改进 LOD 的大规模三维漫游场景简化策略及分割算法①

张志强[1,2], 宋海生[3]

[1](顺德职业技术学院 电子与信息工程系, 佛山 528333)
[2](华中科技大学 计算机科学与技术学院, 武汉 430074)
[3](中国科学院 研究生院, 北京 100049)

摘 要: 提出一种改进 LOD 的大规模三维漫游场景简化策略及分割算法, 首先应用基于最小二乘粗糙度的节点简化策略对复杂场景网格进行简化, 通过改进 Lindstrom 算法, 增加简化准则来解决误差模型精确度下降和粗糙度值变化的问题; 然后对简化后的模型采用多通道切割算法进化分割; 分割后的场景数据, 应用基于误差因子的评价系统来判断; 最后实验, 通过与传统算法分析比较, 证明本文提出的简化策略和分割算法对降低大规模三维场景绘制复杂度, 加快场景实时性显示等方面具有较明显的优势.

关键词: LOD; 最小二乘粗糙度; 简化; 多通道; 分割

Simplified Strategy and Segmentation Algorithm on Modifying LOD for Large-Scale 3D Riding Scenes

ZHANG Zhi-Qiang[1,2], SONG Hai-Sheng[3]

[1](Electron & Information Engineering Department, Shunde Polytechnic, Foshan 528333, China)
[2](School of Computer Science and Technology, Huazhong University of Science & Technology, Wuhan 430074, China)
[3](Graduate School of Academia Sinica, Beijing 100049, China)

Abstract: A simplified strategy and a segmentation algorithm to modify LOD for large-scale 3D riding scenes are presented. The simplified strategy based on the nodes of least square roughness is applied to simplify the complex meshes of scenes. The decreased accuracy of the error model and the variation of roughness values are solved by modifying Lindstrom algorithm and adding rules of simplification. Then the simplified model is segmented by the multi-channel cutting algorithm. The scene data resulted from the segmentation is judged by an evaluation system based on the error factors. Finally, the comparison with the traditional algorithm proves that the simplified strategy and the segmentation algorithm proposed by the paper are more effective to reduce the complexity of the large-scale 3D scene rendering and accelerate the real-time display of scenes.

Key words: LOD;least square roughness; simplify; multi-channel; segment

大规模三维场景漫游技术广泛应用于虚拟现实、游戏娱乐、媒体技术、地理信息等领域. 其研究的热点是如何利用有限的硬件资源来高效实现海量数据的实时绘制, 即解决复杂场景实时渲染的问题. 三维场景的复杂性体现在场景数据的几何结构和形态描述, 场景由于数据量大, 占用内存空间多, 将影响计算机处理速度和视觉效果, 而大部分实时性应用系统(如虚拟现实系统)要求图形显示速度必须跟上视点移动速度, 不能出现显示延时现象[1]. 要实现大规模三维场景实时漫游需取决于两方面: 一是提高计算机的运算速度; 二是通过减小三维目标计算量来增加实时性. 尽管目前计算机的运行速度有很大提高, 但在实时显

① 收稿时间:2013-04-18;收到修改稿时间:2013-06-03

示大型场景时, 尤其在系统要求具有较高的交互帧频率时, 会出现显示滞后现象. 因此简化大规模场景算法和技术的研究具有一定的现实意义.

计算机三维场景的绘制主要以三角形网格的形式呈现, 网格数量与场景的大小相关联. 为保证绘制的效率, 多层次细节(Level Of Detail, 简称 LOD)是研究最多也最行之有效的算法. 该算法通过逐级简化场景表面细节来降低场景的几何复杂度来提高图形绘制效率[2]. 但随着场景规模和数据复杂度的不断增加, 单纯采用 LOD 技术难以满足系统实时性能需求, 因此需要在简化策略和分割算法上进行改进.

国内外针对 LOD 技术进行了深入研究, 并提出了多种简化策略和分割算法. Ddboer[3]提出的 GeoMipMaps 算法将三维场景进行分块处理, 简化了 LOD 的计算量, 分块后的场景更易三角形条带化, 有利于 GPU 进行批处理, 但对相邻场景层次级别的差需小于或等于 1, 且未考虑地势对地表细节的影响; Lintstorm[4]提出通过对场景表面进行递归四叉树剖分生成场景近似网格模型, 但其计算复杂度高, 对输入数据量较敏感, 且处理 LOD 变化时产生的裂缝问题繁琐, 时间耗费量大; Dmitry Brodsky[5]等人提出基于顶点聚类的方法, 该算法根据系统负载能力进行模型剖分, 比串行算法有更大的加速比, 但没考虑不同分辨率的 LOD 模型的拼接, 不宜用于生成多分辨率场景模型; LOSASSO 等[6]提出构建规则嵌套式网格, 配合使用 GPU 进行加速, 使实时绘制大规模场景成为可能, 但该方法没考虑地势起伏的情况, 最后绘制的场景真实感不足; 文献[4]提出基于二叉树的场景网格细分方法生成多分辨率 LOD 模型, 但网格细分与视点无关, 难以随着视点变换来生成连续光滑的三维场景; Duchaineau[7]提出实时优化自适应网格算法, 该算法引入合并/剖分三角形队列, 能较真实地实时绘制三维场景, 但维护队列计算量过大; SCHNEIDER 等[8]提出对场景网格进行条带化的优化方法, 提高了绘制场景的效率, 但该方法受场景网格的大小有限制.

与其他三维物体相比, 场景的绘制由于数据量更大, 表面细节更复杂, 需要设计相关的绘制算法, 以实现对大规模三维场景快速、逼真的实时绘制. 绘制大规模场景需要大量内存空间, 对系统资源消耗较大, 因此, 如何减少内存存储的数据量, 将直接影响三维场景绘制的效率[9]. 本文在现有文献的基础上提出改进 LOD 的大规模三维慢游场景简化策略及分割算法, 首先分析 LOD 简化场景的算法原理, 接着提出基于最小二乘粗糙度的节点简化策略来简化三维场景网络, 然后提出多通道切割算法; 对于分割后的场景数据, 采用基于误差因子的评价系统来判断是否需要继续分割或者已达到最高分辨率; 最后对本文提出的简化笨略和分割算法进行实验, 并与传统的算法进行分析比较, 通过实验证明本文提出的简化策略和分割算法对降低大规模三维场景绘制的复杂度, 提高场景实时性显示速度等方面具有较明显的优势.

1 改进LOD的简化策略

1.1 LOD 简化原理

LOD 技术的核心思想是对场景或场景中的物体, 根据三维可视化的实现原理, 在屏幕上的投影面积由场景或物体的实际面积、距离视点位置及与屏幕的夹角共同决定. 实际面积越小、距离视点越远、与投影平面的夹角越大, 则单位网格在屏幕上的投影面积就越小[10]. 根据人的视觉特征, 可以降低模型的显示分辨率, 而不会对视觉产生太大的影响, 因此在场景绘制前, 根据不同控制误差 δ_i 提前生成多个不同分辨率的简化模型, 绘制时, 根据场景中某点距离视点的位置 d 、允许屏幕误差 p 来计算出实际的最大允许误差 $\delta_{\max}$, 即:

$$\delta_{\max}=(2\times tg\frac{a}{2}\times d\times p)/(\cos\beta\times L\times\lambda) \quad (1)$$

选择 $\delta_i<\delta_{\max}$ 且与 $\delta_{\max}$ 值最接近的简化模型. 当视点移动时, 根据 $\delta_{\max}$ 的值重新选择相应的简化模型进行绘制. 公式(1)满足两个原则: 在指定误差上界情况下, 模型的顶点数最少; 在给定模型顶点个数的基础上, 简化模型与原始模型之间的误差最小.

传统 LOD 层次模型如下图 1 所示, $l_1,l_2\cdots l_{n-1},l_n$ 为约定视点与物体之间的距离.

$LOD_1,LOD_2\cdots LOD_{N-1},LOD_n$ 为随着视距变化而改变粗细程度的模型. 其算法结构描述如下:

① 随着视距从近到远, 分别对物体生成模型 I 、模型 II…模型 N-1、模型 N.

② 对每个模型设定不同视距 $l_1,l_2\cdots l_{n-1},l_n$.

③ *if* $0<$视点距离$\le l_1$, 显示模型 I;

else if $l_1<$视点距离$\le l_2$, 显示模型 II;

……

else if $l_{n-1}\le$视点距离$\le l_n$, 显示模型 N. 图 2 为某

场景应用 LOD 生成的网络图.

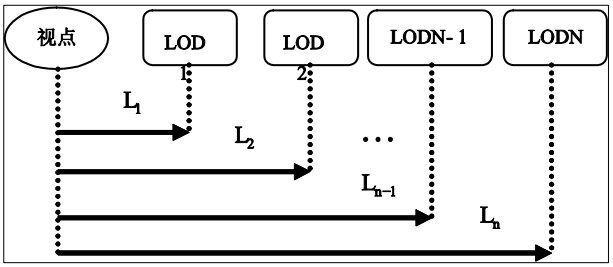


图 1 传统 LOD 层次模型

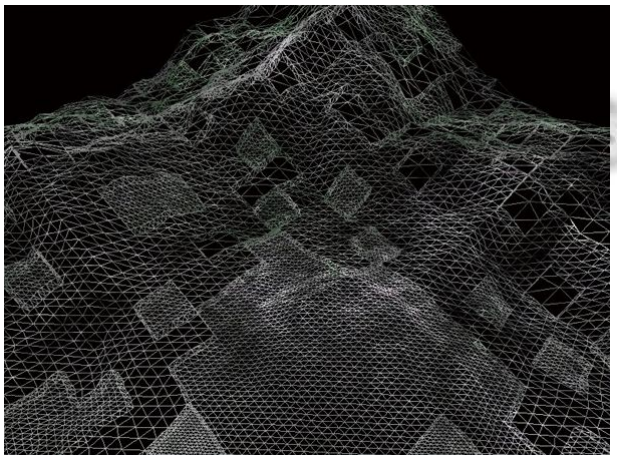


图 2 某场景应用 LOD 生成的网络图

1.2 基于最小二乘粗糙度的节点简化策略

简化准则通常通过误差模型(*Error Model*)来实现. 在三维场景中, 除了最高节点外, 其余节点均与原始数据存在一定误差, 可以根据误差大小来决定该节点是否符合简化准则. *Lindstrom* 提出基于顶点的简化准则(*Vertex based Simplification*). 设某场景块含有 9 个高程点 A~I, δ_B、δ_D、δ_F、δ_H 为 B、D、F、H 点到四边中点之间的误差. *Lindstrom* 的顶点简化准则核心思想是: 节点取舍由 δ_B、δ_D、δ_F、δ_H 投影到屏幕上的大小决定. 如图 3 所示, σ 为节点误差 δ 投影到屏幕之后的大小, θ 为视见向量(顶点到视点之间的向量)与垂线(与水平面垂直的直线)之间的夹角[11]. 则:

$$\sigma = (\delta \times a) / d \times \sin\theta \quad (2)$$

由公式(2)可知, 投影误差 σ 不仅和 δ 相关, 还与顶点和视点之间的距离 d 和视见向量的角度 θ 有关,令 $\sigma_{\max}$ 为 δ_B、δ_D、δ_F、δ_H 中的最大值, 并作为节点误差, 如果 $\sigma_{\max}$ <误差限 τ, 则该节点向上简化, 否则独立绘制 .

Lindstrom 的基于顶点的简化策略符合 LOD 算法与视点位置相关的思想, 但该策略仅考虑到四个边中点的投影误差情况, 而忽略其余顶点(例如子节点的边中点)的误差, 有可能出现子节点的投影误差比父节点大的不合理现象. 因此, 本文在 *Lindstrom* 的简化策略基础上增加如下规定, 当某子节点的误差比父节点大, 则用子节点误差代替父节点误差, 如公式:

$$\sigma = \max(\sigma, \sigma_1, \sigma_2, \sigma_3, \sigma_4) \quad (3)$$

其中 $\sigma_i (i = 1,2,3,4)$ 为子节点误差, 公式(3)的计算与遍历树的顺序相关, 当自下而上遍历树时, 可应用公式(3)计算每个顶点的投影误差; 当自上而下遍历树时, 需要简化误差算法, 即当节点离视点的距离很远时, 节点大小忽略不计, 则对于该节点内部的每个顶点, $a / d\sin(\theta)$ 均相同. 这时, 公式(3)可以简化为:

$$\sigma = \max(\delta, \delta_1, \delta_2, \delta_3, \delta_4) \times a / (d \times \sin(\theta)) \quad (4)$$

经过简化后的误差算法适用于自上而下的遍历原则. 但其误差模型精确度下降, 同时, 基于顶点的简化算法所计算出来的误差模型只反映单个顶点的误差值, 而不是整个场景块的整体误差. 因此, 本文在该算法的基础上提出一种更简便的粗糙度求取算法: 最小二乘法. 通过以下原则判断某节点应该往下细分或往上简化:

① 当某节点离视点较近, 或其代表的场景块较粗糙, 则对该其往下细分, 分裂为四个子节点;

② 当某节点离视点较远, 或其代表的场景块较平滑, 则对其往上简化;

③ 当某节点与视点距离适中, 或不太粗糙也不太平滑, 则独立绘制.

如图 4 所示, 某三维场景块中 A、B、C、D 为 O 的四个子节点, 根据以上简化原则, 假设 A、B 节点较平滑, C、D 节点粗糙度适中. 则只绘制 C、D 节点, 每个节点用八个三角形表示; A、B 节点各用 2 个三角形表示即可.

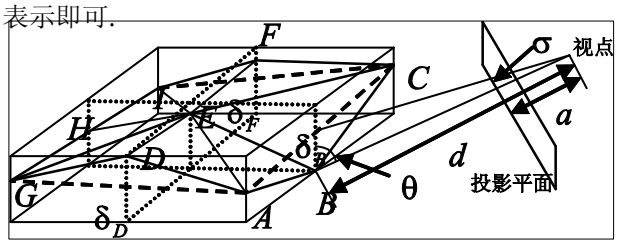


图 3 Lindstrom 的顶点简化示意图

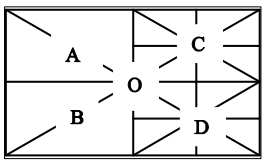


图 4　节点简化图

采用最小二乘法建模, 粗糙度以常量值表示, 该常量值不随视点移动而改变, 是一种基于块的处理算法. 最小二乘法对节点进行评估时需要考虑 3 种因素: 场景块的大小, 场景块与视点之间的距离以及场景块的粗糙度. 用 $Z_{x,y}$ 表示场景块中某顶点的高度, 由于场景块为方形区域, 满足条件 $x_{\min}<x<x_{\max}$, $y_{\min}<y<y_{\max}$, $(x,y$ 取整数值). 则场景块的平均高度 $\overline{Z}$ 可以式(5)表示:

$$\overline{Z}=1/((x_{\max}-x_{\min})(y_{\max}-y_{\min}))\sum_{x=x_{\min}}^{x_{\max}}\sum_{y=y_{\min}}^{y_{\max}}Z_{x,y} \quad (5)$$

采用最小二乘法求取粗糙度的公式如下:

$$\mu=1/((x_{\max}-x_{\min})(y_{\max}-y_{\min}))\sqrt{\sum_{x=x_{\min}}^{x_{\max}}\sum_{y=y_{\min}}^{y_{\max}}(Z_{x,y}-\overline{z})^2} \quad (6)$$

场景块边长用 d 表示, 场景块中点与视点之间的距离用 l 表示. 基于块的简化准则可以表示为:

$$(\mu\times\sin\theta)/(l\times d)<\tau \quad (7)$$

其中 $l=y_{\max}-y_{\min}$.

公式(7)中 θ 为视见向量和垂线之间的夹角, τ 为误差限. 当条件 $l=y_{\max}-y_{\min}$ 成立时, 则独立绘制该节点, 否则继续遍历其四个子节点. 由于场景块的粗糙度为常值, 它不随视点的移动而改变. 因而计算粗糙度的过程可在数据预处理阶段完成, 不需再更新, 从而大大提高算法运算的速度, 图 5、图 6 分别为运用最小二乘法对某山脉场景绘制的线框图和效果图.

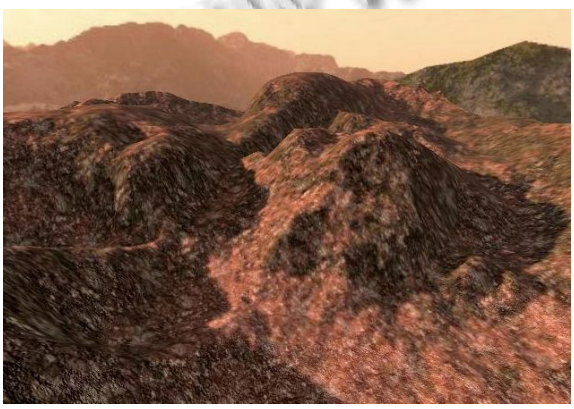


图 5　运用最小二乘法对某山脉场景绘制的线框图

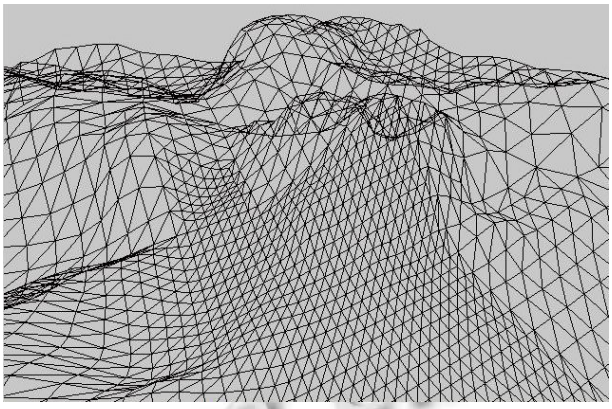


图 6　效果图

2　分割算法

2.1　多通道分割算法

在场景调度过程中, 仅依靠简化算法难以满足海量数据的实时绘制, 需要对底层分辨率较高的三维场景做切割和优化[12]. 在实时性应用系统(如虚拟现实系统)中, 场景分割是把主机上大型视景体分割为多个视景体的过程, 其分割原则为分割后的视景体在某个方向上拼接后正好等于分割前的视景体, 以保证场景分割前后一致.

本文采用多通道分割算法对大型三维场景进行分割, 这是因为近年来, 计算机硬件在计算能力和图形处理能力方面发展迅速, 基于高性能的多通道系统架构不但可以大幅度提高图形处理能力, 又能有效降低成本, 应用前景广泛[13]. 本文算法的核心思想是将一个型三维视景分多个小场景, 每个场景彼此连续, 然后对小场景分别独立渲染, 最后重组为一幅完整、流畅的画面. 需要解决两个关键问题, 分割后的场景会出现变形现象, 如果保持场景并接后与原来一致; 需要设计一个有效的评价系统来判断场景是否需要继续分割. 下面的该算法的程序描述.

步骤 1: 首先定义主机从机互送的数据结构(如下图 7 所示), 定义主机视景体函数为 *gluperspective*, *gluperspective* 包括 4 个参数, 视点包括 9 个参数, 代码如下:

```
typedef struct    //定义视景体
{double, fovy, aspect, znear, zfar;}VIEW _VOLUME;
    typedef struct    //定义摄像机
{double,evex,eyey,eyez,centerx,centery,centerz,upx,upy,upz;}
    CAMERA_ PARA;
```

typedef struct //定义需传输场景的控制参数

{*VIEW _VOLUME viewVolume*;*cameraPara*;}

SCENE _PARA

步骤 2: 当从机接收到 *SCENE _PARA* 场景控制参数时, 根据 *viewVolume* 参数计算视景体, 具体解算过程如下:

① 对参数 *fovy*, *aspect*, *znear*, *zfar* 用空间立体几何学相关公式转换为 *glFrustum* 所需要的 *left*, *right*, *bottom*, *top*, *near*, *far* 6 个参数, 作为各分机视景体拼接后的视景体参数:

② 根据所计算的 6 个参数与本机在系统中所处的行、列号进行运算, 设从机中第 i 行、 j 列的视景体参数为 $left(i,j)$, $right(i,j)$, $bottom(i,j)$, $top(i,j)$, $near(i,j)$, $far(i,j)$, 分机总行列数分别为 m、n, 其中 $i=0,1,2,\cdots,(m-1)$, $j=0,1,2,\cdots,(n-1)$, 则

$left(i,j)=left+j\times(right-left)/n$;

$right(i,j)=left(i,j)+(right-left)/n$;

$top(i,j)=topt+i\times(bottom-topt)/m$;

$bottom(i,j)=top(i,j)+(bottom-topt)/m$;

$near(i,j)=near$; $far(i,j)=far$

通过以上程序得到本机视景体参数, 如图 8 所示.

步骤 3: 通过以上步骤分割得到的视景体没有考虑场景变形问题, 要解决视景体分割前后不一致的问题, 在步骤 2 计算完后, 加入以下判断代码.

if $(n>m)$; $top*=m/n$; $bottom=-top$

if $(n\le m)$; $right*=n/m$; $left=-right$

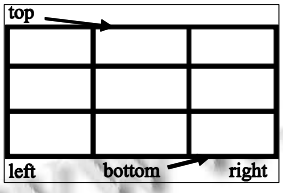


图 7 视景体分割计算

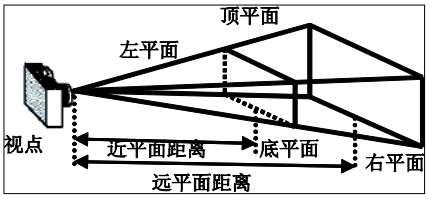


图 8 函数 gluPerspective()指定的透视投影景体

由于分割后的视景体关于原点对称, 因此通过上式规定 $bottom=-top$, $left=-right$, 限制视场在水平或者垂直方向的大小, 以保持场景分割前后形状的一致性.

2.2 评价系统

裁剪后的场景数据, 需要对视域内部节点做进一步评价, 以判断其是否需要继续分割. 其核心思想就是判断视点到节点中心的距离, 根据人的视觉特征, 由于近处物体看起来比较清楚, 需要用较高分辨率表示; 而远处物体看起来比较模糊. 可用较低分辨率表示[14]. 考虑到屏幕允许误差, 设定边长为 D 的节点在投影平面上的长度为, 的值为像素数, 则判断公式如下:

$$\tau=(L\times\lambda\times D)/(2\times\tan\frac{\theta}{2}\times l) \tag{8}$$

公式(8)中, λ 为场景物体在三维空间中单位长度的平面投影像素数, θ 为视点到投影平面的视角, L 为投影平面的长度, l 为视点到某节点中心的距离. 设定屏幕允许的投影误差为 ε, 当 l 的值足够小, 其在屏幕上的误差可以大于 ε, 则需要对该节点进行分割, 否则不需对该节点进行分割. 对顶点的评价不单要考虑视相关因子, 还需考虑场景地表的粗糙度因子. 因为三维场景起伏程度越高, 需要描绘的层次精度就越高. 场景粗糙度可用节点的静态误差表示. 当三维场景中的某节点中心在渲染时, 需连接四个角点, 所以对节点处分割的评价就变为对四个边点 $(dh1,dh2,dh3,dh4)$ 和四个子节点 $(dh5,dh5,dh7,dh8)$ 被分割和不需要分割时所产生 8 个误差值的评价, 如图 9 所示. 取这 8 个误差的最大值作为静态误差度量, 设定某一阈值 $C1$, 下面需要对静态误差进行控制, 当误差小于 $C1$ 时, 不分割节点, 否则分割节点:

$$E(C_j)=MAX(dh_i)<C1 \quad (i=1,2,\cdots,8) \tag{9}$$

公式(9)综合考虑了与当前节点相关的各个误差情况, 这样的优点是使得三维场景中局部较大的误差可以向上层传递, 体现自顶向下逐渐细分的原则, 从而可以很好地体现局部细节. 同时, $E(C_j)$ 的计算在预处理阶段就完成, 大大提高算法实时运行的效率. 根据公式(8)和公式(9)可以建立以下综合评价系统:

$$e(C_j)=(L\times\lambda\times D\times E(C_j))/(2\times\tan\frac{\theta}{2}\times\varepsilon\times l) \tag{10}$$

根据公式(10)建立的综合评价系统, 对三维场景数据作进一步简化, 并随着视点的移动实时调节场景各处的分辨率, 从而大大提高了三维场景实时性显示

的效率.

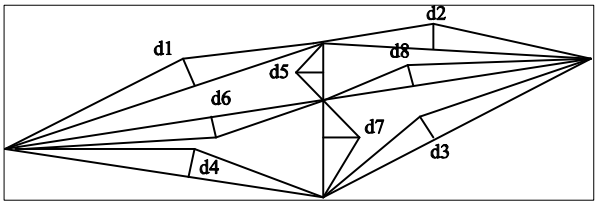


图 9　节点粗糙度信息

2.3 场景分割的优化

使用多通道分割算法的优点是比传统的场景分割算法具有更快的处理速度, 但由于应用了主从机架构, 需要使得各个从机生成的视景在无缝拼接时, 每个从节点要同步显示同一场景. 因此, 在三维场景实时显示时要求相互之间画面延迟要控制在一定范围内, 这就需要在主、从节点之间提供可靠快速的通信. 其核心思想如下:

(1) 多通道分割的系统架构中, 在实时显示大型三维场景时, 主节点发出到各个从节点的每一帧都包括视景体和视点参数, 而在实际应用中, 主机视景体的所有参数对从机来说并不是每帧都需要, 只有当主机窗口大小发生改变时, 其视景体参数才发生变化, 因此, 可以把 *VIEW_VOLUME* 型变量从 *SCENE_PARA* 结构中取出, 仅当主机窗口大小发生改变时, 才向从机发送 *VIEW_VOLUME* 类型的视景体参数, 从而有效降低多通道数据的传输量.

(2) 另外, 仅当主机视点发生改变时, 才需要向从机发送 *CAMERA_PARA* 类型的视点参数. 这样就使得每帧通过多通道传输的参数个数由原来的 13 个下降为平均不超过 9 个.

(3) 当从机接收到 *VIEW_VOLUME* 参数后, 需要计算其透视投影变换矩阵, 设由视景的 4 个参数 *fovy*, *aspect*, *znear*, *zfar* 转换为 *glFrustum* 的 6 个参数 $left=l$、$right=r$、$bottom=b$、$top=t$、$near=n$、$far=f$, 则相应的透视投影矩阵为:

$$\begin{bmatrix} \frac{2n}{r-1} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{r+b}{r-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

3　实验

由于具有山脉的场景是大型三维游戏和虚拟现实等领域中出现较多的自然场景, 因此, 本文用具有山脊滑坡的大型场景进行实验, 地形大小为 2049*2049 像素, 原始网格点数为 1847416 个, 直接生成的三角形个数约为 3697152 个, 如此庞大的三角形数目, 将占用大量系统资源. 实验采用 4 台 PC 机(1 台主机和 3 台从机)做多通道视景仿真试验, 其中主机硬件配置为 CPU Intel Core i5 2550, 2GB 内存, 3 块 1.5T 7200 转硬盘, NVIDIA GeForce GTX 560 显卡; 从机硬件配置为 CPU Intel Core i5 3220, 内存 1G, 显卡 NVIDIA Geforce GTS450; 采用 100M 网线, DLink 交换机, 网络传输使用 UDP 通信方式. 主从机软件配置中操作系统均为 Windows XP, 屏幕分辨率为 1920×1080, 编程环境是在.NET 架构下, 使用 Visual C#编程, 通过调用 OpenGL API 函数库进行仿真. 实验步骤如下:

步骤 1: 首先使用本文算法对场景进行简化, 由于模型简化是一个比较耗时的过程, 本实验对一个拥有 1847416 个顶点的复杂场景使用基于最小二乘粗糙度的节点简化策略对网络顶点进行简化, 简化前场景的三角形个数为 3697152 个, 简化后三角形个数减少为 749430 个, 利用本文算法, 网格的简化率达到 77.9%, 平均帧速为 59.3F/S, 达到了实时显示的要求. 下图 10 为对实验场景应用基于最小二乘粗糙度的节点简化策略生成的纹理效果图和网络图.

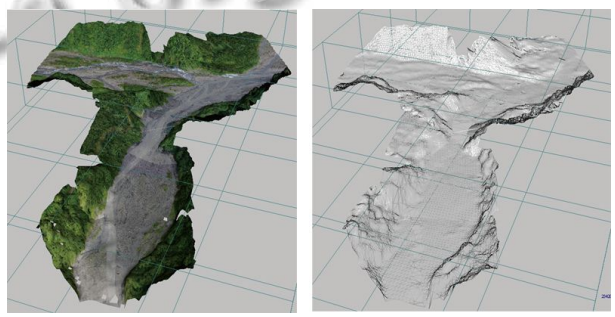


图 10　实现场景带纹理效果图和网络图

实验中, 未简化时的地形显示帧速率约为 2.8F/S, 经基于最小二乘粗糙度的节点简化策略生成的多分辨率 LOD 场景, 漫游时平均帧速率可达到 60F/S. 从上图可以看出, 当模型网格被大量简化时, 模型特征仍能基本保持. 但在交互性实时显示时, 场景的显示速

度必须跟上用户的操作速度，单靠简化算法难以保证场景实时显示的流畅性，因此需要进行步骤 2，对简化后的场景作进一步分割.

步骤 2: 应用多通道分割算法对场景进行分割. 设场景的渲染速率为 V，主节点与从节点之间的通信时间为 $T1$，从节点的场景渲染时间为 $T2$，从节点个数为 n，则

$$V=1/(2nT1+T2) \quad (11)$$

由于场景分割的计算公式都是一些简单的数学运算，并且为降低主机负担及网络数据传输量，其计算在从机进行，其消耗的 CPU 时间可忽略不计. 由公式(11)可以看出，从机场景的渲染速率与从节点的个数呈反比例关系，主要取决于网络传输速度及从节点个数. 限于试验环境，网络设置只采用 100M 网线. 在实际应用中，主从节点可采用高档图形机，网络采用千兆以太网以使系统性能进一步提高.

系统运行后，经过快速的视见体裁剪，有效地减少需要绘制的三角形数量，明显提高帧频率. 同时，根据视点的变化相应地增加和减少细节层次，实现地形的多分辨率显示. 算法运行效果良好，画面十分流畅，能够达到 200F/S 左右的帧频率. 图 11 为实验场景的四通道视景仿真的渲染截图.

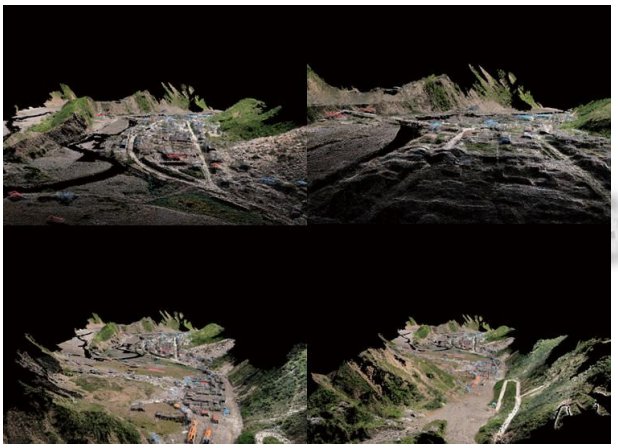
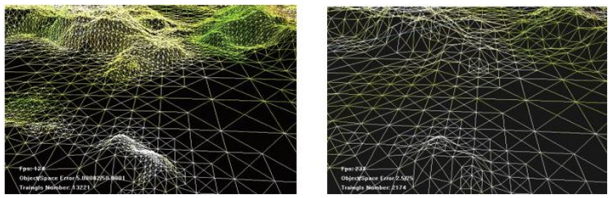


图 11 四通道视景仿真的渲染截图

步骤 3: 对实验结果分析和对比. 在实验中，本文使用一个参数 *radio* 来控制模型的简化程度，*radio* 为简化后模型的顶点数量与简化前顶点数量的比值. 下表 1 是对实验场景做简化计算时分别取 *radio* 为 0.5 和 0.25 所得的模型及各数值结果，图 12 为 *radio* 分别为 0.5 和 0.25 时的渲染网络图.



Radio=0.5 *Radio*＝0.25

图 12 渲染网络图

表 1 某场景简化结果

	原始模型	*Radio*=0.5	*Radio*=0.25
顶点数	1847416	832513	272658
面片数	3697152	218312	90889
简化时间(ms)		4831	6529

为了更好地验证算法的有效性，实验分别采用本文提出的改进算法与传统四叉树算法、GeoMipMaps 算法以及无 LOD 渲染方法进行比较，场景采样点数统一为 512*512，随机抽样其中的 30 次，比较指标分别为绘制帧率/(帧)和实时渲染效果，比较结果见下表 2. 由表 2 可见，本文提出的改进算法生成场景的实时性速度最快，为 187F/S; GeoMipMaps 算法次之; 传统四叉树算法较慢; 当然，无 LOD 的实时性渲染最慢，为 27F/S，特别随着生成表面规模增大时，本文算法的优势更为明显.

表 2 三种简化方法对比

方法	采样点数	绘制帧率/（帧）	显示效果
无 LOD	512*512	27	好
四叉树	512*512	74	较好
GeoMipMaps	512*512	81	一般
本文方法	512*512	187	好

综合以上实验结果，文中提出的简化策略和分割算法，能快速减少场景中需要绘制的三角形数量，在交互式漫游中能够保持较高的帧频率和系统响应，并且有效地控制了内存等系统关键资源的使用量. 该算法加入到仿真平台中，获得十分流畅的仿真画面. 与传统的简化和分割算法相比较具有较明显的优势.

4 结语

本文提出的改进 LOD 的大规模三维慢游场景简化策略及分割算法能快速生成逼真的三维场景，具有数据裁剪速度快，效率高; 内存消耗低，实时渲染速度快; 算法简单，易于实现，漫游画面流畅的优点. 通过对实验结果分析，该算法相对传统的简化和分割算

法对运行时间和空间的要求较小, 完全满足用户对场景漫游实时性交互的要求. 将来的主要工作有: 将场景 LOD 判断及裂缝的消除等操作转移到 GPU 端进行; 研究动态场景的绘制技术, 用以表现动态变化的场景.

参考文献

1 Jin HL, Lu XP, Liu HJ. View dependent fast real-time generating algorithm for largescale terrain. Procedia Earth and Planetary Science, 2009, (1): 1147−1151.

2 Zhong DH, Liu J, Li MC. NURBS recon- struction of digital terrain for hydropowerengineering based on TIN model. Progress inNatural Science, 2008, (18): 1409−1415.

3 Kennelly PJ. Terrain maps displaying hill-shading with curvature. Geomorphology, 2008, 102(3): 567−577.

4 孔川,罗大庸刑用动态多分辨率 LOD 技术的地形简化研究. 计算机工程与应用,2010,46(27):32−41.

5 杨硕磊,郝爱民,王莉莉.运用矩阵结构的可并行地形层次细节算法.计算机辅助设计与图形学学报,2011,23(2):276−283.

6 Francisco RM, Zhang Q, Reid JF. Stereovision three-dimensional terrain maps for precision agriculture. Computers and Electronics in Agriculture, 2008, 60(2): 133−143.

7 张淑军,陈芳,周忠.基于四叉树剖分的 L0D 地形绘制算法. 系统仿真学报,2008,20(z1):25−32.

8 罗景馨,唐珊.基于改进四叉树分割和结点存储的 LOD 算法.计算机工程,2009,35(20):202−204.

9 李白云,赵春霞.GPU 实时构建四叉树的快速地形渲染算法. 计算机辅助设计与图形学学报,2010,22(12):2259−2264.

10 Chen H, Long AQ, Peng YH. Building panoramas from photographs taken with an uncalibrated hand-held camera. Chinese Journal of Computers, 2009, 32(2): 328−335.

11 郎兵.复杂场景中海量外存地形模型的实时绘制算法.系统仿真学报,2009,21(20):6510−6514.

12 李成名,王继周,马照事.数字城市三维地理空间框架原理与方法.北京:科学出版社,2008.

13 张和平.RFID 在供应链物流管理中的应用.中国市场, 2007,(41):104−105.

14 Genske DD, Heinrich K. A knowledge—based fuzzy expert system to analyze degraded terrain. Expert Systems with Applications, 2009, 36(2): 2459−2472.

(上接第 170 页)

Biology and Biomedicine(ACM-BCB). 2010. 170−179.

9 Leung HCM, Yiu SM, Yang B. A robust and accurate binning algorithm for metagenomic sequences with arbitrary species abundance ratio. Bioinformatics, 2011, 27: 1489−1495.

10 Chatterji S, Yamazaki I, Bai Z. Compostbin: a DNA composition-based algorithm for binning environmental shotgun reads. Proc. of the 12th annual international conference (RECOMB'08). Springer. 2008. 17−28.

11 Tanaseichuk O, Borneman J, Jiang T. Separating metagenometic short reads into genomes via clustering. Proc. of the 11th Algorithms in Bioinformatics.(WABI). 2011. 298−313.

12 Zhou F. Barcodes for genomes and applications. BMC Bioinformatics, 2008, 9(1): 546.

13 Frey BJ, Dueck D. Clustering by passing messages between data points. Science, February 2007, 315: 972−976.

14 王开军,张军英,李丹.自适应仿射传播聚类.自动化学报, 2007,12(33):1242−1245.

15 许文竹,徐立鸿.基于仿射传播聚类的自适应关键帧提取. 计算机科学,2010,12(1):268−270.

16 Richter DC. MetaSim:a sequencing simulator for genomics and metagenomics, PLoS ONE,3,pp.e3373,2008.

17 ftp://ftp.ncbi.nih.gov/ ftp://www.ncbi.nlm.nih.gov/.