视点相关的动态ROAM算法研究*

任远红 杨克俭
(武汉理工大学计算机科学与技术学院 武汉 430063)

摘 要 三维真实感地形的生成一直是计算机图形学领域中关注的焦点课题。通过系统研究目前比较常用的地形渲染算法,归纳总结出各主要算法的优缺点,重点选择一种适用性比较广的ROAM算法,详细剖析其算法思想,做了一些改进,并运用于实际的三维地形模型渲染中。

关键词 地形渲染 LOD技术 ROAM算法
中图分类号 TP317.4

1 引言

室外场景的实时渲染技术一直是计算机图形学里研究的热点,其应用领域十分广阔,GIS系统、飞行模拟系统、VR系统、视频游戏以及数字地球技术等都离不开室外场景的实时渲染技术。

目前的地形渲染技术主要有两种Voxel和LOD。Voxel(Volumetric Pixel)就是所谓的"体素",它有一个极大的优点就是渲染的时候和场景的大小没有关系,而且由于自带的裁剪功能,所以不会渲染多余的东西,它的复杂度只和需要的视野以及分辨率有关。由于它是一种伪三维技术,灵活性不够成为它非常严重的缺点。

LOD(Level of Detail)是层次细节的简称,不同于Voxel技术,它是一种使用多边形的真正的3D渲染技术。它根据一定的规则来简化物体的细节,可以根据视线的主方向、视线在场景中的停留时间、景物离视点的距离及景物在画面上投影区域的大小等因素,来综合决定景物应该选择的细节层次。在实时交互时,根据观察者当前时刻所处的位置及视线方向,在近距离区域采用高精度模型地形块,而在远处采用低精度地形块,以达到在保证实时图形显示的前提下,较好地协调场景真实感与绘制速度的矛盾,最大程度地提高视觉效果。

在大规模室外场景渲染中,地形数据量是相当大的,如果每一帧都显示处理所有的数据,在现有的硬件条件下是很难实现实时漫游的,所以各种实时地形绘制算法一般采用视点相关的连续细节层次技术和裁减来减少绘制的数据量,以达到实时要求。

2 实时地形绘制算法

实时地形绘制算法根据其生成的网格可以分为基于不规则三角形网格(TIN)的算法和基于规则网格的算法两大类[1]。前者的优点是在同样的误差条件下生成的三角形网格的顶点数目比后者要少得多,而且网格的形状也可以得到优化。其缺点是与CLOD技术比较难结合,处理时需要占用较大内存,计算量较大,灵活性差,效率较低。文献[2]中的网格就是基于TIN的。与基于TIN的算法相比,基于规则网格的方法虽然生成网格中的顶点数目和三角形数目都较多,但是易于裁剪和自顶向下简化,也易于地形跟随和碰撞检测,对动态地形的处理也较方便,所以现在的地形渲染较多的采用基于规则网格的算法。

基于规则网格的算法又主要分为四叉树算法[3]和二叉树算法两类。四叉树的实施简单、有效,但对于裂缝和T型头的处理比较复杂。二叉树算法中比较有代表性的是Duchaineau等人提出的ROAM(实时优化自适应网格)算法[4]。它是基于二元三角树结构的,由于其对于任意给定的分解误差产生最少数量的多边形,而且可以指定确切的帧率和地形网格的精度,使得该算法成为目前使用最广的实时地形绘制算法。下面将重点介绍该算法的思想及其实现方法。

* **收到本文时间**:2006年10月16日
基金项目:国家自然科学基金项目(批准号:60073057)资助。
作者简介:任远红,女,硕士研究生,主要研究方向:图形图象。杨克俭,博士研究生,教授,博士生导师,主要研究领域为虚拟现实、科学计算可视化与计算机仿真。

3 ROAM 算法思想

3.1 数据结构

ROAM 算法之所以能生成最优网格,主要是因为它使用二元三角树来保持三角坐标而不是存储一个巨大的三角形坐标数组来描绘地形结构。这里的每一个小块(Patch)都是一个简单的正二等边三角形。从三角形的顶点到其斜边的中点对其进行分解,可生成两个新的正二等边三角形。这个分解过程是递归的并且可以被子三角形重复直到达到希望的细节等级。ROAM 分割成小方块的速度很快,而且可以动态更新高度图。图 1 给出了二元三角树的最初几个等级。

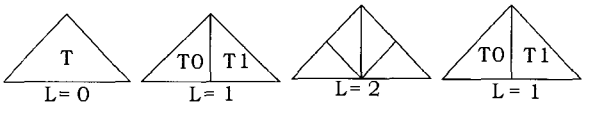

图 1 二元三角树的结构等级(0-3)

在算法中我们用 TriTreeNode 结构来保存二元三角树,同时还追踪 ROAM 所需要的五个最基本的邻接关系,即五个指向其它树节点的指针。其中两个指向自己的儿子节点,另外三个分别指向其邻居节点,具体参见图 2。

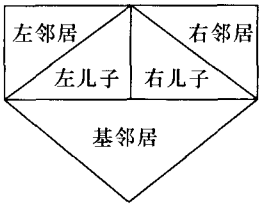

图 2 二元三角树节点的五个指针

TriTreeNode 结构可定义如下:

```
struct TriTreeNode {
    TriTreeNode * LeftChild; // 左儿子
    TriTreeNode * RightChild; // 右儿子
    TriTreeNode * BaseNeighbor; // 基邻居
    TriTreeNode * LeftNeighbor; // 左邻居
    TriTreeNode * RightNeighbor;//右邻居
};
```

3.2 实时网格的生成

在地形漫游过程中,随着视点的移动,ROAM 算法需要在每一帧都重新生成地形结构。二元三角树的生成过程是递归的,它从一个单个的节点(根节点)开始,然后判断该节点是否处于所希望的细节层次,如果未达到此等级,则分解该节点生成两个儿子节点,对于每个儿子节点递归进行上述操作。

我们根据地形表面的粗糙程度和当前节点离视点的距离来决定每个节点的希望细节等级。一般情况下,只要高度图不是动态改变的,每个节点的变差(variance)值是不会变的,它们被预先计算好并存储在一个变差数组中。在实时漫游过程中,通过计算分解误差来决定某个节点是否需要细分以及其分解的深度。

3.2.1 分解误差的确定

节点变差只是另一种描述地形粗糙度的方式。二元三角树中一个节点的局部变差相对来说比较容易计算,因为分解一个节点只是改变了该三角形斜边中点的高度值。变差可以通过该点的插值高度与其在高度图中的实际高度之差来计算,即 Local Variance = abs(centerZ - ((leftZ + rightZ)/2))。树中每个节点的最终变差是其局部变差和其两个儿子节点变差中的最大值,这又是另外一个递归计算的过程。

在每个生成二元三角树的递归步骤中都要进行分解误差的计算。文献[4]中提出了基于嵌套的世界空间范围的分解误差,虽然这个误差计算更加精确,但同时计算速度也相当缓慢。在本文中采用文献[5]中的误差计算方法,即 Split Metric = (Variance * Width * 2) / Distance。其中 Variance 是上述所说的变差,Width 是整个地形网格的宽度,Distance 是当前节点与视点之间的距离。这种分解误差综合考虑了如前所述的地表粗糙度和观察点与节点的距离,计算较简单。

在这里我们引进一个 gFrameVariance 的全局变量,它限定当前帧中所有节点的期望变差的上界。如果条件 Split Metric > gFrameVariance 成立,则当前处理的节点需要分解。该算法中我们还通过动态修改每帧中 gFrameVariance 的值来确保恒定的帧率。

3.2.2 节点的分解

在 ROAM 算法中,共享同一条底边的两个节点三角形构成一个菱形(diamond)。对于当前节点,我们考虑该节点所在三角形的斜边中点,当它的误差大于所给定的阈值时,就从顶点到斜边中点的连线将此三角形一分为二,但是这样简单的分解会造成裂缝。因此,为了避免在分解的过程中会产生裂缝,还必须保证相邻三角形之间的层次等级不能超过 1,可以通过如下简单的规则得以保证:一个节点只有当它和另外一个节点在一个菱形中时才能被分解,这样该节点和它的基邻居同时被分解。当然,如果一个节点的底边是网格边缘的一部分,那么它就没有基邻居,这种情况下就只用分解当前节点。

如果一个节点需要被分解但它又不在菱形中,此时就要强制分解其基邻居。这个强制分解过程

是对网格的递归回溯直到找到一个菱形或边缘三角形。具体步骤如下：每当要对一个节点进行分解，首先检查它是否处于一个菱形中，如果不是，调用另一个对基邻居的分解操作来产生菱形，然后继续最初的分解。对第二个分解的调用同样做相同的检查，如果需要再一次重复此过程。一旦发现某个节点可以被合法的分解，递归就被展开，沿路一直分解下去。如图3，左边灰色的阴影部分是最初需要分解的节点，右边是强制递归分解后的结果。

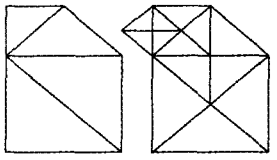

图3 节点的强制分解

利用上述的分解误差来限定节点的分解，其网格渲染结果是导致离视点近的区域有比较高的层次细节，稍远区域的层次细节递减，这比较符合人的视觉特性。通过上述的简单规则可以保证网格中不出现裂缝。

4 ROAM算法的实现

ROAM算法的实现比较简单，核心部分为Landscape，Patch两个类和一个TriTreeNode结构。TriTreeNode结构在前面已经介绍过，下面对这两个类的功能及其一些主要操作进行说明。

Landscape类管理大的地形块，协调块与块之间的沟通。它相当于地形渲染的所有细节的封装体。它主要包括Init(unsigned char *hMap)函数，用来初始化所有的地形块；Tessellate()函数，用来分割地形块以生成近似网格；Render()函数，用来渲染每个地形块，并调整gFrameVariance的值；Reset()函数，用来在每帧渲染的开始重新设置每个地形块，并在需要的时候重新计算变差值。

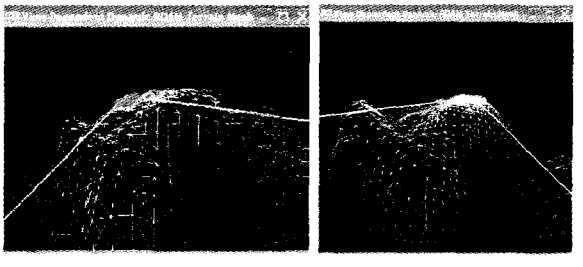

图4 低LOD和高LOD的地形网格

Patch类负责管理地形的每个小块，它也有上述四个函数，用来供Landscape类调用。除此之外，它还有Split(TriTreeNode *tri)函数，对当前节点进行递归分解；RecurseTessellate()函数，用来分割一个小块直到满足一定的细节层次，它供该类的Tessellate()函数调用；RecursRender()函数，用来递归渲染每个叶子节点，供该类的Render()函数调用；RecursComputeVariance()函数，用来计算所有节点的变差；ComputeVariance()函数，调用RecursComputeVariance()函数完成块中各个节点的变差计算；SetVisibility()根据嵌套包围盒算法来判断地形块的可见性，从而实现裁剪。

5 结果

本文详细阐述了ROAM算法的基本思想及其网格生成的具体过程，最后将该算法用VC 6.0和OpenGL在PC机上加以实现，效果图如图4，图中的黄色直线代表视线的方向。可以看出，同一幅图中，离视点近的地形块中有比较多的三角形，因而具有较高的层次细节，层次细节随着离视点的距离的增加而递减；同时起伏程度比较大的区域比平坦的区域拥有较高的层次细节。

由于文中的ROAM算法只用了节点的分解(Split)而丢弃了节点的合并(Merge)，所以每次渲染都要从根节点开始，从最低精度地形块重新实时生成网格，未能很好地利用帧与帧之间的相关性。同时它采用GL_TRIANGLES而不是GL_TRIANGLE_STRIP或GL_TRIANGLE_FAN的渲染方式，使得同一个三角形的顶点被多次送往Vertex Buff-er，最终导致帧渲染率的下降，这都需要在以后的工作中得到改进。

参考文献

[1]卓亚芬，赵友兵，石教英. 实时地形绘制算法综述[J]. 计算机仿真，2005，22(3)：3~7

[2]H Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering[C]. IEEE Visualization 1998：35~42

[3]Lindstrom P, Koller D, Ribarsky W, etal. Real-Time, Continues Level of Detail Render-ing of Height Fields [C]. SIGGRAPH'96, Los An-geles, California, 1996

[4]Mark Duchaineau, Murray Wolinski, David E. Sigeti, Mark C. Miller, Charles Aldrich and Mark B. Mineev-Weinstein. ROAMing Terrain: Real-time, Optimally Adapting Meshes[C]. Proce-edings of the Conference on Visualization 97, 1997, 10：81~88

[5]Turner, B. Real-Time Dynamic Level of Detail Terrain Rendering with ROAM[DB/OL]. URL：http://www.gamasutra.com/features/20000403/turner_01.htm, 2000