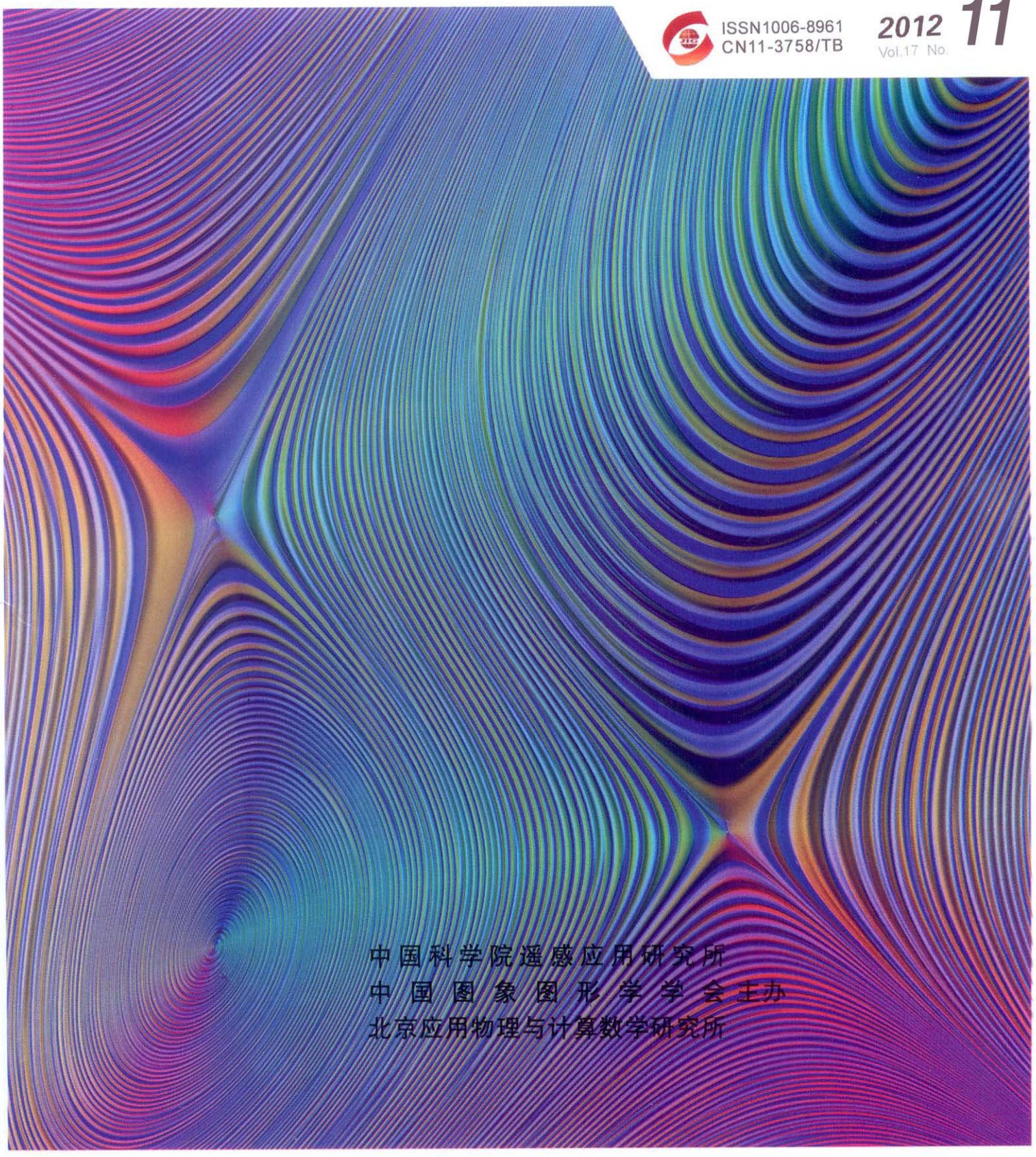
Journal of Image and Graphics

中国图象图形学报

ISSN1006-8961
CN11-3758/TB

2012 11
Vol.17 No.



中国科学院遥感应用研究所
中国图象图形学学会主办
北京应用物理与计算数学研究所

中国精品科技期刊

中国图象图形学报

Zhongguo Tuxiang Tuxing Xuebao

2012 年 11 月　第 17 卷　第 11 期(总第 199 期)

目　次

中国图象图形学报

刊名题字：宋　健　　月刊(1996 年创刊)　　第 17 卷　第 11 期　　2012 年 11 月 16 日出版

主管单位	中国科学院
主　　办	中国科学院遥感应用研究所 中国图象图形学学会 北京应用物理与计算数学研究所
主　　编	李小文
编辑出版	《中国图象图形学报》编辑出版委员会 北京 9718 信箱　邮编 100101 电子信箱：jig@ irsa. ac. cn 电话：010-64807995　010-82614429 网　址：www. cjig. cn
印刷装订	北京北林印刷厂
广告经营许可证	京朝工商广字第 0346 号
总 发 行	北京报刊发行局
订　　购	全国各地邮局
国外发行	中国国际图书贸易总公司 (中国国际书店) (北京 399 信箱　邮编 100044)

Superintended by	Chinese Academy of Sciences
Sponsored by	Institute of Remote Sensing Application, CAS China Society of Image and Graphics Institute of Applied Physics and Computational Mathematics
Chief editor	LI Xiaowen
Editor, Publisher	Editorial and Publishing Board of Journal of Image and Graphics (P. O. Box 9718, Beijing 100101, China) E-mail: jig@ irsa. ac. cn
Distributed by	Beijing Bureau for Distribution of Newspapers and Journals
Domestic	All Local Post Offices in China
Foreign	China International Book Trading Corporation (P. O. Box 399, Beijing 100044, China)
Printed by	Beijing Beilin Printing House

ISSN 1006-8961　CN11-3758/TB　CODE ZTTXFZ　国内邮发代号：82-831　国外发行代号：M1406　国内定价：45.00 元

中图法分类号: TP391.41 **文献标识码**: A **文章编号**: 1006-8961(2012)11-1431-08
论文引用格式: 张兵强,张立民,艾祖亮,张建廷. 屏幕空间自适应的地形 Tessellation 绘制[J]. 中国图象图形学报,2012,17(11):1431-1438.

屏幕空间自适应的地形 Tessellation 绘制

张兵强, 张立民, 艾祖亮, 张建廷
海军航空工程学院电子信息工程系飞行仿真技术研究所, 烟台 264001

摘 要: 为了在大规模真实感地形渲染中利用 GPU 硬件加速的 Tessellation 技术,在对地形 Tessellation 原理分析的基础上,提出一种屏幕空间自适应的地形 Tessellation 绘制算法,实现了在 GPU 内部对地形模型的三角形自适应细分。该算法采用 Tile 和 Patch 的形式对地形数据进行分层组织,在 CPU 和 GPU 上分别以 Tile 和 Patch 为基础实现地形 LOD(level of detail)的自适应简化;提出在 Hull Shader 上基于 Patch 边界的细分系数计算模型,确保了 Patch 细分时的无缝连接;给出了 Domain Shader 上置换贴图的处理过程,以实现细分顶点的高程纹理映射;并且采用了两级视锥体裁剪机制,减少了渲染数据的冗余量。实验结果表明,该算法具有较好的屏幕空间自适应性和渲染性能,能够在输入粗糙网格的基础上,渲染输出高分辨率几何细节特征的地形模型。

关键词: 地形渲染;拆嵌细分曲面;置换贴图;细节层次;图形处理器

Screen-space adaptive tessellation for terrain rendering

Zhang Bingqiang, Zhang Limin, Ai Zuliang, Zhang Jianting
Institute of Flight Simulation Technology, Department of Electronic and Information Engineering, Naval Aeronautical and Astronautical University, Yantai 264001, *China*

Abstract: In order to render large-scale terrain on graphics processing units (GPU) using hardware-accelerated tessellation, a screen-space adaptive tessellation algorithm for terrain rendering is presented. The triangulation is performed entirely on the GPU, based on analyzing the principle of terrain tessellation. The proposed approach organizes the terrain data hierarchically by tiles and patches, which is the base for a terrain LOD simplification approach processed separately on the CPU and GPU. The edge-based tessellation LOD model for each patch is constructed to compute the tessellation factors in the Hull Shader for the water tightness surface. The procedure for terrain displacement mapping in the Domain Shader is designed to offset and transform each vertex height. Furthermore, a two-level view frustum culling mechanism is used to minimize the data to be rendered. The experimental results show that the algorithm has better screen-space adaptivity and rendering performance. It can produce the terrain model with high resolution geometric details in spite of inputting coarse triangle meshes.

Key words: terrain rendering; tessellation; displacement mapping; level of detail; graphics processing unit

0 引 言

大规模 3 维地形的实时渲染一直以来都是计算机图形学和军事仿真领域的一个研究热点。虽然当前的图形硬件条件和地形 LOD 算法已经可以实时绘制相当逼真的 3 维地形,但是不断发展的卫星遥感技术和数字测量技术为快速获取地表环境的描述信息提供了充足的高精度数据来源。随着高精度地形数据规模的增加,海量地形数据与现有计算机图

收稿日期:2011-12-01;**修回日期**:2012-05-14

第一作者简介:张兵强(1981—),男,讲师,海军航空工程学院通信与信息系统专业博士研究生,主要研究方向为战场综合环境仿真、虚拟现实。E-mail: zbqwh1981@163.com

形硬件有限绘制能力之间的矛盾仍将是大规模3维地形绘制面临的主要问题。

当前,随着ATI和NVIDIA陆续推出了支持硬件Tessellation(拆嵌细分曲面)技术的GPU,其内部真正具有了专门细分曲面的硬件单元。在OpenGL 4.X的渲染管线中通过Tessellation control sdhader(TCS)和Tessellation evaluation shader(TES)两个可编程着色器提供对Tessellation技术的支持。而在DirectX 11渲染管线中,则通过在镶嵌器(Tessellator)前后增加的Hull shader(HS)和Domain shader(DS)两个可编程着色器来完善细分曲面管线。

在地形渲染中利用GPU的硬件Tessellation技术,GPU可以通过对接收的粗糙地形网格进行插值来增加新的顶点,将粗糙地形模型的三角形拆分得更细小、更细致,也即大大增加三角形数量,从而提升地形模型的几何精度,渲染输出更加逼真的地形效果,并可以减少数据传输对带宽的压力。

基于Tessellation技术的地形渲染开始成为人们研究的方向。文献[1-2]在对DirectX 11细分曲面管线介绍的基础上,探讨了地形Tessellation的一般原则和方法;而文献[3-4]则探讨了OpenGL的Tessellation性能及其在地形渲染中的应用;文献[5]利用Tessellation实现了多分辨率地形的无缝渲染,但其是一个基于虚拟地形的绘制方法,不能直接用于大规模真实地形的渲染;文献[6]采用递进编码的有损数据压缩,提出通过两个处理阶段来构造多分辨率的LOD地形,一个是原始粗糙网格的LOD选择,另一个是Patch细分的LOD选择,但是其基于Tessellation Block误差的细分LOD计算方法需要较大的HS计算开销。

在分析地形Tessellation原理的基础上,研究了大规模真实感地形绘制中的Tessellation算法,将地形数据预处理成为适合GPU批处理的分块,采用四叉树对内存地形数据块进行有效的管理,并组织成Patch和Tile的形式发送到GPU进行渲染,提出一种屏幕空间自适应的细分系数计算模型,在GPU上实现地形LOD的计算和裂缝处理,并且采用二级视锥体裁剪机制,减少了渲染数据的冗余量,给出了DS上实现Displacement Mapping(置换贴图)的处理过程和实验分析结果。

1 地形Tessellation实现原理

在地形渲染中利用Tessellation技术需要与Displacement Mapping结合使用[7]。Tessellation技术通过在原始地形网格中构建新的顶点,来直接增加三角形的数量。置换贴图能够通过顶点高程纹理采样让一张平面的网格真正实现具有不同形状的外观,如图1所示。只要使用置换贴图方法将地形高程纹理映射到细分后的网格顶点上,就能够让细分后的顶点提升/降低到不同的相对高度,然后经过像素着色处理从而形成最后输出的地形渲染效果。

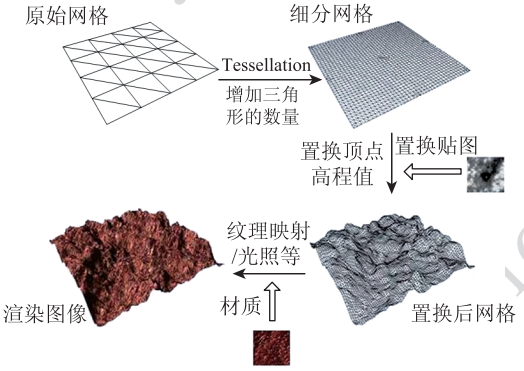


图1 Tessellation和Displacement Mapping实现地形绘制过程

Fig. 1 Terrain rendering procedure with tessellation and displacement mapping

从原理上看,基于Tessellation技术的地形渲染能够在输入低分辨率网格的基础上,绘制出高分辨率的地形,其效果相当于传统地形渲染算法在输入高分辨率地形网格时输出的渲染效果,但是其效率却要高的多,仅需要较少的几何数据就可以将平面纹理贴图改造成为具有立体感的几何图形,大大增强地形场景的真实性。

2 算法的数据组织

当前,采用四叉树对大规模地形场景进行分割是LOD地形渲染算法的主流[8],优点是可以快速地通过自顶向下的递归遍历实现地形块的查找和定位。在地形Tessellation算法中,采用基于四叉树分块的地形数据组织形式,如图2所示。每个高层次的地形块节点覆盖的区域范围是其相邻低层次地形块范围的4倍,所有层的地形块节点具有相同的顶

点个数和拓扑结构,各地形块的采样点均匀分布,处于同一层次的地形块具有相同的范围大小。

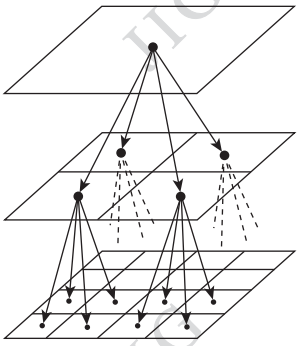


图 2　地形块的四叉树组织

Fig. 2　Terrain blocks organized in quatree

与传统基于四叉树的地形渲染算法不同的是,本文算法在描述地形块属性信息的元数据中,除了保存整个地形块覆盖区域内高程的最大值和最小值以及误差最大值(h_{max}, h_{min}, ε_{max})外,还分别保存了地形块 4 条边界上顶点的高程最大值和最小值以及误差最大值,如图 3 所示。边界上的这些元数据用于在 GPU 上计算地形块的 Tessellation 细分系数,而整个地形块的元数据用于在视锥体裁剪时构建地形块的 AABB (axis-aligned bounding boxes)包围盒。

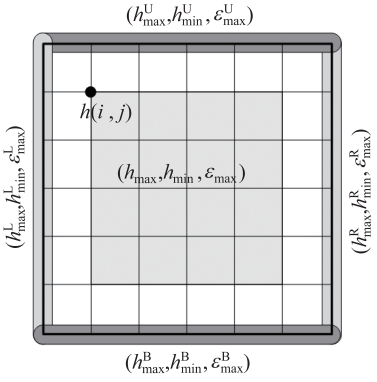


图 3　地形块的元数据分类

Fig. 3　Metadata for terrain blocks

上述的地形块四叉树结构用于实现在 CPU 上渲染节点的遍历和选择,该四叉树的深度和地形块的大小及元数据在地形数据预处理时就已确定。在渲染时,为了减少 DP (draw primitives)调用的次数和简化 CPU 的计算,将地形块组织成 Quad List 的形式发送到 GPU,每个 Quad 称为一个 Patch,包含了 $N \times N$ 个地形块,而每个 Quad List 构成一个方形的 Tile,由 $M \times M$ 个 Patch 组成,如图 4 所示。其中 M 和 N 均是 2 的幂次方。

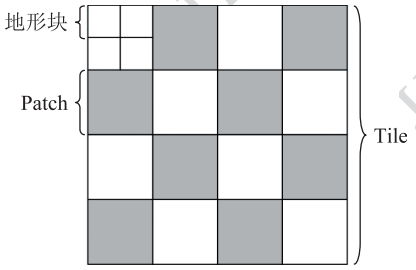


图 4　Tile 和 Patch 的数据组织

Fig. 4　Data structure of Tile and Patch

以 Patch 为 GPU 细分的基本单元。由于当前 GPU 的 Tessellator 单元最大细分系数(1 ~ 64)的限制,Patch 图元的每条边最多分为 64 份,即每个 Patch 细分后最多包含 65 × 65 个顶点,相当于覆盖了 4 个 33 × 33 地形块的大小,每个 Tile 细分后包含的顶点个数最多为 $(64M+1) \times (64M+1)$。

在 CPU 上以 Tile 作为粗糙 LOD 的选择单元。每个 Tile 内的 $M \times M$ 个 Patch 组织成一个列表的形式发送给 GPU 进行渲染,每个 Patch 有 4 个控制点,在 HS 上基于 Patch 的 4 条边分别计算细分系数,也就是计算每个 Patch 的 LOD 细分层次。

3　屏幕空间自适应的 LOD 模型

由于大规模地形渲染的视野范围从几米延伸到几千米,不可能全部采用处于同一层次大小相等的 Tile,每个 Patch 也不会具有一致的细分系数,所以,本文算法中 LOD 的选择计算涉及两个方面,一个是 Tile 的 LOD 计算模型,另一个是 Patch 的 Tessellation LOD 计算模型。Tile 的 LOD 计算由 CPU 执行,属于地形粗糙 LOD 的选择;Patch 的 LOD 计算由 GPU 的 HS 执行,属于精细 LOD 的选择。

3.1　Tile 的 LOD 模型和裂缝处理

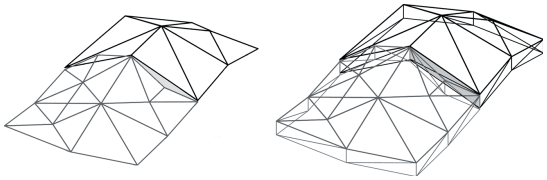
四叉树的每个节点地形块的在预处理时都计算了一个最大高程误差 ε_{max},根据视距法则和表面粗糙度法则[9-10],对于给定的屏幕空间误差阈值 τ,可以计算出 Tile 的保守屏幕空间投影误差

$$\rho = \frac{\varepsilon_{max}}{d} k \tag{1}$$

式中,d 取为视点到 Tile 包围盒最近的距离,$k = W/[2\tan(\theta/2)]$ 是一个与视窗宽度 W 及水平视角 θ 相

关的固定参数。当 $\rho < \tau$ 时,选择该 Tile 进行实例化渲染,由于 Tile 的 LOD 计算属于粗糙的 LOD 选择,可以选取保守的阈值 τ。

在不同 LOD 层次的 Tile 边界处,由于它们覆盖范围的大小和外围 Patch 细分系数等的不同,将会导致裂缝的产生。为了保持 Tile 的独立性,Tile 之间的裂缝消除采用了垂直"裙"(Skirt)的方法[11],其原理如图 5 所示。



(a) 边界处产生的裂缝 (b) 垂直"裙"遮盖裂缝

图 5 "裙"消除裂缝的示意图

Fig. 5 Illustration of processing cracks with skirts

"裙"消除裂缝方法的优点是:每个 Tile 都有自己的"裙",不需要 CPU/GPU 通过遍历查找相邻 Tile 之间的 LOD 关系和预计算边界模板,即不需要考虑当前 Tile 与其他 Tile 之间的边界连接、层次差别、拓扑关系等,每个 Tile 具有很强的独立性,非常适合在面向 GPU 地形渲染算法中实现。"裙"上边界的顶点由各 Tile 的外围 Patch 细分后得到,下边界顶点的高度值为 0,上下边界顶点具有相同的平面坐标。

虽然"裙"消除裂缝的方法避免了裂缝造成的视觉上的不连续现象,但是在地形块的拼接处人为形成了"悬崖"。由于这种"悬崖"通常很小,投影到屏幕所占的像素更少,不会造成明显的视觉影响,尤其是对于飞行仿真中的大规模地形渲染来说,强调从高空俯视地面的渲染效果,垂直"裙"的影响更加微小。

3.2 Patch 的无缝细分模型

3.2.1 Tessellation 细分模式选择

在地形 Tessellation 算法中,GPU 可以对 Patch 的每条边分别指定不同的细分系数。目前,GPU 的 Tessellator 支持 4 种细分模式 Integer、Pow2、Fractional_even 和 Fractional_odd。由于支持浮点数的分数阶细分模式可以实现顶点之间的平滑过渡,从而可以消除顶点细分引起的突跃(Popping)现象,本文算法选择 Fractional_even 细分模式,以实现地形细分顶点移动的连续性,其细分效果如图 6 所示。

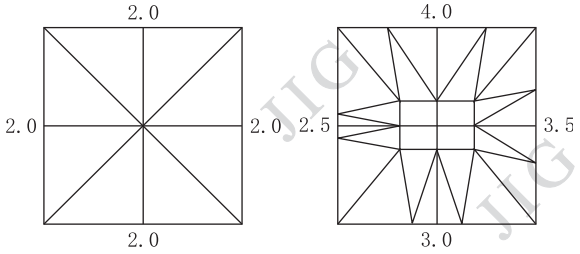


图 6 Fractional_even 细分模式效果

Fig. 6 Type of partitioning of Fractional_even

除了需要计算 Patch 每条边的细分系数之外,还需要计算 Patch 内部水平和垂直方向上的细分系数,通常情况下,内部水平方向的细分系数可取上下边细分系数的最大值、最小值或者平均值,而内部垂直方向上的细分系数则可取左右边细分系数的最大值、最小值或者平均值。图 6 的内部细分系数是取平均值的细分效果。文献[6]将 Patch 内部的细分系数分别取为边界系数的最小值,这样可以生成相对要少的三角形数量,但是在细分系数较大的边界附件顶点之间的空间连续性变差。因此,在本文算法中,Patch 内部的细分系数取为对应边界细分系数的平均值。

3.2.2 Patch 细分系数计算

为了实现地形的自适应 Tessellation,需要在 HS 中对 Patch 的边界计算不同的细分系数,同时还要确保相邻的 Patch 在共享的边界上具有相同的细分系数,否则就会导致裂缝的产生。由于 HS 对每个 Patch 的边界是单独处理的,相邻 Patch 的信息无法获知,除非像文献[5]中那样,在 CPU 上事先计算好 Patch 的相邻信息,并在渲染的时候发送给 GPU 供查询,但这样必然会加大 CPU 的运算量。

Patch 细分系数的计算是整个地形 Tessellation 算法的核心工作,既要考虑到相邻 Patch 的无缝连接,又要考虑到地形绘制的自适应性,使得距离视点较远的 Patch 具有较低的 LOD。本文结合预处理生成的地形块元数据,给出了一种屏幕空间自适应的 Patch 边界细分系数的计算模型。

由于前面分别保存了地形块 4 条边界的元数据,并且相邻地形块的共享边界具有相同的元数据,因此,以这些边界上一致的元数据为参数来计算 Patch 的边界细分系数,可以确保 Tile 内 Patch 之间的无缝连接。

若每个 Patch 包含了 4 个地形块,则 Patch 边界上顶点的极值参数等可以很容易计算出,如图 7 所

示。以 Patch 的上边界为例,其包含了地形块 1 和地形块 2 的上边界,则 Patch 上边界的顶点高程最大值 $h_{\max}^{U_P}$、最小值 $h_{\min}^{U_P}$ 和误差最大值 $\varepsilon_{\max}^{U_P}$ 分别为

$$\begin{cases} h_{\max}^{U_P} = \max(h_{\max}^{U_1}, h_{\max}^{U_2}) \\ h_{\min}^{U_P} = \min(h_{\min}^{U_1}, h_{\min}^{U_2}) \\ \varepsilon_{\max}^{U_P} = \max(\varepsilon_{\max}^{U_1}, \varepsilon_{\max}^{U_2}) \end{cases} \tag{2}$$

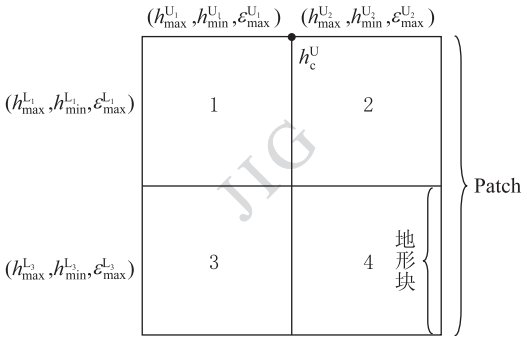


图 7　Patch 的边界参数

Fig. 7　Boundary parameters of patch

在世界空间中,以上边界的中心点 h_c^U 为中心,构建平行于 z 轴的线段 AB,如图 8 所示。

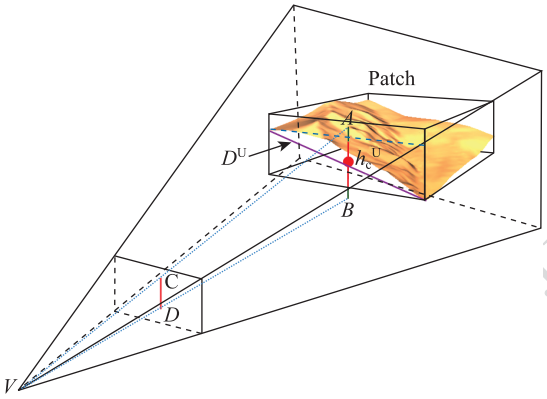


图 8　Patch 边界在屏幕上的投影

Fig. 8　Projection on screen of patch boundary

取线段 AB 的长度 L_{AB}^U 为

$$L_{AB}^U = \max((e_{\max}^{U_P} - e_{\min}^{U_P}), D^U) + \varepsilon_{\max}^{U_P} \tag{3}$$

式中,D^U 为 Patch 的上边界两端顶点之间的空间距离。根据投影矩阵 $\boldsymbol{M}_p$ 将 AB 投影到屏幕上,则可以求出对应线段 CD 的长度 L_{CD}^U。给定屏幕空间中的三角形细分门限值 ΔS,则上边界的细分系数为

$$F^U = \text{clamp}\left(\frac{L_{CD}^U}{\Delta S}, 2, 64\right) \tag{4}$$

式中,ΔS 设定了投影到屏幕上三角形所占像素的最小值,也就是最小三角形的分辨率。式(4)计算的细分系数也就决定了上边界剖分三角形的个数。Patch 其他边界的细分系数计算方法与此相同。

为了实现 Patch 内部的平滑过渡,其内部的水平细分系数取为上下边界细分系数的平均值,而垂直细分系数则取为左右边界细分系数的平均值。

上述 Patch 边界和内部 LOD 的计算方法,同时考虑了 Patch 的边长、高程值、误差值和视点的影响,可以获得较精确的边界细分系数,同时确保了相邻 Patch 的无缝连接。

图 9 给出了地形 Patch 细分后的效果图,中间 4 个 Patch 相邻的边界具有相同的细分系数,实现了地形的无缝绘制,而图 9 中外围的紫色狭长区域则是由于 Tile 内部 Patch 公共边界的细分系数不一致而导致的裂缝现象。

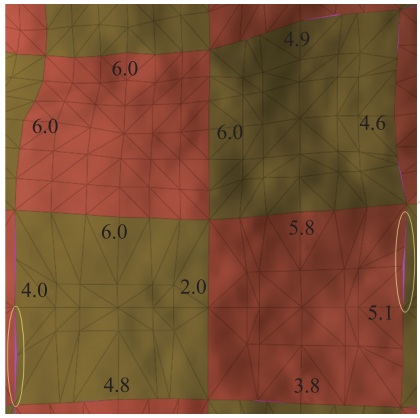


图 9　Patch 的细分效果

Fig. 9　The effect of tesselated patch

4　两级视锥体裁剪

本文算法采用两级的视锥体裁剪机制。首先,在 CPU 上使用一种层次裁剪方法,以 Tile 为单位进行 AABB 包围盒裁剪,然后在 GPU 的 HS 上以 Patch 为单位执行进一步的精细裁剪,位于视锥体之外的 Patch 不执行细分运算操作,以提高绘制效率。

4.1　Tile 的视锥体裁剪

Tile 的裁剪在 CPU 遍历四叉树节点的过程中执行。首先根据元数据构建 Tile 的世界空间包围盒,然后判断该包围盒与 6 个视锥体平面的位置关系,有 3 种状态:位于视锥体内部、相交或者完全在视锥体外部,只有当该包围盒完全位于视锥体外部时才抛弃此 Tile。

在实现时,Tile 的裁剪与 Tile 的 LOD 计算是同

时进行的。在递归遍历四叉树时,首先进行 Tile 的裁剪判断,然后对保留的 Tile 计算其 LOD。最后得到的结果是位于视锥内部或与视锥相交的、具有适当 LOD 的 Tiles,如图 10 中彩色的方框所示。然后这些 Tile 分别被组织成 Quad 图元列表的形式送到 GPU 中作进一步的处理和绘制。

4.2 Patch 裁剪

更进一步的 Patch 裁剪在 GPU 上执行,在 HS 计算 Patch 各边界的细分系数之前,首先判断 Patch 与视锥体的相交情况,以 Patch 的中心为球心,以 Patch 的对角线为直径,构建 Patch 的包围球,将包围球与视锥体进行相交测试,位于视锥体外部 Patch 的边界和中心细分系数被设置为 -1,从而 GPU 不再对其进行细分和绘制。图 10 中深彩色的部分是被裁剪掉的 Patch。可以看出,经过基于 Patch 的视锥体裁剪,渲染输出网格的冗余量比基于 Tile 的视锥体裁剪大大减少,进一步提高了绘制效率。

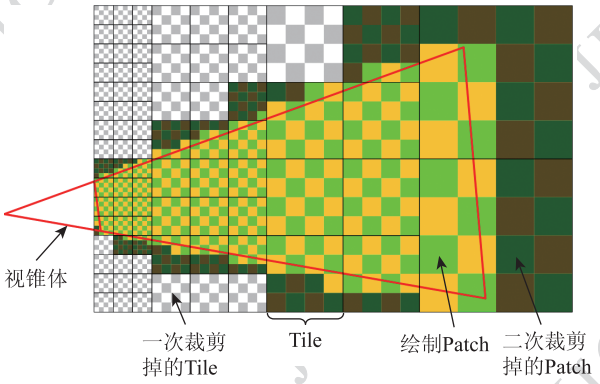


图 10 两级视锥体裁剪示意图

Fig. 10 Two-level view frustum culling

5 置换贴图

经过 Tessellator 细分的 Patch 虽然顶点数量大为增加,但是这些顶点处于同一网格内,还不能反应地形的起伏变化情况,需要结合置换贴图技术,实现细分顶点的高程纹理采样。

本文算法的置换贴图功能在 GPU 的 DS 中完成,每一个细分后的顶点都要执行一次 DS 的代码。DS 从固定功能的 Tessellator 接收细分顶点的 UV 归一化坐标,从 HS 输入 Quad 的 4 个控制点世界空间坐标、细分系数以及 PrimitiveID(Patch List 中的序号)等数据。在 DS 的内部完成插值、置换、变换和顶点属性的计算,其数据处理流程如图 11 所示。

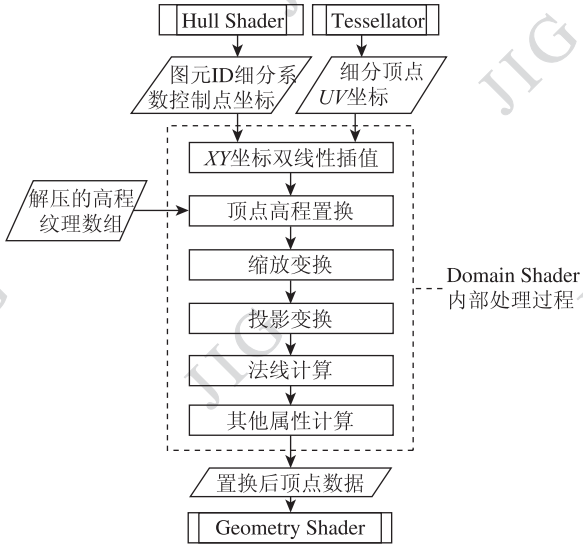


图 11 Domain Shader 数据处理流程

Fig. 11 The data flow of Domain Shader processing

对 Tessellator 输出的归一化 *UV* 顶点坐标,采用双线性插值的方法来计算其对应的世界空间 *XY* 坐标,双线性插值用到了 Quad 的 4 个控制点坐标,这 4 个控制点坐标是在 Vertex Shader 中采样高程纹理获取的。

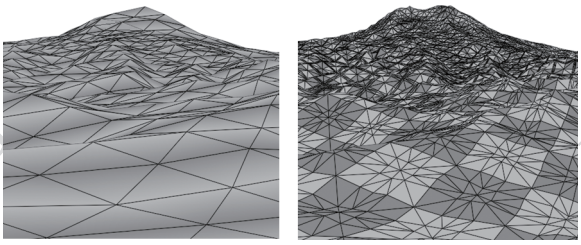
渲染时,CPU 将地形块的高程数据以纹理的形式上传到 GPU 的显存中,通过高程纹理采样完成顶点置换,然后进行比例缩放变换,从而获得顶点对应的真实高程值。

DS 还要完成顶点坐标的投影变换、法线和其他属性的计算等功能,最后输出的顶点将进入 Geometry Shader 以及后面的光栅化阶段。

6 实验结果及分析

本文地形算法的运行需要具有硬件 Tessellation 功能 GPU 的支持,实验的计算机配置为 Intel Core i5 760@2.8 GHz CPU,4 GB RAM,NVIDIA GeForce GTX460,1 GB VRAM,Windows 7 操作系统。图形渲染引擎为 Direct 3D,GPU 编程使用 HLSL 语言。以典型的 Puget Sound 高程图作为算法的测试数据,四叉树节点地形块大小为 32×32,深度为 9 层。每个 Tile 分为 8×8 个 Patch,每个 Patch 覆盖 2×2 个地形块。渲染输出屏幕窗口大小为 1 280×800。

图 12 给出了 Patch 细分前后地形网格渲染效果的对比图。图 12(a)是对输入到 GPU 的粗糙网格 Patch 直接渲染的结果,图中的每个四方形网格表示一个 Patch,每个 Patch(包含 4 个顶点)用两个三角形表示。由于顶点数量比较少,地形网格比较稀疏,所以渲染的地形几何模型比较粗糙;图 12(b)是对同样的 Patch 进行屏幕空间自适应细分后的网格渲染结果,Patch 以棋盘相间方格的形式表示。对比图 12(a)可以看出,Patch 细分后绘制的三角形数量明显增加,表现的地形细节更加丰富。



(a) Patch直接渲染网格图　　(b) Patch细分网格图

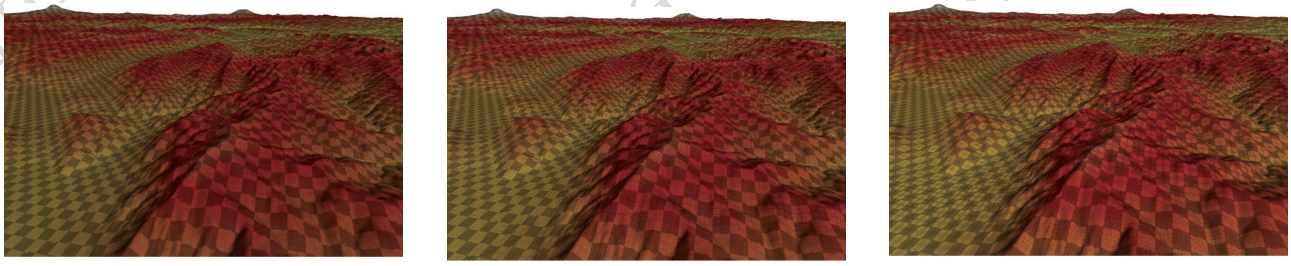
图 12　地形 Tessellation 算法网格细分实现效果对比

Fig. 12　Patch rendering wireframe comparison between with and without tessellation

另外,实验中还分别测试了本文算法在不同的屏幕空间三角形细分门限值 ΔS 下,地形渲染的帧率和渲染输出的三角形数量。图 13 分别是 $\Delta S=3$、6 和 12 时带有纹理的地形渲染效果图。

可以看出,随着 ΔS 的增加,Patch 的细分系数变小,其内部的三角形网格越来越稀疏,细分后的三角形数量明显下降;对于同一个 ΔS 的渲染效果图,随着视点到 Patch 距离的增加,细分系数也在下降,并且远处 Tile 的 LOD 变小,Patch 的表示范围增大。

为了更加直观地表示 ΔS 对 Patch 细分和地形渲染的影响,表 1 给出了 ΔS 从 1 增大到 10 时,帧率和渲染三角形的数量统计值,图 14 所示的曲线是这些数据的直观表示。



(a) ΔS=3　　(b) ΔS=6　　(c) ΔS=12

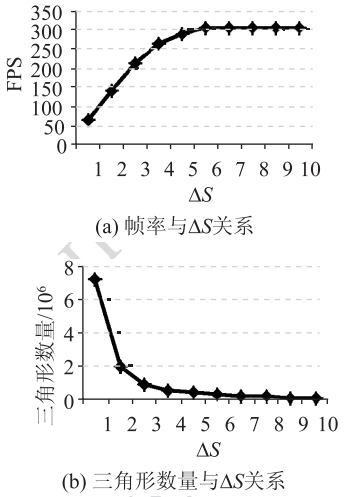
图 13　不同 ΔS 的地形渲染效果

Fig. 13　Terrain rendering effect with differentΔS

表 1　帧率和三角形数量统计值

Table 1　Frame rate and triangles per frame statistics

ΔS	帧率/(帧/s)	渲染三角形数量
1	68.63	7 213 870
2	147.57	2 005 672
3	218.19	981 406
4	272.51	615 914
5	294.16	460 440
6	310.61	342 078
7	311.54	254 336
8	312.64	196 392
9	313.22	159 248
10	313.58	135 792



(a) 帧率与ΔS关系

(b) 三角形数量与ΔS关系

图 14　ΔS 对帧率和三角形数量的影响曲线

Fig. 14　The effect of ΔS on FPS and total triangle count

可见,在渲染三角形数量达到7.2 M时,该算法仍具有60以上的帧率;随着 ΔS 的逐渐变大,渲染三角形的数量急剧下降,而帧率大幅上升,当 ΔS 增大到6以后时,帧率和三角形数量都趋于平缓。这是因为对于固定的视点和视角,由式(3)计算的投影线段长度也是一定的,而 ΔS 每增大1倍,Patch细分的细分系数减少一半,输出三角形数量成指数级减少。这与文献[5-6]中的帧率和三角形数量变化趋势是相类似的。

本文基于硬件Tessellation的地形自适应渲染算法在大大提高地形模型几何精度的同时,仍然具有较高的渲染帧率。但是细分门限值 ΔS 设定的越小,曲面细分系数就会越高,硬件Tessellator单元由于负担的计算压力过大从而会成为性能的瓶颈,GPU渲染管线的其他单元就会处于空闲状态;另一方面,当细分系数超过一定程度时,人眼就几乎觉察不到地形几何提升的效果。所以,本文地形Tessellation算法应该选择适度的细分,ΔS 等于5左右时具有较好的地形渲染效果和较高的帧率。

7 结 论

在地形渲染中采用GPU支持的硬件Tessellation技术,可以在CPU输出粗糙网格的基础上,渲染输出高精度几何特征的地形模型。本文利用当前最新的GPU可编程细分曲面管线,提出了一种屏幕空间自适应的地形Tessellation绘制算法,以Tile和Patch为基础实现LOD地形的自适应简化,给出了在HS上实现的二级视锥体裁剪机制和置换贴图的GPU处理流程。该地形绘制算法能够充分发挥GPU的硬件Tessellation性能,渲染输出更加真实的地形效果。

参考文献(References)

[1] Story J, Cebenoyan C. Tessellation performance [EB/OL]. (2010-03)[2011-10-11]. http://developer.download.nvidia.com/presentations/2010/gdc/Tessellation_Performance.pdf.

[2] Ni T Y. DX11 tessellation [EB/OL]. (2010-08)[2011-10-11]. http://www.nvidia.asia/content/asia/event/siggraph-asia-2010/presos/Ni_Tessellation.pdf.

[3] Rollin P, Oster B. Open GL and CUDA-based tessellation [EB/OL]. (2011-08)[2011-10-11]. http://www.nvidia.com/content/siggraph/Rollin_Oster_OpenGL_CUDA.pdf.

[4] San Jose C. OpenGL 4.0 Tessellation for professional applications [EB/OL]. (2010-09)[2011-10-11]. http://www.nvidia.com/content/GTC-2010/pdfs/2227_GTC2010.pdf.

[5] Cantlay I. DirectX 11 Terrain tessellation [EB/OL]. (2011-01)[2011-08-16]. http://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/TerrainTessellation_WhitePaper.pdf.

[6] Yusov E, Shevtsov M. High-performance terrain rendering using hardware tessellation [J]. Journal of WSCG, 2011, 19(3): 85-92.

[7] Castaño I. Water-tight, textured, displaced subdivision surface tessellation using Direct3D 11 [EB/OL]. (2008-08)[2011-10-11]. http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=13446

[8] Li B Y, Zhao C X. A GPU based run-time quad-tree construction method for fast terrain rendering [J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(12): 2259-2264. [李白云, 赵春霞. GPU实时构建四叉树的快速地形渲染算法[J]. 计算机辅助设计与图形学学报, 2010, 22(12): 2259-2264.]

[9] Lindstrom P, Pascucci V. Visualization of large terrains made easy [C]// Proceedings of IEEE Visualization. Alamitos, California: IEEE Computer Society Press, 2001: 363-370.

[10] Levenberg J. Fast view-dependent level-of-detail rendering using cached geometry [C]// Proceedings of IEEE Visualization. Los Alamitos, California: IEEE Computer Society Press, 2002, 259-265.

[11] Ulrich T. Rendering massive terrains using chunked level of detail control [C]// Proceedings of SIGGRAPH 2002 Course Notes35. San Antonio, Texas: ACM Press, 2002.