = \{h_0, \cdots, h_l\}$,则其必须满足 $H_n^i \subseteq H_n$,且 H_n^0 必须大于或等于 $(2^7+1)\times(2^7+1)$。

条件 2:设 H_n^i 中任一三角形为 T_n^i,则 H_n^{i-1} 中的对应区域三角形 T_n^{i-1} 也必需满足 $T_n^{i-1} \subseteq T_n^i$。

在细节层次构造过程中,算法使用 top-down 的网格分裂方法,按照 LOD 四叉树的两个约束条件控制分裂的进程,同时控制同一层次的结点中所有两个结点共享的边都不再分裂,从而保证层次数据网格不会产生裂缝。在分裂时采取对称的分裂构造方式,保证层次数据的视点无关性。此构造过程属于预处理阶段,构造过程如图 2 所示。

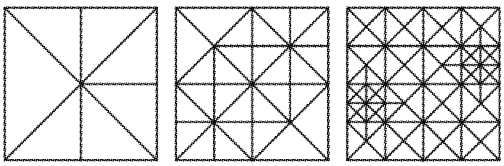


图 2 LOD 四叉树的网格简化

1.3 镶嵌四叉树 LOD 算法

镶嵌四叉树 LOD 算法包括数据四叉树数据结点层次的计算和此结点对应的 LOD 四叉树的细节层次计算两部分。

数据四叉树是不同分辨率的采集数据,体现了数据的高度层次。设数据四叉树第 i 层数据的分辨率为 m 米,第 $i+1$ 层为 n 米,如图 3 所示,通过由俯视图到侧视图的过渡,将数据分辨率差值映射为高度投影差,然后再将高度投影差映射到世界坐标系下的高度阈值,得到计算高度阈值的近似公式 $\rho(m,n) = \left(\frac{n-m}{\phi} - t\right)\times\mu$,其中 ϕ 表示屏幕误差

容忍度,μ 表示高度分辨率容忍度,t 为修正参数。

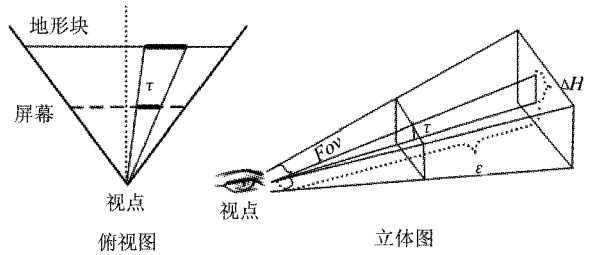


图 3　投影转换示意图

在预处理阶段,为数据四叉树计算出各个层次结点的阈值,在实时阶段,定义数据层次选择函数 $D(v)$ 为视点与地面的垂直距离,按照下述规则对数据四叉树进行层次选择:

//从 DQT 根结点开始计算

if ($D(v) > \rho_i(m,n)$)

calculate LODQT of node; //计算当前结点对应的 LOD 四叉树

else

recursively calculate 4 *children*; //递归计算当前结点的四个孩子结点

LOD 四叉树是对数据四叉树结点的 LOD 建模,决定了地形网格的构成,算法通过投影误差来控制 LOD 层次,对 LOD 四叉树任一区域,定义 L_i 为 $m \times m$ 的第 i 层数据, L_{i+1} 为 $n \times n$ 的第 $i+1$ 层数据,定义两层之间世界坐标系下的最大高度差为 $\Delta H = \max|L_{i+1} - L_i|$, 计算为 $\Delta H = \max\left|\underset{\substack{x=0,\cdots,n\\y=0,\cdots,n}}{H_{l_{i+1}}}(x,y) - \underset{\substack{w=0,\cdots,m\\v=0,\cdots,m}}{H_{l_i}}(w,v)\right|$, 按照图 3 所示,算法通过屏幕投影推导出 LOD 四叉树的误差因子阈值公式为 $\varepsilon = \dfrac{\Delta H \times R}{2\tau} \times \tan(Fov/2)$, Fov 为纵向视域角, $\tan(Fov/2)$ 为视锥近平面距离与高度的比值, R 为屏幕的分辨率, τ 为分辨率误差系数。在预处理阶段,就为 LOD 四叉树结点计算阈值,在实时阶段,定义细节选择函数 $D'(v)$ 为视线距离,即区域中心与摄像机的距离。按照下述规则进行细节选择:

if ($D'(v) > \varepsilon$)

Put level H_n^i *of this node into list*; //将当前细节结点加入绘制队列

else

recursively calculate 4 *children*; //递归计算当前结点的四个孩子结点

在此处,算法的区域以 LOD 四叉树结点为单位,由上述推导过程可见误差因子使用的是整个结点的最大误差,相比逐顶点计算,虽然会增大绘制时三角形数量,但是可以减少 CPU 的计算量,发挥批 LOD 计算的优势。

1.4　渐进 LOD 数据传输

按照模型构建规则,镶嵌四叉树具有分层隐式嵌套的特性,因此本文提出从 CPU 到 GPU 渐进 LOD 数据传输策略,通过分层批量传输,充分利用显存 cache 数据,减少内存和显存之间数据总线的传输数据量,提高系统速度。

对 LOD 四叉树任一结点,用 H_i 表示结点的第 i 层数据,传输规则的伪码描述为:

if ($n == 0$) //第一次传输

transmitted _ level = 0; //已经传输的数据层次标记为 0

transmit _ data = H_0; //传输最低数据层次 H_0

current _ level = 0; //当前传输的数据层次标记为 0

else if (*current _ level* < *transmitted _ level* || *transmitted _ level* == *max*)

//当前传输层次小于已经传输的层次或者已经传输的层次已达最大

transmit _ data = 0; //不需要数据传输

else if(*current _ level* > *transmitted _ level*) //当前传输层次大于已经传输的层次

transmit _ data = ($H_i - H_{i-1}$); //传输数据增量

transmitted _ level = *current _ level*; //已经传输的数据层次标记为当前传输层次

对于一个 $2^n \times 2^n$ 的数据,理想状态下(不考虑边界裂缝与三角化方法的差异),不使用渐进传输,从粗到细传输的数据总量为 $\sum_{i=1}^{n}((2^{i-1}+1)^2)$, 按照上述规则传输的数据总量为 $\sum_{i=2}^{n}(2^{2i-3} + 2^{i-1} + 2^{2i-4}) + 4$, 可以减少传输的数据量为 $\sum_{i=2}^{n}(2^{2i-3} + 2^{i-1} - 2^{2i-4} + 1)$。

为了与分层嵌套模型和渐进传输规则相适应,算法设计了分层增量存储结构和动态顶点索引。如图 4 所示,顶点数组(VertaxArray)中的每一层数据实心部分表示新的 LOD 层次相对上层 LOD(虚线部

分)的增量顶点数据。随着 LOD 层次的逐渐精细,只传输图中实心部分的增量顶点,原有上层顶点在显存中的存储位置不变,为了能够动态调整数据的三角化,对于每层 LOD 的顶点缓冲都建立对应的索引数组(IndexArray),如图 4 所示,索引数组与顶点数组按照 LOD 层次一一对应,索引数组通过每帧实时计算顶点三角化索引,控制了顶点数组中数据网格构成方式。

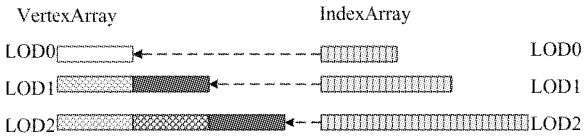


图 4 数据渐进存储结构

在实时阶段,算法将裁减后的待渲染四叉树结点群组成的数据簇统一批传输到显存,由于每次只传送数据增量,减少了数据的传输量,并且显卡通过直接内存存取(DMA)直接访问加速图形端口(AGP)内存取数据到显存,GPU 可以和 CPU 并行工作,从而提高了系统的整体速度。

1.5 out-of-core 数据调度

在交互过程中,本文把三维场景浏览操作分为漫游性移动和俯冲性移动,漫游性移动是指视点的高度变化在一定范围内,对应镶嵌四叉树模型中的数据四叉树层次不变,而俯冲性移动是指视点高度剧烈变化的移动,此时模型中的层次需要快速切换。

对于漫游性移动,如图 5(a)所示,算法使用以视点为圆心的圆形区域(在图 5(a)以虚线所示)对视锥区域进行扩展,为了简化计算,实现时以圆的包围矩形代替圆形,形成数据区 A,A 区可以有效避免视线快速任意旋转所带来的“呼吸”现象,是第一级数据缓冲区。数据区 B 是固定大小的二级数据缓冲区,为了减少 I/O 操作,B 内预取的数据只使用 LOD 四叉树中的最低 LOD 层次。当视点移动时(参见图 5(b)),系统使用 B 区内的数据结点补充 A 区的数据,然后增量读取 A 区内所需的 LOD 数据层次,空闲时预取 L 形数据补充 B 区数据。

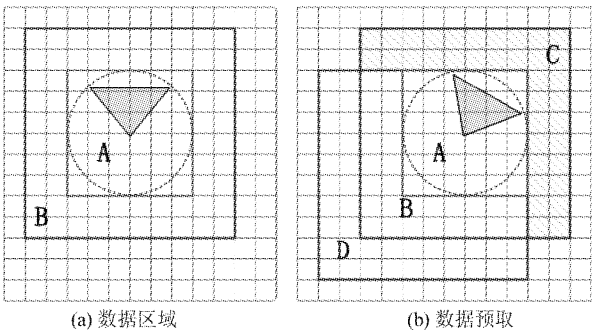


图 5 调度区域示意图

对于俯冲性移动,需要处理视点高度持续变化引起的数据四叉树层次更迭。因此除了在水平方向进行上述操作外,还要进行高度方向的预取。数据四叉树的层次是以实际高度判别,所以在计算时只要设定当前视点高度与数据四叉树的层次高度阈值的差在指定范围内,即可预取数据四叉树下一层次的数据,预取的数据都是最低的 LOD 层次。如果俯冲速度超过换页的能力,预取的数据可以使系统保持视觉连贯性。同时数据四叉树的层次有限,高度预取发生的频率较低,不会占用资源阻塞水平数据的预取。

在实现上使用多线程机制来加速调度的过程,减少基于 I/O 的等待。具体算法描述如下:

步骤 1:主线程初始化数据 A 区与 B 区。

步骤 2:主线程通过线性索引遍历镶嵌四叉树将 A 区内的当前可见活动结点集合放入可见结点数据请求队列,此队列具有最高优先级。

步骤 3:子线程对可见结点数据请求队列的请求进行响应。

步骤 4:主线程预测计算 A 区内的数据,将 A 区内当前非可见的数据结点放入 A 区结点数据请求队列,然后进行高度预测和 B 区数据预测,把高度预取的结点索引放入高度数据请求队列,把 B 区的增量 L 形区域 C 的结点索引放入 L 增量数据请求队列。

步骤 5:子线程完成可见结点数据请求后,如果有空闲就响应 3 个预测数据请求队列的请求。按照 A 区请求队列、高度数据队列、B 区请求队列优先级由高到低的顺序进行响应。对失效数据采取 LRU(least recent use)策略释放。

步骤 6:主线程在得到当前视锥内可见数据后,按照传输规则将数据送入显存,GPU 对数据进行绘制。

在整个过程中,对内存和外存调度,A 区的预取基本上覆盖了可见结点的数据请求,所以 I/O 操作主要集中在 A 区和 B 区的预取操作,因此实时浏览时用户可以基本消除 I/O 等待时间。对内存和显存调度,由于浏览时帧与帧之间的相关性很强,同时数据在 GPU 上按照增量方式传入,在显存 cache 里可以利用已有数据,所以从内存到显存的总线数据传输量也得到控制,因此整个场景渲染速度可以保持快速稳定。

2　实验结果与分析

本文算法使用 VC7 和 OpenGL 实现,实验使用的硬件环境为 PentiumD 2.80GHz,内存 512M,显卡为 NVIDIA Geforce 7300,显存为 128M,设置窗口大小 1024× 768,颜色位数 32bit。实验数据为美国的 Puget Sound 区域。具体信息见表 1。

表 1　实验数据详细信息

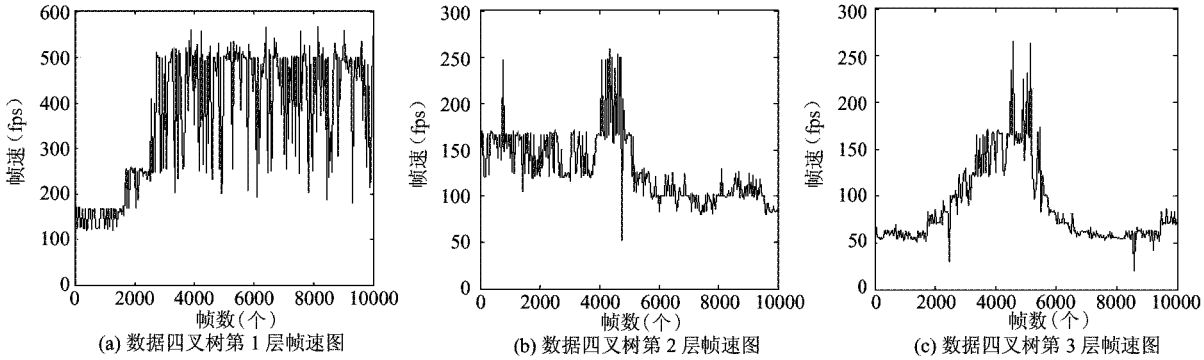
DQT 层次	格网大小	分辨率	地形文件(ASCII)	纹理文件(bmp)
1	1k × 1k	160m	4.9M	3M
2	4k × 4k	40m	78.8M	48M
3	16k × 16k	10m	1080M	768M

首先进行漫游性浏览,误差系数 τ 取 2,此时数据四叉树没有层次切换,实验结果数据如表 2 所示。

从表格 2 中可以看出,在漫游性移动的方式下,数据四叉树没有层次的更迭,以数据量最大的第 3 层来看,同等条件下,帧速的实验结果比 Ulrich 批 LOD 算法[3]的 34fps 提高了 162%。实验结果内存与数据总量比率为 83MB/1848MB≈4.5%,优于文献[7]8%的实验结果。由于采用了批 LOD 处理,实验中随着数据量的变化,CPU 峰值保持小于 50%。从而证明了在处理多源海量数据时,本文算法在帧速、内存占用和 CPU 峰值控制上都取得了较好的优化效果。各次实验帧速见图 6 所示。

表 2　漫游性浏览实验结果与比较

DQT 层次	LODQT 层次	内存峰值	CPU 峰值	每帧三角形	帧速(fps)	Ulrich 算法[3]帧速(fps)
1	5	19M	19%	160.89K	361.83	198.8
2	7	44M	32%	364.29K	128.83	75.1
3	10	83M	48%	617.57K	89.05	34.0



(a) 数据四叉树第 1 层帧速图　(b) 数据四叉树第 2 层帧速图　(c) 数据四叉树第 3 层帧速图

图 6　漫游性移动各层帧速图

在同样条件下,进行俯冲性移动的实验,屏幕误差 φ 取 2,屏幕高度系数 μ 为 40 像素,修正参数 t 为 1。从数据四叉树的最高层开始,同时进行高度和水平方向的移动,可以得到实验数据如图 7 所示。

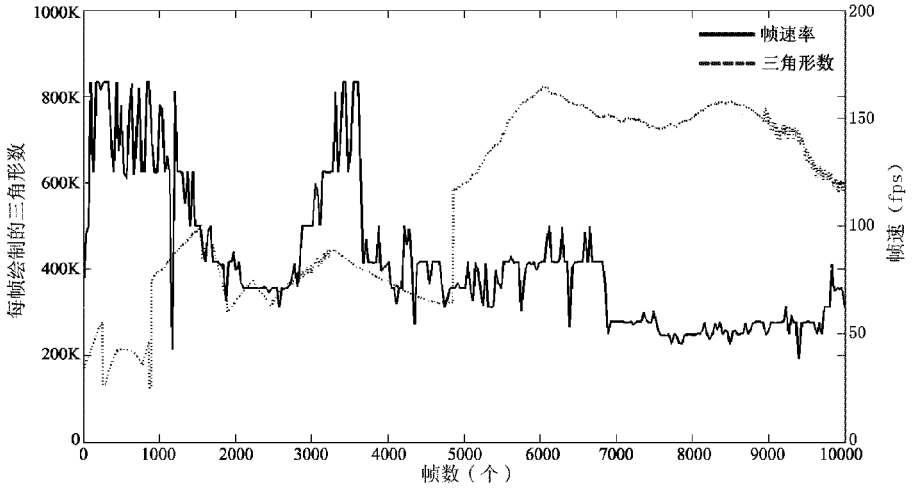


图 7　镶嵌四叉树俯冲性浏览实验结果

从图中可以看到,在数据四叉树同一层次内,LOD 四叉树的绘制帧数和三角形数都基本保持稳定,当实验中浏览高度连续降低时,数据四叉树更迭到了更细分辨率的数据层,会有帧速下降和三角形数突增的一个阈值区,然后就会稳定在相应的范围内。同时在俯冲性移动操作中,帧速都要小于漫游性移动对应的帧速,原因在于进行了高度预取,导致计算量和 I/O 负载的增大。整个俯冲性移动的平均帧数是 83.38,完全可以满足实时浏览的需要。

3 结 论

多分辨率地形数据源的处理技术对于充分利用现有各类测量地形数据具有重要的意义,本文提出了使用镶嵌四叉树模型来处理不同分辨率数据源的镶嵌,通过结合多源数据镶嵌和细节层次算法,解决了多源数据的组织和表示的问题,通过基于 CPU/GPU 渐进传输的批 LOD 算法,以及双向两级的 out-of-core 调度算法,实现了对数据的高性能绘制。通过实验数据可知,算法达到了较高的绘制效果和处理速度。在今后的工作中,将考虑加入遮挡剔除算法进一步加快绘制的速度,以及引入光滑视觉变换技术来提高视觉效果。此外随着网络地形应用需求的增多,研究在网络条件下如何实现基于镶嵌四叉树的大规模多分辨率地形数据源的组织、调度与绘制算法将是我们下一步的工作重点。

参考文献

[1] Lindstrom P, Koller D, Ribarsky W, et al. Real-time, continuous level of detail rendering of height fields. In: Proceedings of the 23rd International Conference on Computer Graphics and Interactive Techniques, New Orleans, LA, USA, 1996. 109-118

[2] Lindstrom P, Pascucci V. Visualization of large terrains made easy. In: Proceedings of the IEEE Conference on Visualization, Stuttgart, Germany, 2001. 363-370

[3] Ulrich T. Rendering massive terrains using chunked level of detail control. In: Proceedings of the 29th International Conference on Computer Graphics and Interactive Techniques, San Antonio, Texas, USA, 2002

[4] Duchaineau M, Wolinsky M, Sigeti D, et al. ROAMing terrain: Real-time optimally adapting meshes. In: Proceedings of the IEEE Conference on Visualization 1997, Phoenix, AZ, USA, 1997. 81-88

[5] Levenberg J. Fast view-dependent level-of-detail rendering using cached geometry. In: Proceedings of the Conference on Visualization, Boston, USA, 2002. 259-266

[6] Cignoni P, Ganovellt F, Gobbetti E, et al. BDAM-Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 2003, 22(3): 505-514

[7] 吴金钟,刘学慧,吴恩华.超量外存地表模型的实时绘制技术.计算机辅助设计与图形学学报,2005,17(10):2196-2202

Real-time rendering of massive terrains based on tessellated multi-resolution datasets

Lang Bing[*,**], Fang Jinyun[*], Han Chengde[*]

([*] Spatial Information Technology Laboratory, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] Graduate University of Chinese Academy of Sciences, Beijing 100049)

Abstract

For handling multi-resolution terrain data sources, the paper proposes a tessellated quadtree model to tessellate multi-resolution terrain datasets and integrate level of detail (LOD) algorithms. An aggregated LOD algorithm based on the model is given for progressive transmission of vertices from CPU to GPU. By analyzing the roaming pattern of the tessellated quadtree, a new out-of-core data paging and prediction algorithm is presented. The real-time loading of datasets is achieved by employment of two orientations of prediction and the double level paging strategy. The experimental results show that the proposed algorithm can handle large-scale multi-resolution data sources and balance the tradeoff between CPU, GPU and I/O, and realize real-time rendering under limited resources.

Key words: tessellated quadtree (TQT), level of detail (LOD), progressive LOD transmission, terrain rendering, out-of-core model