

文章编号:1671-5942(2013)04-0079-04

基于 Geometry Clipmap 的地形绘制技术研究^{* 1}

王 宇¹⁾ 孙永维²⁾ 王铭伟³⁾ 宋省身⁴⁾

(1)空军航空大学军事仿真研究所,长春 130022
(2)空军航空大学飞训基地,长春 130022
(3)空军航空大学作战指挥系,长春 130022
(4)空军航空大学航空航天情报系,长春 130022

摘 要 提出一种改进的 Geometry Clipmap 算法,增加了地形数据调度模块,重构了节点的组织方式,并引入了渐进纹理更新技术。仿真实验表明,改进的算法显著地提高了飞行模拟系统的工作效率和显示效果。

关键词 地形可视化;飞行模拟;Geometry Clipmap 算法;实施绘制;渐进纹理更新

中图分类号:TP391

文献标识码:A

TERRAIN RENDERING TECHNOLOGY BASED ON GEOMETRY CLIPMAP

Wang Yu¹⁾, Sun Yongwei²⁾, Wang Mingwei³⁾ and Song Xingshen⁴⁾

(1) Military Simulation Technology Institute, Aviation University of Air Force, Changchun 130022
(2) Base of Flight Training, Aviation University of Air Force, Changchun 130022
(3) Armament Department, Aviation University of Air Force, Changchun 130022
(4) Aeronautics & Astronautics Intelligence Department, Aviation University of Air Force, Changchun 130022

Abstract An improved Geometry Clipmap algorithm is presented, in which we increase the number of topographic data scheduling module, reconstruct the organization of the note, and introduce the progressive texture update technology. The experiments show that new algorithm improves the work efficiency and display effect of flight simulation system.

Key words: terrain visualization; flight simulation; Geometry Clipmap algorithm; real-time rendering; progressive texture update

1 引言

作为飞行模拟系统的重要组成部分,飞行模拟地形可视化的真实性和实时性直接影响仿真的逼真度、沉浸感及有效性。为给飞行员提供真实的训练环境,地形数据应来源于真实地形的采样模型数据,以保证逼真的模拟效果。同时,由于飞行模拟过程

中视点的运动速度较快,地形的显示速度必须能够跟上视点的改变速度,具有较高的实时性要求。因此,采用 LOD(Level of detail)技术绘制大规模地形成为重点研究方向,传统的 LOD 技术大多是基于 CPU 的,例如 Lindstrom 等^[1]提出的连续细节层次实时高度场绘制算法、Duchaineau 等^[2]提出的 ROAM 算法、Hoppe^[3]提出的渐进网格算法等,但是由于

* 收稿日期:2013-01-18

基金项目:国家社会科学基金“十二五”规划课题(BCA110020)

作者简介:王宇,助理工程师,硕士研究生,主要研究方向为虚拟仿真。E-mail:25669468@qq.com

通讯作者:孙永维,教授,硕士生导师,主要研究方向为飞行模拟器。E-mail:sunyongw@sohu.com

CPU 的设计结构问题,不能满足日益增长的需要。随着近些年来图形硬件技术的高速发展,出现了一批基于 GPU 的绘制算法,例如 Geo-Mipmap 算法、Chunked LOD 算法、Geometry Clipmap 算法等。

本文在研究 Geometry Clipmap 算法的基础上对其进行了补充和扩展,首先增加了对地形进行固定大小的分块,建立了地形的数据调度模块;然后改进了内存中 Clipmap 堆栈的数据结构,在纹理更新过程中采用了渐进纹理更新技术;并引入了 Geometry Instancing 技术,提升了地形渲染速度。

2 Geometry Clipmap 算法简介

Geometry Clipmap 算法^[4]将地形的高程图作为一张 2D 纹理图来保存,并预先把它过滤为一个包含 L 层 Clipmap 的金字塔(图 1),对每层进行窗口大小为 $n \times n$ 的采样并缓存。对于层数 L 的确定需要考虑显示最大范围、视点的位置和最终渲染结果的精度要求。同时,由于精度高的纹理处于精度低的纹理的中间位置,因此采样点个数 n 不能取成 2 的幂次方,而必须取成 2 的幂次方加 1 或减 1,才能保证下一层的纹理处于本层纹理的中间。

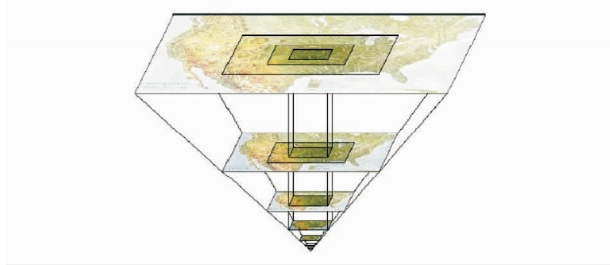


图 1 Clipmap 金字塔结构
Fig. 1 Pyramid structure of clipmap

只有最高的细节层次渲染为完整的方形网格,其他层次都渲染为空心环,空心部分依次由较高层次填充^[5]。把每层的采样的 (x,y,z) 信息分为两部分: (x,y) 坐标储存为常量顶点数据, z 坐标储存为一个单通道的 2D 纹理高度图。通过 GPU 的顶点着色器对高度图进行顶点纹理采样,获得高程数据,然后使用 GPU 的光栅管道来处理高程数据。如高程数据计算、地形细节合成、法线计算以及渲染都完全由 GPU 来执行,这样就大大减轻了 CPU 的负担,提高了大规模地形的绘制效率。

3 地形数据管理

由于大规模地形的数据量比较大,高程数据无法一次性载入内存,内存中只保留当前浏览的范围内数据,大部分数据储存在外存中,需要加载时才被载入。因此对大规模地形需要采用基于四叉树结构

的管理模式,首先将整块地形划分成长度为 2 的 n 次幂的正方块,数据的载入以地形块为基础,这样方便组成树状结构和计算机的优化处理;然后将整块地形作为根节点,依次向下分为四等块作为子节点,最终的节点末端为独立的数据块(图 2)。

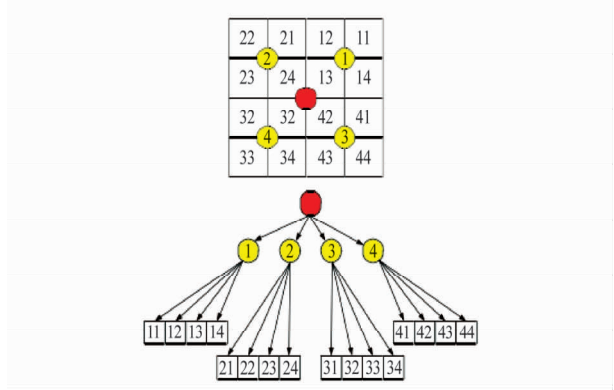


图 2 地形块四叉树结构
Fig. 2 Quadtree structure of terrain blocks

在二维平面上通过判断视截体与地形块子节点的相交情况,来更改显示列表,然后根据四叉树组织结构快速查找出位于视点范围内的地形块。由异步动态装载卸载应用技术得知,从磁盘到内存加载的地形块数据量较大,同步加载实时性较差^[6]。因此在内存中建立一个缓冲区,对地形块采取预读处理,根据视点的移动方向和速度,判断将要被访问的地形块,预先载入缓冲区。通过预加载处理,显著提高了地形显示速度,增强了地形绘制的实时性。

4 窗口更新方式

随着视点的移动,显存中地形的纹理数据和作为纹理存储的高程数据都要相对更新,每帧更新都要涉及内存与显存间的数据交换,因此针对 Clipmap 的更新特点将纹理的寻址方式设置成循环寻址方式,使不变的地方保持不变,更新需要变化的部分(图 3)。

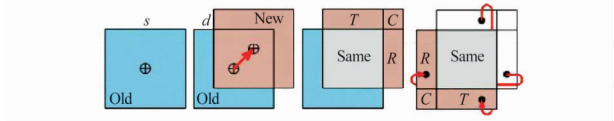


图 3 Clipmap 更新方式
Fig. 3 Update mode of clipmap

采用更新方式后,每帧所需更新的数量将大大减少。同时,由于飞行模拟的移动速度快,每帧所更新的数据量仍然较大,容易造成帧速率降低,因而在更新 Clipmap 过程中我们优先更新 LOD 等级较低的纹理和高程数据,以进一步提高地形绘制的实时性。

5 地形渲染

5.1 嵌套网格的过渡

在高程数据渲染过程中,由不同 LOD 等级网格混合而成的地形,在连接处容易产生 T 型裂缝^[7]。本文针对 T 型裂缝采用了边界消除法来进行处理。消除边界的方法就是在两个不同 LOD 层级之间引入一片过渡区,过渡区的大小可以预先指定。过渡区中的点的高程值由当前 LOD 等级地形结构的高程 z_f 和邻接层中相同位置的高程 z_c 进行插值,插值因子取决于当前位置离视点的距离,分别计算 x 轴方向和 y 轴方向的插值因子,两者中取较大的那个,计算公式为:

$$\begin{aligned} a_x &= clamp\left[\left[|x - x_v| - \left(\frac{n-1}{2} - w - 1\right)\right] / w, 0, 1\right] \\ a_y &= clamp\left[\left[|y - y_v| - \left(\frac{n-1}{2} - w - 1\right)\right] / w, 0, 1\right] \\ a &= \max(a_x, a_y) \end{aligned} \tag{1}$$

式中, x 和 y 为当前位置坐标, x_v 和 y_v 为当前视点的坐标, w 为过渡带的长度。由插值因子 a 可根据插值公式计算出最终高程值 z :

$$z = (1 - a)z_f + az_c \tag{2}$$

其中 z_f 为当前层的高程值, z_c 为邻接精度较低层的高程值。

5.2 顶点和索引缓冲

把网格的 (x, y) 值作为顶点数据,同时, z 方向的高度值保存为单通道的浮点纹理。为每层的环定义一个单独的顶点缓冲来保存 (x, y) 数据是一种有效的方法^[8]。但是为了减少内存空间和实现可视区域的裁剪,我们把环分为一些小 footprint 块(图 4)。

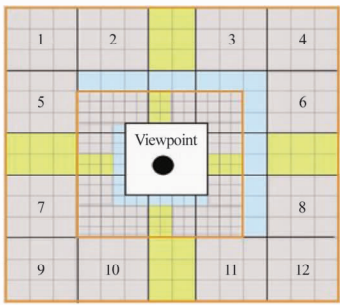


图 4 顶点缓冲区分块方式

Fig. 4 Classified blocks of vertex buffer

由图 4 可见,大部分环都由 12 个渲染单元块组成(图中的灰色区域)。因为 2D 网格是规则的,同一个层中的 (x, y) 都进行同样的变换。此外不同层次之间也可以使用统一的缩放值。因此,使用一个

只读的顶点和索引缓冲,以及一些额外的参数,就可以让顶点着色器对每个渲染单元块进行缩放和变换,完成所有渲染。

对于每一层来说,需要调用 14 次 DrawPrimitive (DP)方法:12 个块每个一次,footprint 调用一次,剩下的三角形再调用一次。但是在视平载体裁剪之后,平均每帧只需要渲染 12 个块中的 4 个。对于细节最高的层来说,则需要多调用 5 次来填充中间的空间。因此,总的来说,平均每帧需要调用 $6L + 5$ 次 DP 方法,如果使用 Geometry Instancing 技术把每个层次中的所有大块作为一个统一的整体调用一次 DP,则可以把 DP 的次数减少到 $3L + 2$ 。

5.3 Geometry Instancing 技术

Instancing 技术适用于渲染大量几何信息相同、但其他属性不同的物体(即形态相同,位置等信息不同),通过 Instancing 技术可以把多个 DP 调用合并为一次,可以提高系统性能。

Geometry Instancing 的具体方法是先将同一类型的 footprint 属性信息依次保存在一个数组里,然后传给顶点着色器,保存在常量列表中。在创建顶点缓存时,另外增加一个顶点所属实例的索引并赋值,在顶点着色器处理过程中,根据索引从着色器常量中获取该实例的其他属性信息(如位置矩阵和缩放因子等)完成顶点信息的计算。Geometry Instancing 技术通过将不同实例的属性全部存入顶点缓冲区来减少 DP 函数的调用次数,在 DirectX9.0 支持的 GPU 中可以满足顶点缓冲区的要求。地形渲染的算法流程见图 5。

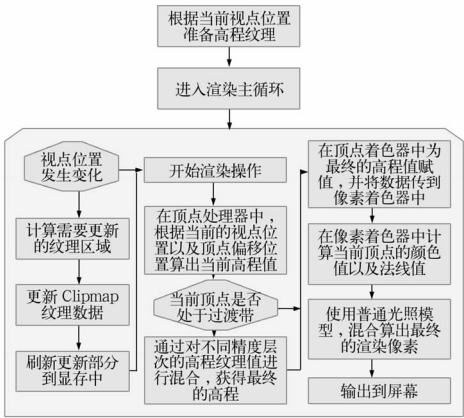


图 5 算法流程图

Fig. 5 Algorithm flow chart

6 实验及分析

选取 $40\,000 \times 40\,000$ 大小的地形数据和影像图纹理进行渲染,采用 Geometry Clipmap 地形结构的大小为 127×127 ,层数 L 为 10,纹理结构的 ClipSize 大

小为 $2\,048 \times 2\,048$,纹理过滤采用 $16\times$ 的各向异性过滤。硬件配置为 Intel Core(TM)2 Duo CPU T7250@2.0GHz,2.0G RAM,Nvidia Geforce 9600GT。实验运行效果如图 6 所示,左为地形网格结构,右为最终渲染的地形。

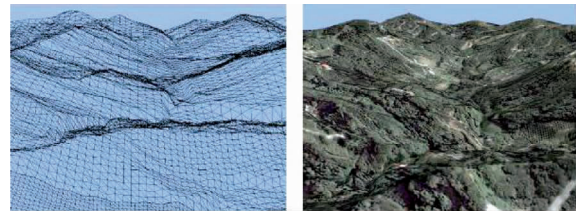


图 6 实验效果
Fig. 6 Rendering results

基于不同视点观察地形,得到不同视点的 FPS、绘制三角形数和 CPU 占用率,如表 1。

表 1 改进算法的渲染结果

视点位置				性能信息	
X	Y	Z	FPS	渲染三 角形数	CPU 占 用率(%)
-3 023	24	2 342	123.3	20 042	4.2
-5 632	3 479	-356 2	123.8	23 402	3.6
3 275	324	-834 5	124.5	22 424	4.6
8 653	2 435	-345 6	124.1	22 344	5.1

由表 1 可以看出,经过扩展后的 Geometry Clipmap 算法相比传统的基于 CPU 的算法,CPU 占用率非常低,当视点变换时,渲染三角形的数目和帧速也很稳定。

7 结语

根据现代图形发展趋势针对 Geometry Clipmap 算法进行了改进,增加了地形数据的管理,采用了循环数据更新方式,并在显存中增加了一个固定的顶点缓冲区,通过引入索引缓冲和 Geometry Instancing

技术减少了渲染次数,进一步提高了渲染速度。本算法还有需要继续完善的内容,例如由于视点变化的随意性导致某些情况下绘制质量发生较大的跳变,需要给出一种基于视点的绘制质量控制方法。

参 考 文 献

1 Lindstrom P and Pascicci V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization[J]. IEEE Trans on Visualization and Computer Graphics,2002,8(3):239-254.

2 Duchaineau M, et al. ROAMing terrain: Real-time optimally adapting Meshes[J]. Proceedings of the IEEE Conference on Visualization, 1997:81-88.

3 Hoppe H. Smooth view-development level-of-detail control and its application to terrain rendering[A]. IEEE Visualization Proc of the Conf on Visualization98[C]. Los Alamitos: IEEE Computer Society, 1998, 124-128.

4 Losasso F and Hoppe H. Geometry Clipmaps: Terrain rendering using nested regular grids[J]. ACM Transactions on Graphics, 2004, 23(3):769-776.

5 Lindstrom P and Pascucci V. Visualization of large terrains made easy[R]. Proceedings of the Conference on Visualization'01, San Diego, 2001. Washington, IEEE Computer Society:363-370.

6 Li Sheng, et al. High performance navigation of very large-scale terrain environment[J]. Journal of Software, 2006, 17(3):535-545.

7 康宁,等. 一种基于图形硬件的海量地形实时可视化算法[J]. 系统仿真学报,2007,19(17):388-392. (Kang Ning, et al. Graphic hardware-based algorithm for real-time visualization of massive terrain dataset[J]. Journal of System Simulation, 2007, 19(17):388-392)

8 张燕燕,等. 飞行模拟器视景系统的设计与实现[J]. 系统仿真学报, 2009, 21(12):3662-3667. (Zhang Yan-yan, et al. Design and implementation of visual simulation system in flight simulator[J]. Journal of System Simulation, 2009, 21(12):3662-3667)