DOI:10.13203/j.whugis20140850 文章编号:1671-8860(2017)03-0402-06

一种基于 GPU Tessellation 的地形无缝绘制算法

董路明[1] 张 斌[1] 赵学胜[1]

1 中国矿业大学(北京) 地球科学与测绘工程学院,北京,100083

摘 要:针对传统多分辨率地形绘制算法构网速度慢、T 型裂缝不易处理等问题,提出了一种 GPU(graphic processing unit)构网的地形无缝绘制算法。首先,引入实时网格细分(tessellation)技术,将地形构网分为 CPU 粗粒度 Tile 网格构建和 GPU 细粒度 Patch 网格细分两个阶段;然后,Tile 网格采用基于视距的细节层次模型进行 LoD 层次选择,Patch 采用基于屏幕空间投影误差的细节层次模型完成网格细分,兼顾了视距和地形粗糙度对地形绘制的影响,实现了地形细节层次的自适应选择;最后,应用 C++语言和 DirectX 11 工具,设计开发了相应的可视化实验系统。实验结果表明,该方法实现了多分辨率地形的自适应无缝表达,保证了地形网格的连续性;并通过合理平衡 CPU-GPU 负担,显著提升了地形渲染效率。

关键词:地形渲染;GPU;曲面细分;裂缝消除

中图法分类号:P208; TP391 **文献标志码**:A

随着 GPU(graphic processing unit)图形处理器处理能力的飞速发展和可编程图形管线的出现,基于 GPU 的海量地形 LoD 绘制算法已逐渐替代传统的 CPU 构网算法,成为进一步提高大规模地形绘制效率的新手段[1,2]。其中,利用 Tessellation 技术的地形渲染方法[3-8]可以对粗糙的地形网格进行插值产生新的顶点,在显著提升地形模型几何精度的同时减少了数据传输的压力,已成为目前基于 GPU 地形渲染方法的主要研究热点。现有的 GPU Tessellation 地形绘制算法,是通过屏幕空间投影误差进行原始粗糙网格和细分图元细节层次选择,完成多分辨率地形构建,实现了地形的高效绘制。在多分辨率地形可视化表达中,相邻不同分辨率网格之间不可避免地产生了裂缝。而上述算法是在原始粗糙网格间采用最简单的"垂直边缘法"[9]消除裂缝,即通过在地形块边界处人为填充垂面,从视觉上掩盖了裂缝。而相邻不同分辨率地形块共享顶点仍然位于不同高度,地形块邻接处具有不同的边界,裂缝问题未从本质上得到解决,导致地形块边缘处边或三角形空间关系难以表达,空间分析难以实现。该算法仅能满足海量地形在视觉上的可视化表达,很难为城乡规划、大型工程设计、军事模拟等领域,提供可视化分析和决策支持[10];同时,该算法在预处理阶段,必须提前计算好不同格网层次间的最大空间误差,这样在数据实时更新时 CPU 需要负担较重的预处理任务,直接影响了地形实时渲染效率。

本文在 GPU Tessellation 技术基础上,提出了一种 GPU 构网的地形无缝绘制算法。结合 CPU 和 GPU 各自的特点,通过合理控制相邻不同分辨率网格的细分等级,确保地形块间连续过渡。在保证地形高效绘制的同时,实现多分辨率地形的无缝可视化表达,为基于海量地形的可视化分析与决策提供保障。

1 GPU Tessellation 渲染原理

Tessellation 是一种利用 GPU 硬件加速,将多边形分解成更加细小的碎片以提升几何逼真度的方法[11-14]。细分的基本图元可以是四边形、三角形或者线段。置换纹理映射技术[15]通过对高度图进行顶点采样,将高程纹理映射到网格顶点上,实现平面网格到真实地形网格的转变。图 1 为基于 GPU Tessellation 的地形渲染流程。其中,曲面细分控制阶段接收网格的控制点作为输入数据,输出必要的细分因子和控制点信息;曲面细分阶段根据细分因子将图元细分成更小的对

收稿日期:2015-03-08

项目资助:国家自然科学基金(41171306, 41171304)。

第一作者:董路明,硕士,研究方向为三维地形可视化。776459764@qq.com.

象,输出细分后新生顶点的标准化纹理 uv 坐标;曲面细分计算阶段根据控制点信息和新生顶点的 uv 坐标计算出细分后网格各点的位置信息,利用置换纹理映射技术,对高度图进行采样,将网格各点位置信息中的高程值调整为真实世界中的高程。通过上述 3 个阶段,CPU 中一个简单的四边形网格通过 GPU Tessellation 细分成为一张复杂的地形格网。最后,在像素着色器中采样贴图纹理。

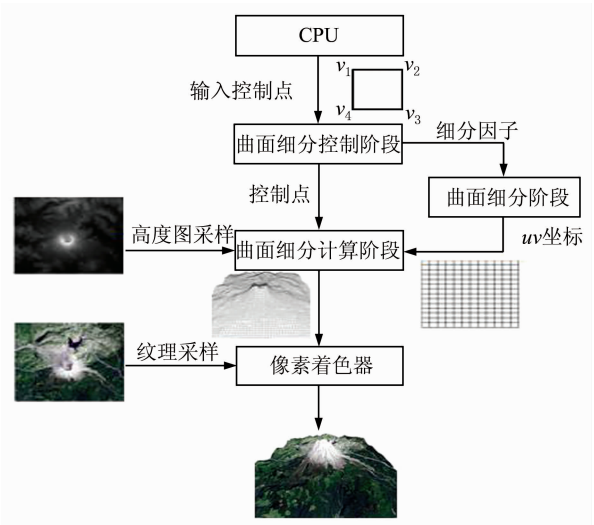

图 1　GPU Tessellation 地形渲染流程

Fig. 1　Terrain Rendering Procedure with GPU Tessellation

本文以四边形作为曲面细分的基本单元,细分因子需要 4 个边细分因子(Edge)和两个内部细分因子(Inner),分别对应于四边形 4 条边和水平、竖直方向的细分程度。图 2 为在不同细分因子作用下,四边形最终的细分效果。从原理上看,基于 GPU Tessellation 的地形渲染算法能够在输入低分辨率网格的基础上,绘制出传统地形渲染算法输入高分辨率地形网格时输出的渲染效果,并且大幅度节约内存开销,提升地形渲染效率。

2　地形数据组织结构

由于当前计算机硬件条件的限制,对于大规模的栅格数据,在实际应用中不可能一次性将所有数据载入计算机内存。因此,需要采用数据分割技术将高程和纹理数据分层分块,构建数据金字塔[16]和地形块四叉树结构[17]如图 3 所示。为了确保地形块间无缝链接,同一层次相邻数据块间共享边界纹理数据。规定每一个满足绘制条件

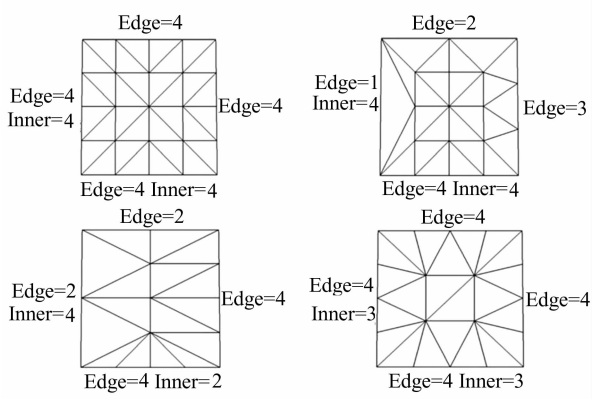

图 2　不同细分因子细分效果

Fig. 2　Effects of Different Tessellation Factors

的叶子节点称为一个 Tile,Tile 中存储着 4 个顶点的位置信息和对应的高度图、纹理图,地形渲染时将这些数据一同传入 GPU 中。本文将高程数据存储为 16 位浮点精度的 DDS[18]格式高度图,有效地降低了纹理实时传输对地形渲染效率的影响。

当前,GPU 绘制函数 DrawPrimitive 的调用次数仍然是限制地形绘制效率的重要因素,每个 Tile 需要调用一次绘制函数。本文将 Tile 均匀分割为 $2^m \times 2^m$ 个 Patch(图 3),Patch 是曲面细分的基本图元。每个 Patch 仅需要存储其在 Tile 中的行列号,就可以精确定位其位置。细分因子的取值受限于 1～64,Patch 最大细分程度下会形成 65×65 个顶点的网格。一个 Patch 对应的高度图像素个数如果超过 65×65 将会产生数据冗余,如果少于 65×65,将不足以表达最高的细分层次。本文将 Patch 对应的高度图像素个数取固定值 65×65,Tile 对应的高度图分辨率相应取值 $(2^{m+6}+1)\times(2^{m+6}+1)$。$m$ 值越大,Tile 分辨率越高,地形渲染时需要的 Tile 个数越少,传至 GPU 中的高度图个数也会相应减少。文献[19]对纹理大小对地形绘制效率的影响进行了深入研究,本文将 m 取经验值 3,既有效减少了屏幕中 Tile 的个数,又避免了 m 值过大时实时传输大纹理导致的帧率震荡现象。因此,Tile 以 8×8 个 Patch 的平面网格进行组织,高度图数据金字塔中每块数据的分辨率是 513 像素×513 像素。

3　基于 Tessellation 的 LoD 地形无缝绘制

3.1　Tile LoD 选择标准

为了尽可能降低 CPU 的负担,本文算法中 CPU 阶段不涉及任何高程数据的操作。所有

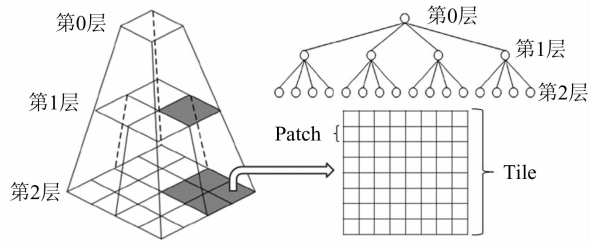

图3 数据金字塔模型和地形四叉树结构

Fig. 3 Pyramid Model and Terrain Quad-tree

Tile均以平面网格的形式传入GPU,高程数据的获取全部在GPU中通过纹理采样方式获得。因此,传统的基于细分前后屏幕空间投影误差标准不适用于本文算法的CPU阶段。Tile的LoD选择属于粗粒度的层次选择,本文使用简单的基于视距的细节层次模型,距离视点越近,地形细节越丰富。其标准如式(1)所示:

$$\frac{l}{d} < C \tag{1}$$

式中,参数 l 表示世界坐标系下地形块的边长;d 表示视点到Tile平面网格中心点的距离;C 是一个可调控的阈值,C 值越小,Tile的LoD层次越高,屏幕中实时渲染的Tile个数越多。

地形渲染时,从根节点开始依次进行判断。如果Tile满足式(1)且该节点位于视锥体内,则将该节点加入渲染列表;如果节点位于视锥体外,则舍去该节点;如果节点位于视锥体内且不满足式(1),则细分该节点,并依次对4个子节点进行递归判断。最终,所有需要绘制的Tile共同构成了一个渲染列表。为了解决相邻Tile之间LoD不一致产生的裂缝问题,每个Tile除了存储着本块需要的数据,还存储着与之相邻的地形块和当前块LoD层次的最大差值。本文用4个整数来表示地形块上、下、左、右4个方向的邻接信息,正值表示相邻地形块LoD层次高于当前地形块,负值表示相邻地形块LoD层次低于当前地形块,0表示相邻地形块和当前地形块LoD层次相同或相邻地形块不存在。以图4中地形块 A 为例,{0,1,2,−1}即为 A 上、下、左、右四个方向需要存储的邻接信息。对于相邻地形块LoD层次低于当前块的情况(如 B),还要额外存储 B 的高度图纹理信息,以确保GPU纹理采样时 A、B 相邻边界各点采样的高程数据相同。本文Tile LoD层次选择标准构建的粗粒度网格邻接关系简单,Tile LoD层次随着视点到Tile距离的增大逐渐降低,相邻Tile LoD层次差一般不超过2,邻接关系处理并不会对地形绘制效率产生明显影响。

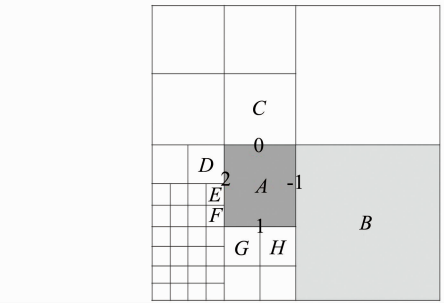

图4 Tile邻接关系示意图

Fig. 4 Adjacency Relationship Between Tiles

3.2 Patch多分辨率无缝表达

Patch的处理是地形多分辨率无缝表达的核心阶段,Patch细分因子的不同实现了地形的多分辨率,Patch相邻边具有相同的细分因子,确保了地形的无缝表达。在多分辨率地形的绘制过程中,仅以视距来确定细分等级是不完善的,还要综合考虑地形粗糙度的影响,地形越崎岖,分辨率应该越高。地形粗糙度的衡量标准有很多种,本文以块内所有顶点的最大高程差表示Patch块空间误差的最大值,并将其投影到屏幕上,将屏幕空间投影误差作为Patch的细分标准。

3.2.1 Patch最大高程差

每个Patch对应于高度图中的65像素×65像素,在曲面细分控制阶段对其按$(2^n+1)\times(2^n+1)$个像素进行均等采样。n 值越大,最大高程差结果越精确,GPU采样时间也会相应增加。将Patch的4条边和内部分开计算,边的空间误差最大值取边上采样点的高程最大差值,内部空间误差最大值取内部采样点的高程最大差值,最大高程差 $\xi_{\max}=h_{\max}-h_{\min}$。表1给出 n 取不同值时,对1 024个Patch进行采样GPU需要消耗的时间。可以看出,n 取6时,GPU仅需要0.9 ms就可以求出Patch的4条边和内部的最大高程差,不会对地形渲染效率产生明显影响。因此,将 n 取值为6,Patch对65像素×65像素全部进行了采样。

表1 GPU采样效率

Tab. 1 GPU Sampling Efficiency

n值	1	2	3	4	5	6
耗时/ms	0.001	0.002	0.003	0.021	0.185	0.905

3.2.2 Patch屏幕空间投影误差

最大高程差能够反映地形的粗糙程度,但是由于视角的可变性,实际映射到屏幕上的范围却可能很小。因此,本文将最大高程差投影到屏幕上,求得其在屏幕上投影的像素个数。Patch块

屏幕空间投影误差的计算参考文献[20]中的方法:

$$\sigma = \gamma \frac{\xi_{\max}}{d} \tag{2}$$

式中,$\gamma = \frac{1}{2}\max(R_h \mathrm{ctg}(\varphi_h/2), R_v \mathrm{ctg}(\varphi_v/2))$,$R_h$ 和 R_v 是视窗的水平和垂直分辨率,φ_h 和 φ_v 是视锥体的方位角和俯仰角;$\xi_{\max}$是 Patch 的最大空间误差;d 是视点到 Patch 包围盒的最近距离。本文将 Patch 的 4 条边和内部分开计算,各参数需要进行相应调整。计算边屏幕空间投影误差时,$\xi_{\max}$取相应边最大高程差,d 取视点到该边包围盒的最近距离。计算内部屏幕空间投影误差时,和原始公式保持一致。

3.2.3　Patch 细分因子计算

为了实现 Patch 的自适应 Tessellation,需要在曲面细分控制阶段计算 Patch 的 6 个细分因子,还要确保相邻 Patch 的邻接边细分程度相同,实现 Patch 间无缝链接。式(2)分别求出了 Patch 的 4 条边和内部屏幕空间投影误差,通过如下公式建立细分因子 F 和屏幕空间投影误差的关系:

$$F = \mathrm{clamp}(\mathrm{ceil}(\sigma \times K), x_{\min}, x_{\max}) \tag{3}$$

式中,ceil( )为向上取整函数;clamp( )为夹紧限制函数;K 是自定义的参数,K 值越大,细分因子越大;$x_{\min}$、$x_{\max}$分别是细分因子需要限制的最小值和最大值,一般取值 1 和 64。细分因子以整数形式均匀变化,避免了文献[3]由于细分因子两倍关系变换引起的顶点突变问题。通过上述方法,可以求得 Patch 四条边的细分因子。水平和竖直方向细分因子由块内屏幕空间投影误差求得,数值相同。

3.2.4　Patch 裂缝处理

本文将 Patch 的邻接关系分为以下 3 种:① Tile 内部 Patch 的邻接关系;② 相同 LoD 层次 Tile 间 Patch 的邻接关系;③ 不同 LoD 层次 Tile 间 Patch 的邻接关系。为了便于说明,将 Tile 以 2×2 的形式分割成 Patch,图 5 为部分 Patch 间的邻接关系。第一种情况,如 P_1 和 P_2,邻接边计算得到的细分因子相同,且高程数据来至同一块高度图,不存在裂缝。第二种情况,如 P_3 和 P_9,邻接边共享一列高度图纹理数据,计算得到的细分因子相同,不存在裂缝。第三种情况,如 P_2 和 P_5,T_1 和 T_2 LoD 层次不同,且高度图位于金字塔不同层次,存在裂缝。本文以 P_2 和 P_5 为例介绍第三种情况裂缝消除方法。

T_1 和 T_2 中存储着 4 个代表邻接关系的整型参数,以 i、j 分别表示 T_1 右侧和 T_2 左侧的参数值。计算 P_5 左侧边细分因子时,在原有计算方法的基础上,添加两个限制条件:① 细分因子仅能取值为 2^n,即只能在{1,2,4,8,16,32,64}中取值,比如通过计算得到的细分因子为 14,实际取值 16,依次向上取值;②式(3)中,$x_{\min}$取值 2^j,避免计算 P_2 右侧边细分因子时出现小于 1 的情况。本文算法采用 GPU 构网,P_2 无法直接访问到 P_5 的细分因子。但是,T_1 中存储着 T_2 的高程纹理信息,可以再次对该纹理进行采样求得 P_5 左侧边细分因子 2^n,则 P_2 右侧边的细分因子为 2^{n+i}。通过上述方法,P_2 和 P_5 相邻边细分顶点刚好重合在一起。此时,只要保证重合顶点在置换纹理映射时获取的高程值相同,就可以实现 P_2 和 P_5 无缝链接。T_1 和 T_2 对应的高度图位于金字塔不同层次,纹理重合部分的采样值可能会出现不同的情况。因此,在对 P_2 右侧边上各细分顶点进行纹理映射时,采样纹理不再是 T_1 所在的高度图,而是 T_2 所在的高度图。实验部分给出了最终的绘制效果,可以看出,最终绘制的地形不存在裂缝,网格连续。

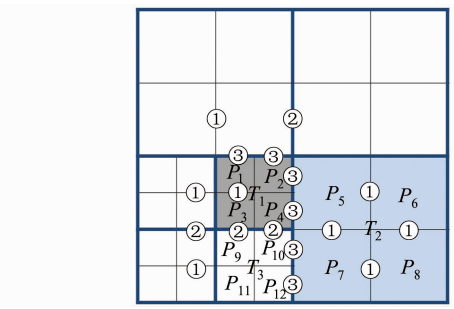

图 5　Patch 邻接关系示意图

Fig. 5　Adjacency Relationship Between Patchs

4　实验结果和分析

为了验证本文算法,在 Windows7 操作系统下,使用 VS2010 开发平台和 DirectX 11 工具以及 HLSL 着色语言设计开发了原型系统,实现了大规模地形实时绘制。硬件环境如下:Intel(R) Core(TM) i5-2380P CPU,8GB RAM,NVIDIA GeForce GTS 450 显卡,屏幕分辨率为 1 280 像素×1 024 像素。实验数据为 Puget Sound 高度图和纹理图,分辨率为 16 385 像素×16 385 像素,四叉树自顶向下分为 6 层,每个节点纹理的分辨率为 513 像素×513 像素。

为了验证本文算法的地形绘制效率,在相同

的硬件环境下,将其与传统的 CPU 四叉树算法、Geometry Clipmap 算法和文献[3]中的算法在绘制同等数量三角形下的平均帧率进行了比较(图 6)。相比 CPU 四叉树算法,虽然本文算法需要额外向 GPU 传入高度图和地形块邻接信息,但却大量减少了 CPU 需要创建的顶点个数,有效降低了高程数据实时获取时间,平衡了 CPU-GPU 负担,显著提升了地形渲染效率。相比 Geometry Clipmap 算法,本文算法也具有较大优势。相比文献[3]的算法虽然渲染帧率略有下降,但文献[3]算法构建的地形网格只是从视觉上消除了裂缝,并未从本质上解决该问题,而本文算法构建的多分辨率地形网格自适应连续,不存在裂缝。

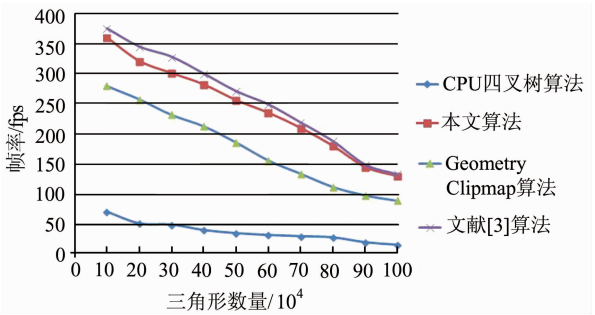

图 6 帧率对比图

Fig. 6 Comparison Chart of Frame Rates

当参数 C 取 1.5,K 取 2 时,本文算法整体和局部网格图以及叠加纹理以后的效果如图 7 所示。可以看出,地形网格绘制效果满足视距和地形粗糙度规则,距离视点越近,地形越粗糙,地形细节层次越高。相邻 Patch 网格链续,不存在裂缝。局部地形平滑过渡,完全满足可视化分析的应用需求。

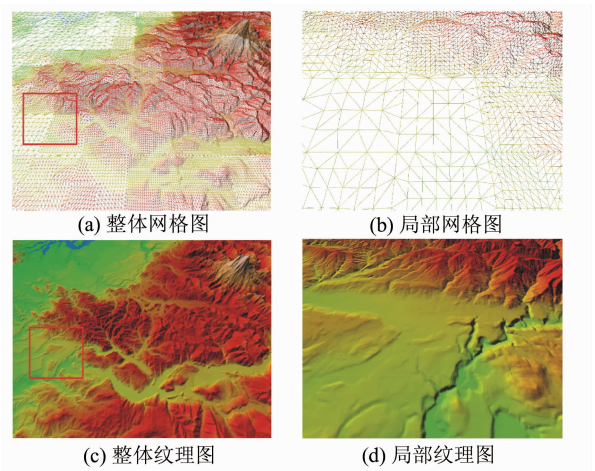

图 7 地形显示效果图

Fig. 7 Display of Terrain Surface

参 考 文 献

[1] Jin Hailiang, Lu Xiaoping, Liu Huijie. Large-Scale Terrain Realistic Rendering Based on Programmable GPU Hardware[J]. *Geomatics and Information science of Wuhan University*, 2010, 35(2): 143-146(靳海亮, 卢小平, 刘慧杰. 利用可编程 GPU 硬件进行大规模真实感地形绘制[J]. 武汉大学学报·信息科学版, 2010, 35(2): 143-146)

[2] Chen Jing, Wu Si, Xie Bingxiong. A GPU Rendering-oriented Visualization Method for Complex 3D Models[J]. *Geomatics and Information Science of Wuhan University*, 2014, 39(1): 106-111(陈静, 吴思, 谢秉雄. 面向 GPU 绘制的复杂三维模型可视化方法[J]. 武汉大学学报·信息科学版, 2014, 39(1): 106-111)

[3] Yusov E, Shevtsov M. High-performance Terrain Rendering Using Hardware Tessellation [J]. *Journal of WSCG*, 2011, 19(3): 85-92

[4] Cantlay I. DirectX 11 Terrain Tessellation[R]. Direct3D SDK 11 Whitepaper, NVIDIA, 2011

[5] Lee I, Kang K K, Lee J W, et al. Real-time Rendering for Massive Terrain Data Using GPUs[C]. Cloud Computing and Social Networking (ICCCSN), Bandung, Indonesia, 2012

[6] Zhang Bingqiang, Zhang Limin, Ai Zuliang, et al. Screen-space Adaptive Tessellation for Terrain Rendering[J]. *Journal of Image and Graphics*, 2012, 17(11): 1 431-1 438(张兵强, 张立民, 艾祖亮, 等. 屏幕空间自适应的地形 Tessellation 绘制[J]. 中国图像图形学报, 2012, 17(11): 1 431-1 438)

[7] Nie Junlan, Zhang Jingwei, Guo Dongliang. GeoMipMap Seamless Terrain Rendering Algorithm with GPU Tessellation[J]. *Computer Applications and Software*, 2012, 29(10): 99-101(聂俊岚, 张精卫, 郭栋梁. GPU 构网的 GeoMipMap 地形无缝绘制算法[J]. 计算机应用与软件, 2012, 29(10): 99-101)

[8] Nie Junlan, Zhang Jingwei, Guo Dongliang. LoD Smooth Transition Terrain Rendering Using GPU Tessellation[J]. *Journal of Chinese Computer Systems*, 2014, 35(1): 121-125(聂俊岚, 张精卫, 郭栋梁. 细节平滑过渡的 GPU 构网地形渲染[J]. 小型微型计算机系统, 2014, 35(1): 121-125)

[9] Ulrich T. Rendering Massive Terrains Using Chunked Level of Detail Control[C]. ACM SIGGRAPH 2002 Course Notes, San Antonio, USA, 2002

[10] Luo Xiangang. Digital Earth Three-dimensional Spatial Information Services Research of Key Tech-

nologies[D]. Beijing: China University of Geosciences,2010(罗显刚. 数字地球三维空间信息服务关键技术研究[D]. 北京:中国地质大学, 2010)

[11] Han Yuanli, Wang Donghan. Advanced Graphic Development with DirectX 11[M]. Beijing: Science Press, 2013(韩元利, 王汉东. DirectX 11 高级图形开发技术实战[M]. 北京:科学出版社, 2013)

[12] Yao Li, Gao Zhan, Xiao Jian, et al. 3D Graphics Programming Fundamentals Based on DirectX 11[M]. Beijing: Tsinghua University Press, 2012(姚莉, 高瞻, 肖健, 等. 3D 图形编程基础-基于 DirectX 11[M]. 北京:清华大学出版社, 2012)

[13] Luna F D. Introduction to 3D Game Programming with DirectX 11[M]. Sudbury: Jones & Bartlett Publishers, 2012

[14] Schäfer H, Niessner M, Keinert B, et al. State of the Art Report on Real-time Rendering with Hardware Tessellation[C]. Eurographics 2014—State of the Art Reports, Strasbourg, France, 2014

[15] Szirmay-Kalos L, Umenhoffer T. Displacement Mapping on the GPU—State of the Art[J]. *Computer Graphics Forum*, 2008, 27(6): 1 567-1 592

[16] Xia S, Li D. Terrain Change Detection and Updating with Image Pyramid[C]. International Symposium on Multi-spectral Image Processing and Pattern Recognition, Wuhan, China, 2007

[17] Chan Y K. BlockImage Retrieval Based on a Compressed Linear Quadtree[J]. *Image and Vision Computing*, 2004, 22(5): 391-397

[18] Li S, Lu Y, Sun W, et al. High Dynamic Range Texture Compression[P]. U S Patent 8,165,393, USA, 2012-4-24

[19] Kang H Y, Jang H, Cho C S, et al. Multi-resolution Terrain Rendering with GPU Tessellation[J]. *The Visual Computer*, 2014, 5(35): 1-15

[20] Joshua Levenberg. Fast View-dependent Level-of-detail Rendering Using Cached Geometry[C]. IEEE Visualization, Boston, USA, 2002

A Seamless Terrain Rendering Algorithm Based on GPU Tessellation

DONG Luming[1]　*ZHANG Bin*[1]　*ZHAO Xuesheng*[1]

1　College of Geoscience & Surveying Engineering, China University of Mining & Technology (Beijing), Beijing 100083, China

Abstract: In this paper, a seamless terrain rendering algorithm based on GPU is proposed to solve the problem that the conventional multi-resolution terrain rendering methods are time-consuming and not suitable for processing T-crack. Firstly, the real-time Tessellation is introduced and the terrain mesh constructing is divided into two steps, i. e. conducting coarse granularity Tile mesh on CPU and subdividing fine granularity Patch mesh on GPU. Then, in order to take the range of visibility and the terrain roughness on terrain rendering into account simultaneously, in each Tile level of detail is selected according to LoD model based on the range of visibility and in each Patch level of subdivision is selected according to LoD model based on screen-space projection error. Thus, in this algorithm the selection of the terrain LoD is realized self-adaptively. Finally, by using C++ language and DirectX 11 tools, a visual experiment system is developed and the experimental result shows that the multi-resolution terrain can be expressed seamlessly and self-adaptively ensuring the continuity of the terrain mesh. Moreover, our algorithm can significantly increase the terrain rendering efficiency through reasonable balance of CPU and GPU.

Key words: terrain rendering; GPU; tessellation; crack elimination

First author: DONG Luming, master, specializes in the visualization of 3D terrain. E-mail: 776459764@qq.com

Foundation support: The National Natural Science Foundation of China, Nos. 41171306 ,41171304.