基于外存的大规模地形可视化框架

杜剑侠, 李凤霞, 战守义
(北京理工大学 计算机系, 北京 100081)

摘要: 为了解决地形模型由于受内存限制而无法进行实时绘制的问题, 作者提出了一个有效的基于外存的海量地形数据实时可视化框架, 该框架由三层体系结构构成: 管理层、调度层、自适应 LOD活动地形渲染层. 利用该框架进行海量地形的实时绘制, 理论上可以达到绘制速度与地形规模基本无关. 并给出了该框架一个实现, 通过对 36块具有 500万个三角片的地形进行测试, 平均可以达到 25帧 / s以上的实时绘制速率, 验证了该框架的有效性.

关键词: 基于外存的地形; 多分辨率; 自适应地形对象; 地形调度
中图分类号: TP301. 6 **文献标识码**: A **文章编号**: 1007 - 855X(2006)05 - 0001 - 05

A General Framework for Out - of - core Visualization of Large Terrains

DU Jian-xia, LI Feng-xia, ZHAN Shou-yi
(Department of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Real - time rendering is limited by the capacity of memory when dealing with large terrains. To overcome the restriction of memory, an efficient framework is presented, which is made up of 3 layers: management, schedule, and real - time rendering. In theory, the rendering rate is almost independent of the scale of terrains. A test of 36 terrains with 5 millions triangles shows that the rendering rate can reach 25 fps, which proves the framework is valid and efficient.

Key words: out - of - core data terrain; multi - level of detail; adaptive terrain object; scheduler

0 引 言

在许多应用中, 地形数据集通常太大而无法全部容于内存之中. 这样就要考虑在快速的内存和较慢的外存 (例如硬盘)之间进行数据交换, 这将严重影响系统的效率, 已经成为大规模数据处理的一个瓶颈.

如何合理地组织数据, 实现数据从硬盘到内存快速交换和读取, 是近来 Out - Of - Core数据设计研究的主要内容. Vitter给出了基于外存算法的数据层次设计方案[1], 在现在的计算机体系结构 (CPU快速缓存、系统内存、硬盘等), 各种存储器的有效使用, 对提高性能会起到很大的作用. 在图形学中, 大规模数字几何处理[2 - 4]和科学可视化中[5, 6]也提出了该问题的求精方法. 由于大规模地形数据不能全部贮存于系统内存中, 要实现数据的实时可视化, 数据的组织和存储是一件很重要的事. 基于外存 (Out - Of - Core)的处理算法的关键是重新组织数据, 使得数据在外存模式下的访问频率尽可能低, 该领域的处理方法在并行及分布式计算中也很重要.

1 基于外存的地形实时可视化框架

1. 1 基本概念

活动地形和不活动地形: 驻留内存中的地形我们称之为活动地形, 不驻留内存的、以某种存储结构在磁盘中保存的地形称为不活动地形.

收稿日期: 2005 - 11 - 25. **基金项目**: 国防科工委仿真技术项目预先研究专题 (项目编号: 413040402).
第一作者简介: 杜剑侠 (1976～), 女, 博士生. 主要研究方向: 虚拟现实与仿真技术. **E - mail:** dujx@bit. edu. cn

自适应地形对象:活动地形以某种封装良好的方式存在,隐藏了内部具体的数据结构和 LOD 算法,它根据视点和投影锥体的变化自适应地改变所显示地形的 LOD 状态.

LOD 算法:实施于活动地形之上的多分辨率显示算法.

地形调度算法:实施于活动地形和不活动地形之间的内外存交换算法.

1.2 三层体系结构

基于外存的地形实时绘制问题的解决主要有三个关键点:一是调度,二是管理,三是地形在不同存储状态之间如何实现平滑无缝的转换,显而易见,它涉及到地形的内存结构,而地形的内存结构决定于 LOD 算法[7-9]. 本文提出了一种基于外存的地形实时可视化框架,用于解决上述三个问题,该框架是一个三层的体系结构,从下往上分别是:地形数据管理层、地形块调度层、自适应 LOD 活动地形渲染层,如图 1 所示.

地形模型数据管理层位于实时渲染系统的最低层,是整个系统的基石,负责数据的组织安排以及从外存装载数据到内存.

调度层位于中间层,实时跟踪视点变化、计算视锥体,并根据预视算法计算哪些地形块需要调入,哪些需要调出. 然后将计算结果告诉管理层,由管理层负责具体实施调入和调出.

渲染层对当前的活动地形进行处理,生成与视点相关的自适应层次细节的地形模型,然后交付渲染引擎进行绘制. 地形渲染层本质上是一个对象阵列,阵列的单元对象是一个自适应地形对象,地理上对应一块矩形的地表.

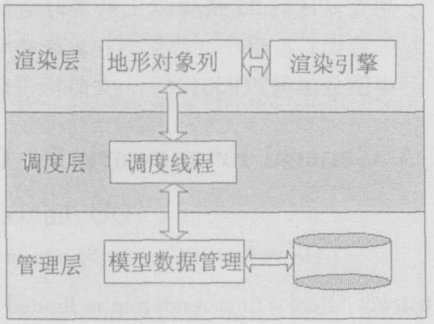


图1 地形渲染系统三层结构

Fig. 1 Hierarchical structure of terrain rendering

2 自适应 LOD 活动地形渲染层

为了提高绘制速率,视域剔除是最基本的处理方法. 视域剔除虽然可以大量地降低系统的负担,但当视点与地形距离较远时,视景体也可以延伸覆盖一个非常大的场景,因此不能从根本上解决该问题. LOD 技术是采用的较多的、比较有效的方法. 因为位于视域内地形的不同部分,根据其曲率以及与视点的距离的差异,对视觉的贡献也是有差别的. 本文采用连续 LOD 算法,即自适应 LOD 算法对视域内的地形进行处理,以获得满足视觉需求的最低分辨率的地形.

活动地形的 LOD 状态由函数 w = w(p, q, r)决定. 其中位置 p是一个三维向量 p(*x*, *y*, *z*), *x*, *y*, *z*是笛卡尔坐标;姿态 q也是三维向量 q(*h*, *q*, *r*), *h*, *q*, *r*是 head, pitch和 roll 视锥体 r是一个六维向量 r(*l*, *ri*, *b*, *t*, *n*, *f*). 自适应 *LOD* 活动地形渲染层的在构成上有两个要点: 1)从数据与算法相分离的角度,它由各种 *LOD* 算法和活动地形数据构成. 2)从面向对象的角度,它由一个对象阵列,其中每个对象是自适应地形对象,它封装了活动地形模型数据和 *LOD* 算法.

地形渲染层的核心部分是一个对象阵列,阵列的单元对象是一个自适应地形对象,地理上对应一块矩形的地表. 它是自适应的,它与外部调用者的接口是:地形对象接受视点和投影体,然后在对象内部执行 LOD 算法,决定哪些部分详细显示,哪些部分粗略显示,哪些部分不显示.

本文采用了 HOPPE的 VDPM 算法 (与视点相关的渐进格网算法)作为自适应 LOD 层处理的算法. 因为目前关于与视点相关的 LOD 算法,只有 VDPM 算法支持 TN 格网地形模型,而 GRID 格网是 TN 格网的特例,故采用针对 TN 格网的 LOD 算法使该框架更具通用性. 地形渲染层设计的数据结构和主要函数伪代码描述如下:

(1)自适应地形对象 TerrainObject:

```
interface TerrinObject{
public Envelope getEnvelope();
```

```
public void render(Frustume fru, Vector3 eye,Vector3 orient);
}
```

其中几个参数的含义:

Frustume:　　**代表四棱锥投影体**.

Vector3:　　**三维向量,用于定义视点位置和方向**.

Envelope:　　TerrainObject**的外包络**.

TerrainObject**是一个接口,它的方法都是公有的纯虚函数**.

(2)**地形阵列** TerrainArray:

```
class TerrainArray{
privateVector m_Tarray;
publicVoid render(Frustume fru, Vector3 eye,Vector3 orient);
}
```

(3)**地形阵列的渲染**:

```
void TerrainArra::render(Frustume fru, Vector3 eye, Vector3 orient) {
int n =m_Tarray.size();          //获得当前地形阵列的大小
for(int I=0; I< n; I++){
TerrainObject tob = m_Tarray.elementAt(i);
Envelope env = tob.getEnvelope();
if(GeometryMath.intersect(fru, eye, orient, env)
tob.render(fru, eye, orient);        //地形的实时渲染
}
}
```

3 地形调度层

调度层通过监控视点与可见区的变化,调入进入可见区或者与可见区距离缩短到某个阈值的不活动地形,调出离开可见区或者与可见区较远的地形,保证活动地形总是覆盖可见区,从而保证视景中环境地形的完整. 这种监控可以是主动的 (异步)或被动的 (同步).

地形调度层的实现由一个 Scheduler**类构成,其中有两个主要函数**, Schedule()**和** switchToLayer()**构成,伪代码描述如下**:

```
class Scheduler{
int curLayer;
double x, y, z;
Boolean switchLayer(); //判断是否越层;是,设置层号返回真,否,返回假.
void switchToLayer(int layer, double x, double y, double z);
void Schedule();
}
```

调度线程

```
void Schedule::Schedule() {
while(1) {
if(switchLayer()) {
switchToLayer(curLayer, x, y, z); //切换到当前层
continue;
}
```

```
if( EyeOnCenter( eye ) ) {
ReconstructTArray();      /重新构建地形阵列
continue;
}
sleep (TimePeriod);       /等待下一次的调度
}
}
```

4 大规模地形管理层

管理层位于系统最底层,建立在文件系统或者数据库系统提供的一般的数据存储、数据管理的功能之上,以增加一个面向地形调度的抽象层,为上层调度提供服务. 该层与调度层的关系主要是数据关系.

大规模地形管理的一个核心功能是为地形数据提供索引,也就是从位置到地形模型数据的映射关系. 调度模块以某种形式给出所要调入的那部分地形的位置,管理层根据它进行空间查询,结果是一个地形单元的集合. 本文采用了金子塔模型结构对地形数据进行管理. 本系统中采用离散 LOD 和连续 LOD 技术构建地形金字塔. 构造方法如下:

(1) 输入:原始的地形数据,如:DEM、TIN、GRID 地形等等.

(2) 处理:

①视点与地形之间的距离集合构成了一个有限区间或无限区间,将该区间分成 n 个小区间,每个区间称为一个层,如图 2 所示.

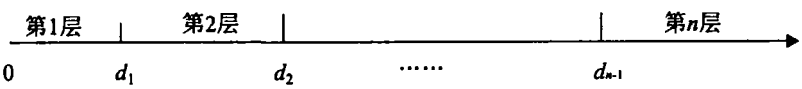


图2 层的划分

Fig. 2 The partition of layers

②将地形分成大小相等的地形块.

③对每个地形块交互使用离散 LOD 技术和连续 LOD 技术构造 n 个地形模型,每个模型对应一个层,模型的细节程度与所在的层数呈反比.

④ n 个地形模型之间存在数据冗余,类似于离散 LOD 技术.

⑤每个地形模型对应一个层,即对应一个距离区间. 在该区间内,一个地形模型实际上蕴涵着无数的地形模型,这些地形模型之间无数据冗余,类似于连续 LOD 技术.

⑥每个层的模型实际上是由诸多地形块构成的有机体,同一层的地形块大小一致,不同层的地形块大小与层呈正比.

(3) 输出:混合 LOD 的地形模型. 若原始地形被分成 m 块,系统定义的离散级别为 n 层,则生成的地形模型约 $m * \log_2 n$ 个,每个模型均是连续 LOD 模型,我们称之为自适应地形模型.

上面生成的地形金字塔模型数据采用平面结构存储在外存,采用索引结构对其进行管理,如下所示:

```
class TerrState{
int BlockID;
char * Address;
float limit[6];
};
```

其中 BlockID 是对地形块数据的编号是为了便于管理而设置的; * Address 是地形块数据的文件名; limit[6]分别代表了此块地形在 x, y, z 坐标方向上的极限值,以便于确立地形块在空间的包围盒,进行视域剔除和碰撞检测.

5 试验及分析

5.1 实验设计

硬件环境: Pentium3 1.0GHz CPU, Geforce4显卡 256M内存.

软件环境:WINDOWS 2000操作系统, VC6.0开发平台, OPENGL图形包.

地形数据: 36个地形块,共有 500万个三角片. 对地形构造多分辨率模型时定义 3个离散级别的LOD,即金字塔拥有 3个层.

测试用例:观察者经历金字塔模型的每个层,在各层内部进行漫游.

测试参数:各层帧速率 (平均,最高,最低),多边形数;实时渲染效果.

5.2 测试结果

表 1显示了观察者经历金字塔模型的各个层时地形实时渲染的帧速率,从表中数据可以看出,帧速率平均都在 25帧/s以上,可以达到实时渲染的要求 (15帧/s以上);表 2显示了观察者经历金字塔模型的各个层时地形实时渲染的每帧的三角片数目,表中数据表明,无论在金字塔的那个层,每帧绘制的三角片数目基本恒定,说明在该框架下,无论可见区内三角片的数目多么庞大,基本上对实时绘制没有太大影响,从而保证了实时的刷新率. 图 3显示了实时漫游过程中不同时刻的渲染效果图,其中图 3(a)和图 3(b)是加载了纹理的效果,图 3(c)是线框图,从中可以更清楚看到实施了 LOD算法后的效果.

表 1 帧速率 (单位:帧/s)

Tab.1 Frame rate (unit: fps)

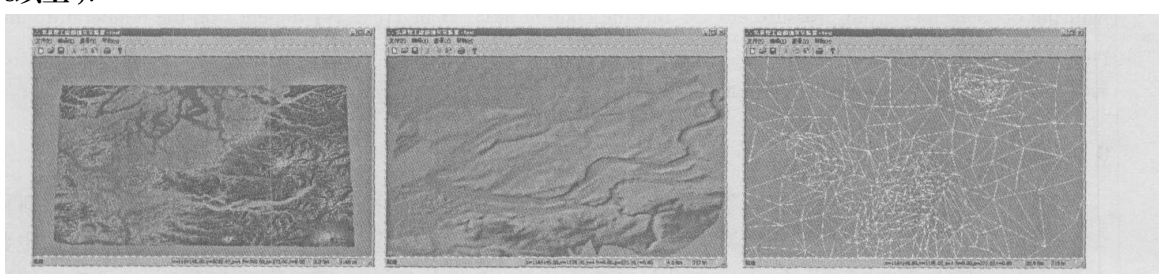
层	平均	最大	最小
1	25	53	16
2	27	40	15
3	29	38	22

表 2 多边形数目

Tab.2 Amount of triangles

层	平均	最大	最小
1	6 752	11 538	4 309
2	7 082	13 021	5 147
3	6 918	14 030	4 283

综上所述,采用该框架,地形实时渲染系统的刷新速率基本不受数据规模的限制,同时表明,即使数据存在外存,在保证视觉效果的前提下,也可以获得实时的刷新率 (25帧/s以上).



(a) t_1时刻渲染图　(b) t_2时刻渲染图　(c) t_3时刻线框图

图3 渲染效果图

Fig. 3 Effect of terrain rendering

6 结 论

当海量数据地形规模延伸到外存时,绘制受到内外存交换速度慢的限制而无法获得实时刷新速率. 为解决这一问题,本文提出了有效的基于外存的海量地形数据实时可视化框架,该框架由三层体系结构构成:管理层、调度层、自适应 LOD活动地形渲染层. 文中对三层结构的构成和各层之间的接口关系、实现技术进行了阐述,并通过试验证明了该框架的有效性.

进一步研究: 1)本文着重讨论地形的几何数据的管理及绘制,对纹理没有做深入讨论,关于多分辨率纹理映射技术是进一步研究的内容; 2)为了说明本框架的有效性,本文在调度中采用了简单的预视技术,预视技术是关系到调度策略和实时绘制效率的一个重要因素,需要进一步研究.

(下转第 13页)

- 29.

[8] Koc R, Folmer JS. Synthesis of Submicrometer titanium carbide powders[J]. Am Ceram Soc, 1997, 80(4): 952 - 6.

[9] Koc R. Kinetic and phase evolution during carbothermal synthesis of titanium carbide from ultrafine titania /carbon mixture [J]. Mater Sci, 1998, 33(4): 1049 - 55.

[10] Swift GA, Koc R, Formation studies of TiC from carbon coated TiO[J]. Mater Sci, 1999, 34(13): 3083 - 93.

[11] Rasit Koc, Lakewood, Gregory C, et al. Process for synthesizing titanium carbide[J]. titanium nitride and titanium carbonitride, US Patent No 5417952.

[12] **方民宪. 碳热还原法制取碳氮化钛的研究**[Z]. **内部资料**, 2004.

[13] Ahlen N, Johnsson M, Nygre M. Synthesis of TiN_x - C_{1-x} Whiskers, Journal of materials science[J], 1999, 18: 1071 - 1074.

(上接第 5页)

参考文献:

[1] Jeffrey S. Vitter. External Memory Algorithms and Data Structures: Dealing with MASSIVE DATA[J]. ACM Computing Surveys, 2001, 33(2): 209 - 271.

[2] Matias Y, Segal E, J. S. Vitter. Efficient bundle sorting[A]. in Proceedings of the 11th Annual SIAM/ACM Symposium on Discrete Algorithms[C]. San Francisco, California, Jan., 2000: 839 - 848.

[3] Lars Arge, Peter Bro Miltersen. On showing lower bounds for external - memory computational geometry problems[A]// External Memory Algorithms and Visualization. James Abello, Jeffrey Scott Vitter, Eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Press, Providence, Rhode Island, 1999.

[4] Goodrich M T, Tsay J J, Vengroff D. E, et al. External memory computational geometry[A]//Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science. Palo Alto, California, Nov., 1993.

[5] Yi - Jen Chiang, Cláudio T. Silva. I/O optimal isosurface extraction[A]// IEEE Visualization '97. Roni Yagel and Hans Hagen, Eds., Phoenix, Arizona, 1997: 293 - 300, IEEE.

[6] C. L. Bajaj, V. Pascucci, D. Thompson, X. Y. Zhang. Parallel accelerated isocontouring for out - of - core visualization [A]//Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium. ACM SIGGRAPH, 1999: 97 - 104.

[7] Hoppe, H. Progressive meshes[J]. Computer Graphics (SIG - GRAPH '96 Proceedings), 1996: 99 - 108.

[8] Hugues Hoppe. View - dependent refinement of progressive meshes[A]//SIGGRAPH '97 Proc., 1997: 189 - 198.

[9] Julie C. Xia, Amitabh Varshney. Dynamic view - dependent simplification for polygonal models[A]//Proceedings of Visualization '96, 1996: 327 - 334.