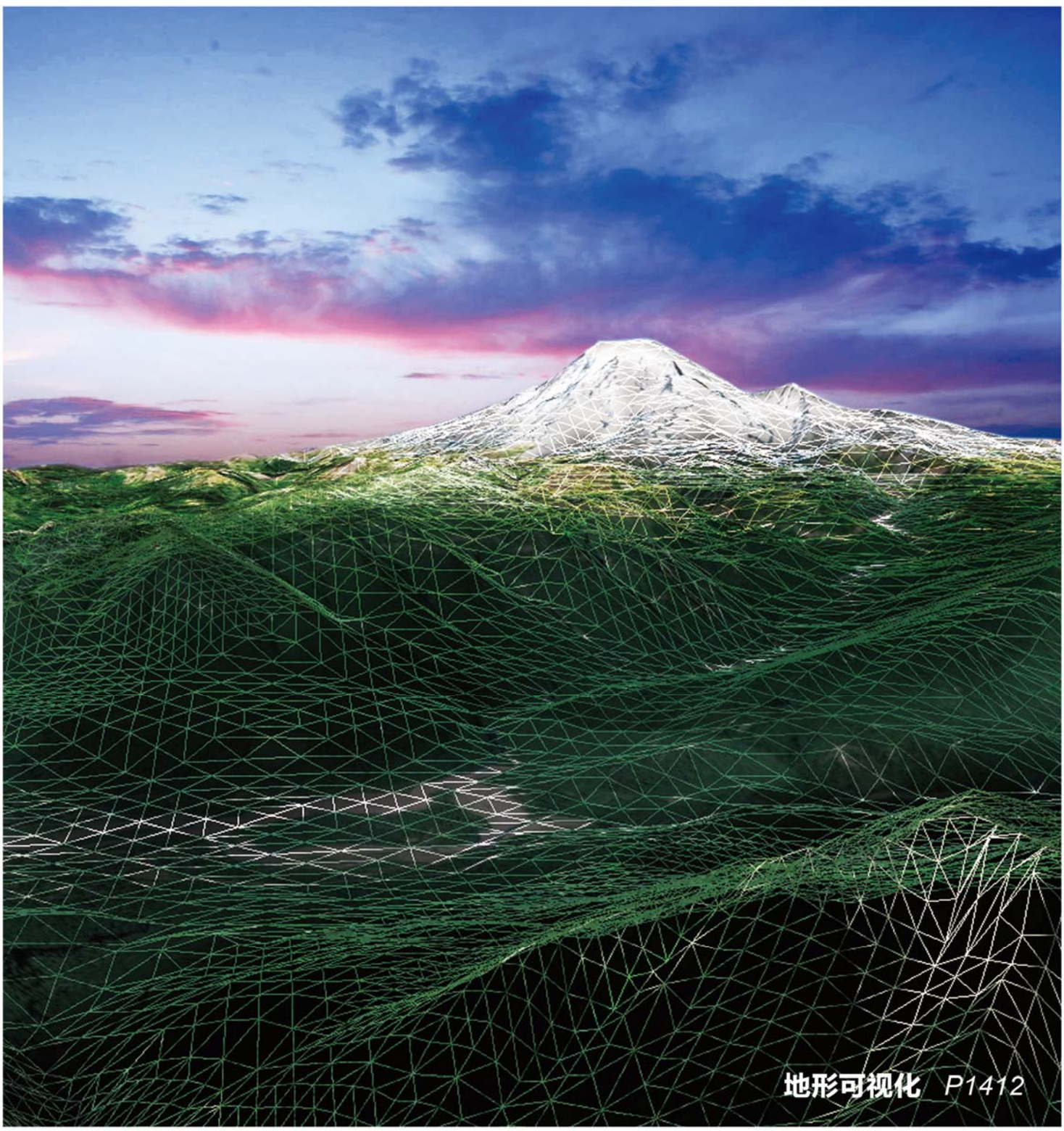
JOURNAL OF IMAGE AND GRAPHICS

主办：中国科学院遥感与数字地球研究所
中国图象图形学学会
北京应用物理与计算数学研究所

中国图象图形学报

2015
10
VOL.20

ISSN1006-8961
CN11-3758/TB



地形可视化 P1412

中国图象图形学报

刊名题字：宋健 | 月刊（1996年创刊）



第20卷第10期（总第234期）
2015年10月16日

中国精品科技期刊
中国国际影响力优秀学术期刊
中国科技核心期刊
中文核心期刊

主管单位 中国科学院
主办单位 中国科学院遥感与数字地球研究所
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
邮政信箱 北京9718信箱
邮　　编 100101
电子信箱 jig@radi.ac.cn
电　　话 010-64807995
网　　址 www.cjig.cn

广告经营许可证 京朝工商广字第0361号
总 发 行 北京报刊发行局
订　　购 全国各地邮局
海外发行 中国国际图书贸易集团有限公司
（邮政信箱：北京399信箱　邮编：100048）
印刷装订 北京科信印刷有限公司

Journal of Image and Graphics

Title inscription: Song Jian | Monthly, Started in 1996

Superintended by Chinese Academy of Sciences
Sponsored by Institute of Remote Sensing and Digital Earth, CAS
China Society of Image and Graphics
Institute of Applied Physics and Computational Mathematics

Editor-in-Chief LI Xiaowen
Editor, Publisher Editorial and Publishing Board of Journal of Image and Graphics
P.O.Box 9718, Beijing, P.R.China
Zip code 100101
E-mail jig@radi.ac.cn
Telephone 010-64807995
Website www.cjig.cn

Distributed by Beijing Bureau for Distribution of Newspapers and Journals
Domestic All Local Post Offices in China
Overseas China International Book Trading Corporation
(P.O.Box 399, Beijing 100048,P.R.China))
Printed by Beijing Kexin Printing Co., Ltd.

CN 11-3758/TB
ISSN 1006-8961
CODEN ZTTXFZ

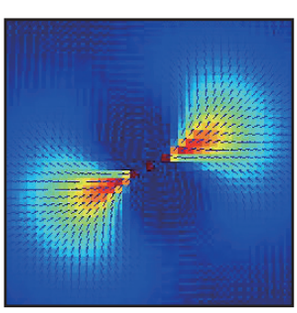
国外发行代号 M1406
国内邮发代号 82-831
国内定价 50.00元

2015年10月 第20卷 第10期（总第233期）
中国图象图形学报 Zhongguo Tuxiang Tuxing Xuebao

目次

JPEG图像双重压缩偏移量估计的篡改区域自动检测定位(第1304页)

基于表情传输的交互式照片编辑(第1366页)



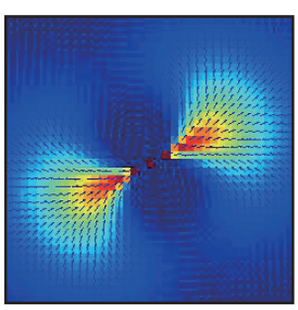
结合张量投票和Snakes模型的SAR图像道路提取(第1403页)

Volume 20, Number 10
Published October 16, 2015

CONTENTS
JOURNAL OF IMAGE AND GRAPHICS

Automatic detection and localization of image forgery regions based on offset estimation of double JPEG compression (P1304)

Interactive photo editing based on expression transmission (P1366)



Road extraction from SAR images using tensor voting and Snakes model(P1403)

E-mail: jig@radi.ac.cn
Website: www.cjig.cn
Tel: 010-64807995

中国图象图形学报
JOURNAL OF IMAGE AND GRAPHICS

中图法分类号：TP391.9　　**文献标识码**：A　　**文章编号**：1006-8961(2015)10-1412-10

论文引用格式：Li S L,Zheng L P,Zhang Y K, Li L. GPU Tessellation method of global terrain visualization[J]. Journal of Image and Graphics,2015, 20(10):1412-1421.[李尚林,郑利平,张迎凯,李琳. GPU Tessellation 全球地形可视化方法[J]. 中国图象图形学报,2015,20(10):1412-1421.]
[DOI:10.11834/jig.20151015]

GPU Tessellation 全球地形可视化方法

李尚林, 郑利平, 张迎凯, 李琳
合肥工业大学计算机与信息学院, 合肥　230009

摘　要：**目的**　目前全球大规模地形可视化问题基本都衍生于分块 LOD(level of detail)方法,该方法在快速地表漫游中依然存在 GPU-CPU 的数据传输瓶颈,其基于裙边的缝隙修复方法既需要额外资源,还存在依然无法完全消除的痕迹。为解决这些问题,提出了一种 GPU 网格生成的地形可视化方法。**方法**　结合 GPU Tessellation 方法、基于视点与屏幕空间误差的 LOD 方法、局部坐标系渲染等算法,使得全球地形可视化的生成效率有明显提高。**结果**　实现了一个全球地形可视化系统 GTVS,提供全球高精度地形数据与多分辨率高清卫星影像数据的调度与渲染等。论文对该系统进行了详实的实验和数据分析,相比传统基于 GPU 的分块 LOD 方法,FPS(frames per second)提升 100% 以上,很好地解决了系统瓶颈问题。**结论**　结果表明所提方法实用、鲁棒、扩展性好,可广泛地适用于大规模的全球渲染系统中。

关键词：全球可视化;地形渲染;GPU 渲染;GPU Tessellation;动态局部坐标系

GPU Tessellation method of global terrain visualization

Li Shanglin, Zheng Liping, Zhang Yingkai, Li Lin
School of Computer and Information, Hefei University of Technology, Hefei　230009, *China*

Abstract: **Objective** Existing methods for visualizing global-scale terrains are basically derived from the chunked LOD method. With such methods, GPU-CPU data transfer bottlenecks still emerge in real-time walkthroughs. Moreover, the addition of skirts to fix cracks requires additional resources and may not even eliminate artifacts completely. To address these issues, we proposed a GPU terrain visualization method that combines the GPU tessellation algorithm. **Method** The view-dependent and screen space error-controlled LOD method, and local coordinate system rendering algorithms. This method significantly improved the efficiency of global terrain visualization. **Result** A global terrain visualization system was implemented to provide global multi-resolution satellite images and achieve high-resolution terrain field data rendering. Extensive experiments were performed to analyze the system benchmark. Compared with the traditional GPU-based chunked LOD method, the proposed method can improve FPS by more than 100%, thus eliminating the system bottleneck problem. **Conclusion** The proposed method is practicable, robust, and widely applicable in large-scale global rendering systems.

Key words: global visualization; terrain rendering; GPU rendering; GPU Tessellation; dynamic local coordinate

收稿日期:2015-04-02;**修回日期**:2015-06-15

基金项目:国家自然科学基金项目(61300118,61370167);国家科技支撑计划(2012BAJ08B01)

第一作者简介:李尚林(1987—),男,合肥工业大学计算机与信息学院计算机技术专业博士研究生,研究方向为计算机图形学。E-mail:lsldd@163.com

Supported by: National Natural Science Foundation of China (61300118,61370167); National Key Technology Research and Development Program of the Ministry of Science and Technology of China (2012BAJ08B01)

0 引　言

全球地形可视化一直是3D GIS 或3D 数字地球系统研究中的重要研究对象。该系统由于具有超大场景尺度,需要调度海量纹理数据和高度数据,并需要进行实时的计算与实时渲染,因此需要在渲染效率、质量和实时性上寻找平衡点。

地形可视化方法的发展始终伴随计算机图形处理器的发展。90 年代由于硬件限制,地形算法局限于逐三角形渲染,因此学术界的研究重点是追求尽可能少的三角形数量得到连续 LOD(level of detail)地形网格。Hoppe[1]首次提出渐近网格算法(PM),并将该算法结合基于视点的 LOD(View-dependent LOD)方法应用到了地形可视化领域[2]。Lindstrom[3]的地形网格简化方法综合了不规则网格(TINs)大幅降低网格复杂性的优点,以及规则网格的灵活性优点,并在其文献[4]中讨论了如何将其应用到全球环境下。而 Heidrich[5]结合使用四叉树的网格优化方法进一步提高了网格生成效率。Duchaineau[6]提出的 ROAM(real-time optimally adapting meshes)算法使用二元三角树来实现对地形网格动态的合并与拆分,一度成为最主流的地形算法之一。上述算法都是基于 CLOD(Continuous LOD)思想的,其最大的问题是需要在 CPU 中处理每个三角形顶点。

从 2000 年开始,计算机硬件的 3D 处理能力开始迅速发展,GPU 每秒能处理的三角形数量不再成为瓶颈;然而地形数据的精度与数据量的增加迫切需求学者研究基于外存(out-of-core)的大规模地形渲染。如何降低 CPU 的负荷、减少向 GPU 提交数据的频率来解决 CPU-GPU 数据传输带宽瓶颈,成为新的研究重点,面向 GPU 的地形算法也开始涌现。GeoMipMaps 方法[7]首先引入了纹理存储的 mipmaps 思想来存储地形网格,Erikson[8]提出使用 HLOD(Hierarchical LOD)组织场景,避免每帧提交大量重复数据给 GPU。Ulrich[9]在 HLOD 以及文献[7, 10]的基础之上博众家所长,提出的分块 LOD(Chunked LOD)方法具有低 CPU 负载、高扩展性、高渲染质量等众多优点,尤其是其基于四叉树的数据结构能很好地扩展为全球四叉树,该算法及其相关的衍生算法逐渐成为最流行的全球地形算法,迄今被广泛应用于商业的 3D GIS 系统里[11],且随着 GPU 技术的进步而不断被研究和发展。

近几年来,随着 GPU 处理能力的进一步发展,大规模地形可视化的研究转向了多个方面,如进一步降低 CPU 负荷、基于 GPU 处理和自适应生成地形网格、提高用户体验的高性能漫游等。文献[12]的全球地形方法本质也是基于分块 LOD,其使用小波变换算法简化地形网格。文献[13]提出的视锥裁剪优化算法能够有效提高大规模地形的视锥裁剪效率,但将其扩展到适用于全球还存在一些效率问题。Geometry Clipmaps 方法[14-15]只需要在首次初始化时与 GPU 传递数据,因此更进一步提高了 GPU 的渲染效率,不过由于专利原因限制了其应用[16]。文献[17-19]聚焦于解决如何生成自适应的地形模型;文献[20]提出了一套系统的 3D 数字地球单精度浮点误差的改正方法,可以直接用于全球地形的渲染中;文献[21]则针对局部区域的地形可视化设计了一种分布式远程渲染架构;文献[22]的分块 LOD 算法使用线性四叉树代替了传统的四叉树,提升了算法效率,但其算法本质仍然是基于 GPU 的分块 LOD,将其应用于全球地形渲染仍然存在 CPU 与 GPU 交换数据的瓶颈。

2009 年,Tatarchuk[23]在 ATI Radeon 2000 上实现了最初的基于 GPU Tessellation 的地形绘制方法,激起了 GPU Tessellation 地形绘制的研究热潮。2010 年 OpenGL 4.0 和 Direct 3D 11 的发布则正式将 Tessellation 阶段[24]引入了可编程渲染管线,使得 GPU 内的曲面细分变得更加方便与高效,涌现了大量的研究成果。Mu[25]采用了基本的 GPU Tessellation 算法实现 GPU 内地形网格生成,但没有深入讨论其 LOD 方法的细节及其特殊性,且未考虑将其扩展到全球地形的情况;Lee[26]提出使用 Feedback Buffer 来进行多次 GPU Tessellation 以实现复杂地形的迭代生成;Jang[27]提出 feature-preserving geometry image(FGI)使得低精度网格顶点尽可能保持高精度网格特征,然而需要对地形数据做额外的预处理;Kang[28]将其与 GPU Tessellation 做了进一步的扩展等。

根据实际项目需求,设计并实现了基于 GPU Tessellation 的全球地形可视化系统,提供全球较高精度的地形可视化应用,可满足用户在离线或基于 C/S 架构的在线实时全球漫游。本文主要工作及其

特点包括：

1)提出了一套基于 GPU Tessellation 的地形网格生成方法，该方法能有效降低 CPU-GPU 数据传输带宽瓶颈；

2)研究结合基于视点与像素空间误差的方法，提出生成无缝的 LOD 地形网格，并使用局部坐标渲染方法规避浮点渲染精度误差错误，保证了高质量的渲染结果；

3)设计了基于 3 层渲染架构的渲染系统，并对系统进行完整的实验与测试，验证架构的高效性。

1 算法基础

1.1 GPU Tessellation 算法

通过传递给 GPU 一个四边形的 4 个顶点坐标，Tessellation Shader 能够通过指定的细分算法将四边形细分成需要的网格。以此为基础，讨论基于 GPU Tessellation 的地形网格生成方法。

文献[29]提出基于三角形法线插值的 Phong Tessellation 算法，并由文献[30]推广至四边形。考虑到地形插值的特点，使用基于四边形的 Linear Tessellation 方法。

给定四边形顶点 $\boldsymbol{p}_0$、$\boldsymbol{p}_1$、$\boldsymbol{p}_2$、$\boldsymbol{p}_3$，在重心坐标系 $(u,v)(u,v\in[0,1])$ 下，插值产生四边形内的所有网格点 $\boldsymbol{P}(u,v)$。插值算法为

$$\boldsymbol{p}(u,v)=(1-u)(1-v)\boldsymbol{p}_0+u(1-v)\boldsymbol{p}_1+v(1-u)\boldsymbol{p}_3+uv\boldsymbol{p}_2 \tag{1}$$

纹理坐标的生成方式同上。将式(1)作为 GPU Tessellation Evaluation Shader(TES)的地形曲面细分算法。

Tessellation Control Shader(TCS)通过设置四边形 4 个边界的细分数量和内部细分数量来计算(u,v)指导 TES 生成地形网格。如何计算这些参数是地形网格生成质量和效率的核心问题。在 2.1 节详细介绍如何计算四边形边界的细分。

1.2 Block 数据结构

分块 LOD 算法中进行渲染和调度的最小单位称为 Chunk，也是全球四叉树结构中的一个节点。每个 Chunk 中的数据如图 1 所示，最主要的部分是大小为 $N\times N+4N$ 的顶点位置(Position)、纹理坐标(TexCoord)，以及大小为 $N\times N$ 的索引数据(index)等信息。此外还包括如包围盒(Bounding Volume)和误差控制(Geometric Error)等其他信息。

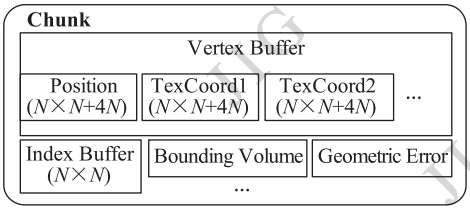


图 1 每个 Chunk 内的数据

Fig. 1 Data in each Chunk

与主流的分块 LOD 衍生方法类似，同样使用全球四叉树进行地形单元块的调度。为了高效地与 GPU Tessellation 方法结合，其在数据结构和方法上有很大差异。本文方法中这样的地形单位块称为 Block。一个 Block 中的数据如图 2 所示，注意到顶点数据、纹理数据的大小都为固定的 2×2，此外还有一些额外的控制 LOD 的信息。

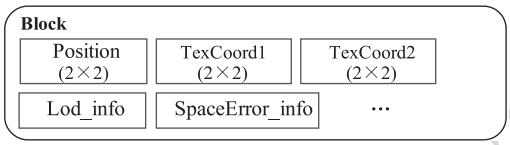


图 2 每个 Block 内的数据

Fig. 2 Data in each Block

结合 Chunk 与 Block 的数据格式，二者的区别如表 1 所示。

表 1 Chunk 与 Block 区别

Table 1 The difference between Chunk and Block

	Chunked LOD	本文 Block
结构大小	与 N 相关	固定大小
存储方式	Vertex Buffer 和 Uniform Buffer	仅 Uniform Buffer
网格大小	由 Vertex Buffer 大小决定	由 Lod_info 动态控制
裂缝修复	网格每边增加 $4\times N$ 个顶点作裙边	动态生成无缝网格
资源调度	当 Chunk 建立时申请 GPU 资源，销毁时返还资源	仅需在初始化时申请一次 GPU 资源

Block 最大的特点在于无论 N 如何取值，都仅需要维护 2×2 个顶点的数据资源，即使 N 在运行期间改变也无需动态申请 GPU 资源。因此 Block 在数据结构上具有占用 GPU 资源少、数据结构简单的优点。

2　地形网格生成

2.1　LOD 计算

地形 LOD 方法一直是地形可视化领域最核心的问题之一，对地形可视化的效率和效果有重要影响。计算 LOD 的方法一般有两种：基于视点和基于地形细节。前者具有直观和易实现的优点，后者具有高差错控制的优点，但是要达到高质量也需要数据预处理。

本文方法需要处理两个层次的 LOD：全球 Block 之间的 LOD，以及 Block 内部的 LOD。为简化描述，下文简称为块间 LOD 与块内 LOD。块间 LOD 使用基于视点的 LOD 方法，简单的线性函数即可保证任意相邻的 Block 之间级别最多相差 1。这种特性的优点能够使得缝隙修复变得简单高效，具体方法 2.2 节会详细介绍。如图 3 所示为一个全球 Block 的示意图。

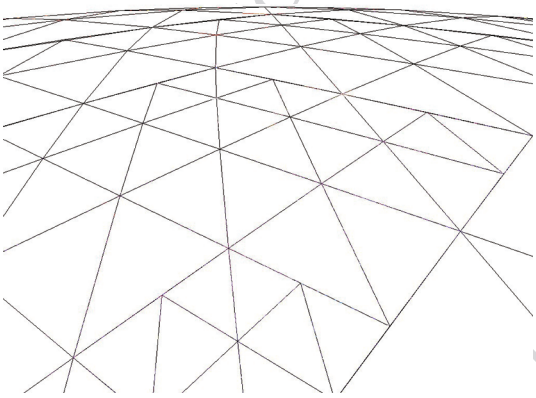


图 3　全球 Block 示意图

Fig. 3　Global Block example

对于块内 LOD，如何计算其细分等级，最普遍且效果较好的计算方法是基于通过分析屏幕空间误差（Screen-Space Error）来控制[31]。通过计算几何形变造成的像素空间误差，可以用于确定其 LOD 级别。其基本思想如图 4 所示。

假设给定屏幕分辨率宽度为 x 个像素，相机 FOV（field of view）为 θ，其与待测对象的距离为 d，地形网格的几何形变为 L（红色箭头标示向量长度），其相应的屏幕空间误差为 e 个像素。其对应公式为

$$e=\frac{L\cdot x}{2d\tan\frac{\theta}{2}} \tag{2}$$

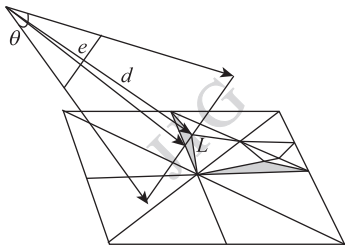


图 4　屏幕空间像素误差控制

Fig. 4　Screen-space error control

因此，对于网格任意一点 $\boldsymbol{P}$ 的高度变化 ΔH，相机位置 $\boldsymbol{V}$，几何形变 L 可由下式推导得出：

$$L=\Delta H\cdot\sin(\arccos(\boldsymbol{V}\cdot\boldsymbol{P})) \tag{3}$$

常规的误差计算都是在 CPU 中的地形更新阶段进行。本文方法的一个优势在于由于地形细分阶段在 GPU 中，因此可以通过传递当前相机坐标以及 x、e、θ 等值给 TCS 来指导具体的地形网格细分等级，从而进一步降低 CPU 的地形更新计算开销。

典型的地形 LOD 效果如图 5 所示，场景使用的最大 Block 内细分数为 32，最小屏幕空间误差为 5 像素。注意场景内山区和山间平坦河谷的不同 LOD 级别。

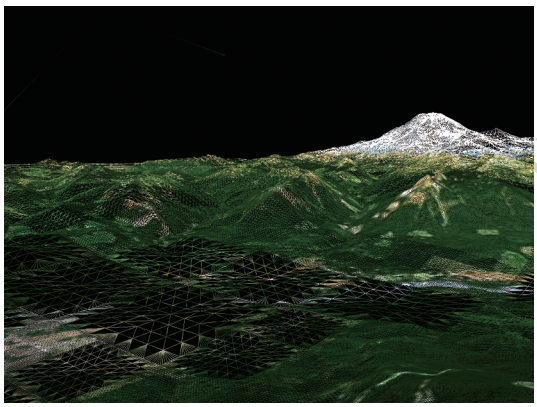


图 5　基于屏幕空间像素误差的 Block LOD

Fig. 5　The LOD based on screen-space error

2.2　裂缝消除

LOD 带来的最大问题是不同 LOD 级别的地形 Block 之间如何消除缝隙。典型的消除缝隙的方法有如下几种：在相邻不同 LOD 等级的网格中增加额外边来消除缝隙（图 6（a）中红色标示的边）、在不同 LOD 网格中加入垂直的四边形遮挡缝隙（如文献[22]等中提到的裙边，见图 6（b）中黑色部分的裙边）、在不同 LOD 网格之间使用过渡的额外网格条

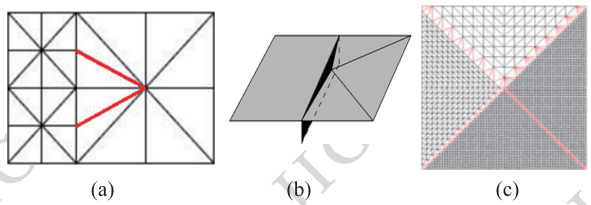


图 6　缝隙修复方法

Fig. 6　Crack fix methods

带进行连接(如文献[32],图 6(c)中红色部分的过渡条带)。

修改边界网格由于涉及修改三角形索引,因此效率最低。垂直裙边法毕竟是使用“掩饰”而不是“消除”的思想,在误差较为明显的边缘地方仍然具有不好的视觉效果。过渡带法效果好但需要额外的资源开销。

本文方法能够确保生成的地形网格任意相邻的两个 Block 之间都能无缝连接。其核心思想是:当全部四叉树节点生成完毕后,遍历所有叶子节点,查找每个叶子节点的相邻节点并询问其块间 LOD,继而计算自己网格边界细分参数。其总体算法框架如下:

算法 1　无缝网格生成算法

1)更新四叉树,所有叶节点放入队列 Q;

2)遍历每个叶节点,查找其左、下、右、上边界的相邻节点;

3)根据其相邻节点边界信息,计算本节点的左、下、右、上边界细分段数;

4)复制叶节点信息到更新完毕队列;

5)清空队列 Q。

如图 7 所示为一个典型的四叉树示例,要确定叶子节点 A 的左边界细分值,必须首先通过父节点自底向上,直到找到他的左邻居 B 节点,最后根据 B 节点的块间 LOD 来决定 A 的左边界细分参数。

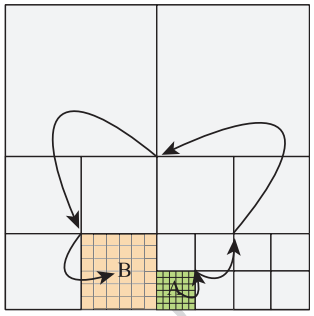


图 7　搜索邻居叶子节点

Fig. 7　Search the neighbor leaf node

对任意一个四叉树节点,用 type(UL, UR, DL, DR)来表示每个四叉树节点的类型(左上,右上,左下,右下),可知 UR 节点的左邻居一定是其兄弟节点 UL,而 UL 节点的左邻居不在其兄弟节点中,但一定是其某个祖先节点的 UR 子节点。根据这个特性,则相应有 Neighbor(UR, UL, DR, DL)表示 type 中每个元素对应的左邻居的类型。

下面以计算叶节点左边界为例给出算法描述,其他 3 个方向的边界算法与其相似。算法使用一个堆栈辅助记录搜索过的节点类型。

算法 2　叶节点左边界细分值算法

```
INPUT:叶节点 A;
    Node p = A;stack < type > S;
    WHILE ( P- > GetType( ) in(UL, DL) )
        p = p- > GetParent( );
        type t = Neighbour[ p- > GetType( ) ];
        S. push(t);
    END WHILE;
    p = p- > parent( );
    WHILE ( !S. IsEmpty && ! p- > IsLeaf( ) )
        Node q = p- > GetChild(S. pop( ));
        if(q = = NULL) break;
        p = q;
    END WHILE;
    A. Border_Left = p- > Border_Right < < (A. GetLOD
( )-p. GetLOD( ));
```

由于在 2.1 小节中已经保证任意相邻块间的 LOD 差别总是 1,因此算法 2 中最后一行只需要做一次移位操作,就能使得两个 Block 边界细分段数保持一致。当然,边界的细分参数初始值 N 必须为 2 的幂次方。

典型的不同 LOD 边界情况如图 8 所示。注意中间着色 Block 与其相邻 Block 的边界连接情况。

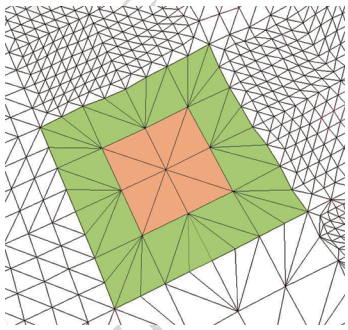


图 8　不同级别 LOD Block 的边界

Fig. 8　The border between different LOD blocks

2.3　地形网格生成

由于全球大规模场景的特殊性，浮点数计算的误差会导致近地表漫游时的渲染错误（抖动、撕裂现象），尤其在地形网格之间会出现明显的裂缝。为解决这些问题，需要避免使用单独的世界坐标系，动态地在局部坐标系上进行地形网格生成。

使用文献[20]的动态坐标系选取方法，能够在尽可能降低动态切换坐标原点频率的前提上获得较高的渲染精度。

计算地形网格顶点高度的过程在 TES 中进行。局部坐标系下计算地形高度、生成地形网格的示意图如图 9 所示。

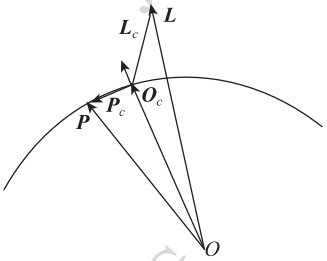


图 9　局部坐标系

Fig. 9　The local coordinate

图中 $\boldsymbol{O}_c$ 为新的坐标原点，$\boldsymbol{P}$ 为待渲染顶点，其局部坐标系下的坐标为 $\boldsymbol{P}_c$。同理，相机坐标 $\boldsymbol{L}$ 的局部坐标为 $\boldsymbol{L}_c$。若把地球视为正球体，通过获取高度数据后得到 $\boldsymbol{P}'_c$的计算方法为

$$\boldsymbol{p}'_c=\boldsymbol{p}_c+\frac{(\boldsymbol{p}_c+\boldsymbol{O}_c)}{\boldsymbol{p}_c+\boldsymbol{O}_c}\text{Texture}(T(u,v)) \tag{4}$$

Texture 通过纹理坐标 $T(u,v)$ 获取纹理中的 DEM 高度值，$T(u,v)$计算方法同式(1)。$\boldsymbol{P}'_c$计算完成后，则可以通过中心差分算法计算法线，从而完成逐顶点光照计算。

3　全球地形可视化系统实现

GTVS(global terrain visualization system)项目是一个全球地形可视化系统，其基本功能包括支持渲染对全球多分辨率高清卫星影像数据、全球地形数据的实时漫游渲染。该系统基于 C/S 架构，海量数据存储于分布式服务器。考虑到本文范畴，网络传输、优化等问题不做讨论。下面只针对 CPU-GPU 地形算法框架、全球视锥裁剪、分层渲染框架 3 个部分进行讨论。

3.1　CPU-GPU 地形算法框架

根据上文描述，本文方法可以分为 CPU 处理阶段与 GPU 处理阶段。其工作任务分别描述如下：

1）CPU 阶段。主要任务为实时的四叉树更新。包括视锥裁剪、块间 LOD 计算、边界计算、纹理与高度数据等资源的加载，以及与 GPU 的数据交换等。

2）GPU 阶段。根据传入的数据，在 TCS 中计算块内 LOD 等级，在 TES 进行地形网格生成，同时计算法线、光照、以及局部坐标系的转换等。

整个算法框架可如图 10。

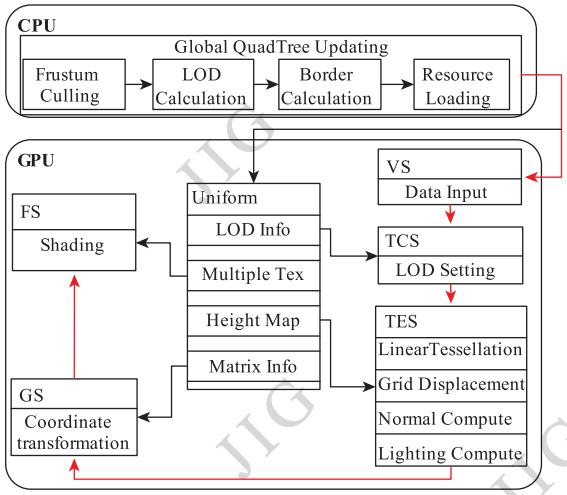


图 10　地形算法框架图

Fig. 10　The framework of terrain generation method

3.2　视锥裁剪

采用改进的文献[13]中的快速视锥裁剪优化算法。基本思想是将视锥阴影体投影到平面，然后剔除不在阴影体内的节点。由于视锥投影到平面后，每次视锥测试和传统的视锥测试相比每个顶点可少测试 3 个面，因此提高了视锥裁剪效率。示意图如图 11 所示。

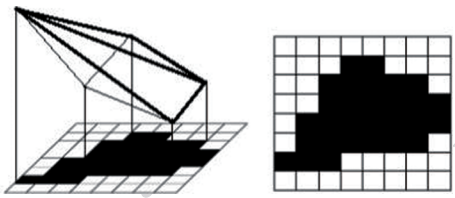


图 11　视锥裁剪方法

Fig. 11　Frustum culling method

该方法用来裁剪四叉树节点是一种很好的加速方法，然而应用于全球四叉树中则存在一些效率问题。主要是如果当视锥整体都高于地面，则反而需要判断大量多余的四叉树节点，如图 12 所示。

在这种距离地球很远的视角，如果使用视锥裁

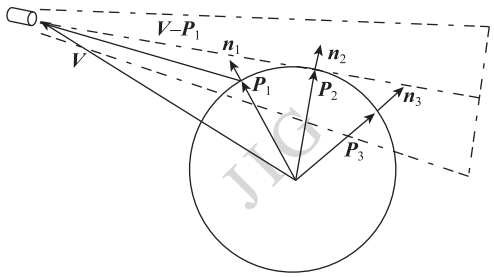


图 12　全球视锥裁剪示意图

Fig. 12　Global frustum culling

剪，则整个地球都应该在视锥内，这显然是不合理的。图 12 中，$\boldsymbol{V}$ 为相机位置的向量，$\boldsymbol{P}_1$，$\boldsymbol{P}_2$，$\boldsymbol{P}_3$ 为处于视锥内的 3 个点，$\boldsymbol{n}_1$，$\boldsymbol{n}_2$，$\boldsymbol{n}_3$ 为其法线。注意到 $\boldsymbol{P}_2$ 已经和相机朝向垂直，而 $\boldsymbol{P}_3$ 显然已经处于地球背面被地球本身遮挡而无法被看见。

为解决这种情况带来的问题，采用一种简单有效的判断方法来剔除此类节点。对于每个点 i，其判断方法为

$$f(i)=(\boldsymbol{V}_i-\boldsymbol{P}_i)\cdot\boldsymbol{n}_i-C \tag{5}$$

这里 C 为常量，C 的选取对裁剪的精确度有较大的影响。如果以 $\boldsymbol{V}_i$ 与 $\boldsymbol{n}_i$ 垂直作为判断的临界点，则可以设为 0。当 $f(i)$ 小于 0 则表示 $\boldsymbol{P}_i$ 应该被裁剪掉。总之，在距离地球很远时，考察地表法线与相机的角度可以更快地剔除不可见节点。图 3 为一个典型的视锥裁剪的例子，其中 Block 总数为 395 个。

3.3　分层渲染框架

Lindstrom 很早就提出地形的渲染与更新应为异步。渲染必须保证实时，而更新速度每秒 1 ～ 5 帧即可。虽然现代 CPU 处理能力已经远远超过该标准，但将更多 CPU 资源释放出来用于提高交互、数据预处理等方面是很有意义的。

本文将系统按照实时性的优先级从高到低分为如下 3 层：

1）渲染层。渲染层必须保证实时性。包括数据渲染、用户交互响应等。渲染层通常会保持固定的刷新频率，如果没有获取到新的数据，渲染层则使用上一次的数据渲染。

2）更新层。更新层的更新频率是灵活可变的，可以根据 CPU 负载情况进行调整。其主要工作是处理数据更新、资源调度与实时数据处理。更新层与渲染层使用双缓冲机制传递数据。新的数据产生后将通知渲染层切换数据缓冲区，从而渲染最新数据。

3）加载层。加载层的主要任务是负责将海量数据库中取得更新层需要的数据到内存。数据通常来源于外存、远端服务器，因此加载层往往是一簇工作线程在更新层的监管下协同工作。

3 层架构的基本示意图如图 13 所示。

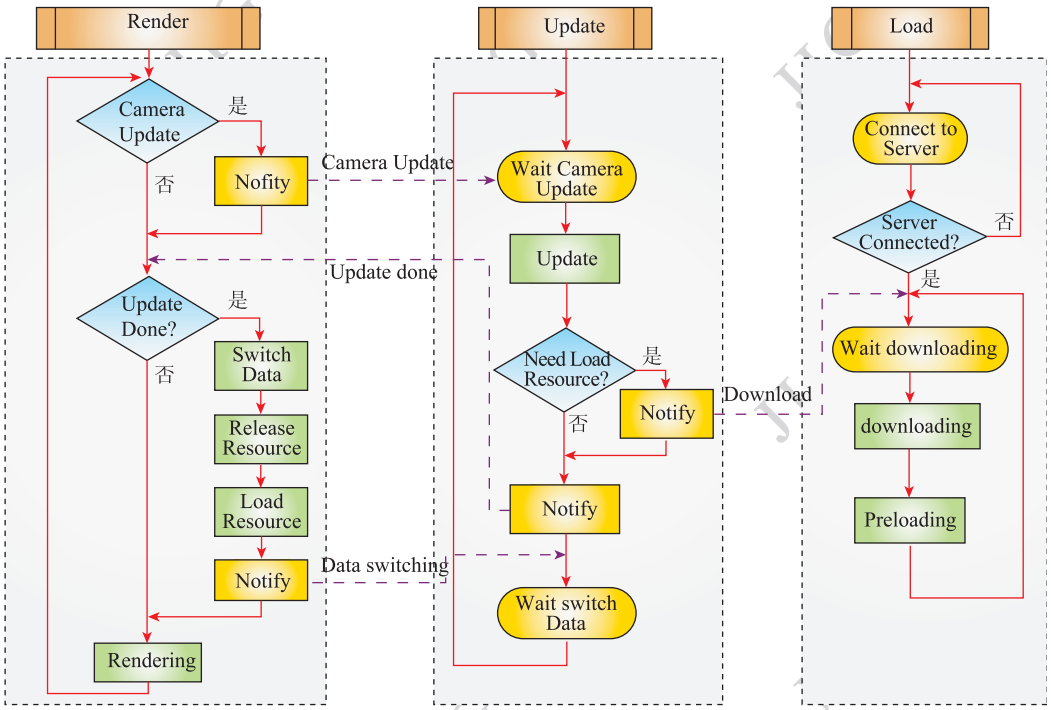


图 13　分层渲染框架

Fig. 13　The hierarchical rendering framework

4　实验与分析

GTVS 系统客户端硬件配置为 8 核 3.40 G CPU、8 GB 内存和 NVIDIA GTX 560 图形处理器(2GB 显存),平台为 Windows 7 64bit 操作系统。测试场景为经典的美国普吉特海峡(Puget Sound)雷尼尔火山,场景大小为 100 km × 80 km。全层级卫星影像数据来源于 Google Map Server,构建 20 级金字塔结构模型存储,原始数据总大小约为 2 GB,采用 JPEG 压缩做预处理;地形数据来自华盛顿大学 Geomorphological Research Group,格式为标准 USGS-DEM,数据精度可达 1/3 arc-second(约为 10 m),数据总大小约为 600 MB。地形数据使用 Yusov[33] 的方法做向下采样以及 $N \times N + 1$ 规格化处理,构建 10 级金字塔结构模型。渲染分辨率统一为 1 024 × 768。渲染结果如图 14 所示,其中 Block 初始化 $N = 16$。

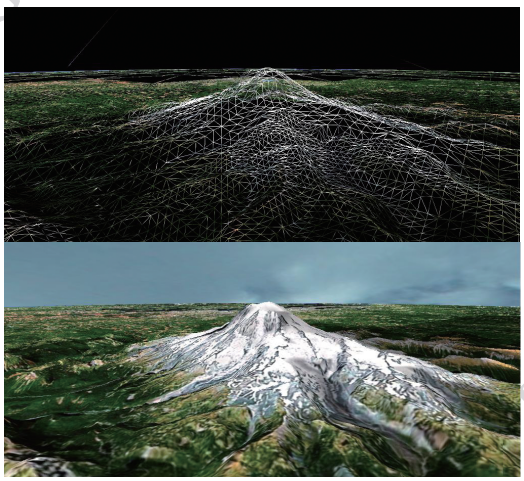


图 14　地形渲染结果

Fig. 14　The terrain render result

首先在不锁定 FPS(frames per second)的情况下,通过实验考查 CPU 负荷达到极限的情况下 GPU 的渲染效率。本文方法与目前主流的基于 GPU 的分块 LOD 方法(以下简称 GPU-CLOD)做对比。总体性能对比如表 2 所示。实验数据来源于场景 Puget Sound 的近地表漫游。

8 核处理器单个线程满负荷理论值为 12.5%,因此在不锁定 FPS 情况下 CPU 渲染线程与更新线程都呈现满负荷运行。在相机快速移动进行场景漫游的情况下,而 GPU-CLOD 只利用了 GPU 资源的 60%,渲染瓶颈显然为 CPU 与 GPU 的数据传输带宽;而本文方法无论相机处于何种状态都能利用 GPU 资源的 99%,基本解决了带宽瓶颈,从而使得 FPS 提升了 100% 以上。因此,总体性能上看,在渲染同等数量规模的三角形网格情况下,本文方法能够更加高效地利用 GPU 资源。

表 2　总体性能测试

Table 2　The overall performance test

	GPU-CLOD	本文方法
地形三角形总数/万	11.8	10.4
FPS	120	250
GPU 负荷(漫游)/%	60	99
GPU 负荷(静止)/%	92	99
CPU 负荷(漫游)/%	23	23
CPU 负荷(静止)/%	17	15

4.1　CPU 性能分析

使用固定的漫游路线进行性能测试,Block 初始细分程度为 $N = 16$。测试时序中,第 3 秒之前为系统启动与初始化阶段,3 ～ 15 s 为相机从卫星视角进入近地表视角的快速漫游,15 ～ 30 s 为相机在场景中沿地表平视远方的一般漫游操作,全程测试锁定 FPS 为 60。实验统计整个漫游过程中系统几个主要功能模块占用 CPU 时间与总时间的百分比以便在不同模块的时间消耗上做纵向对比,使用百分比做统计的目的在于忽略每一帧的差异而从整体上揭示 CPU-GPU 瓶颈的问题。同时为便于在不同的方法中做横向对比,统计出几个主要功能模块平均每帧执行所消耗的时间。分析结果如表 3 所示。

表 3　CPU 性能测试

Table 3　The CPU performance analysis

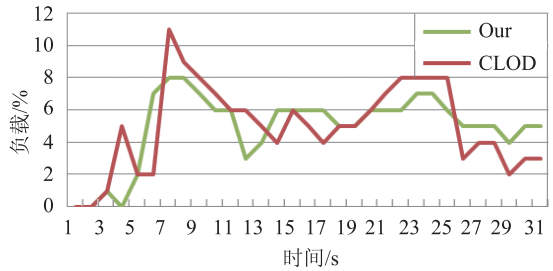
	GPU-CLOD		本文方法	
	平均每帧用时/ms	用时百分比/%	平均每帧用时/ms	用时百分比/%
地形更新	4	23.8	4.6	27.7
边界计算	无	无	1	5.8
LOD 与网格生成	0.6	3.50	0.2	1.20
地形渲染	3.5	21	1.7	10
GPU 数据传送	3.3	20	1.3	8

从表 3 上可见,具体在地形渲染和 GPU 数据传送上,相对于 GPU-CLOD,本文方法能够明显地降低 CPU 负载。尤其在视点快速移动、地形更新极为频

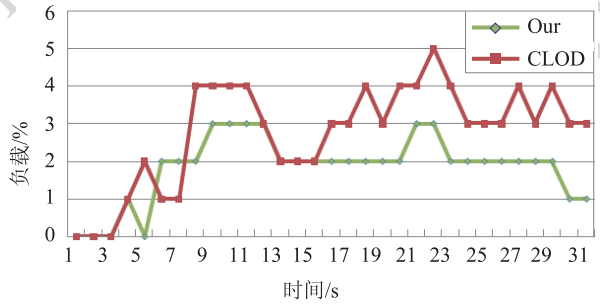
繁的情况下,GPU-CLOD 方法因为需要动态地向 GPU 申请 Vertex Buffer,CPU 的消耗会显著增加,而该情况对本文方法影响不大。本文方法额外的 CPU 消耗是算法 2 描述的边界计算,而块内 LOD 计算与网格生成的工作则转移到了 GPU 中。

4.2 GPU 性能分析

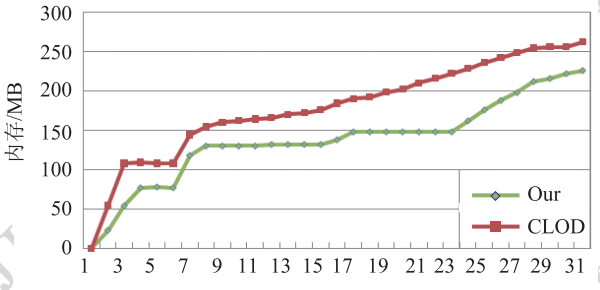
GPU 的性能分析数据来自 4.1 节的测试场景。如图 15 分别为本文方法与 GPU-CLOD 在 GPU 负载、GPU 显存控制器负载、GPU 显存使用 3 个方面的数据对比。



(a) GPU负载



(b) GPU显存控制器负载



(c) GPU显存使用

图 15 GPU 性能测试

Fig. 15 The GPU performance analysis

图 15 中 3 ~ 15 s 相机快速运动时,Block(Chunk)切换较为剧烈,GPU-CLOD 需要频繁地与 GPU 切换数据,GPU 负荷剧烈上升,进入常规地形漫游后会根据漫游的速度和块切换频率成正相关的关系。本文方法 GPU 的负荷对切换频率不敏感,但由于引入了 TCS 与 TES,在常规地形漫游阶段 GPU 的负载出现高于分块 LOD 的情况,尤其在地形较为复杂的地方随着 LOD 级别增加,TES 负荷增加会较为明显。然而考察图 15(b)和图 15(c),本文方法得到了更为平稳的 GPU 显存控制器负载以及 GPU 显存使用总量与变换率。再考虑到本文方法 GPU 承担了关键的 LOD 计算工作,系统的总体性能得到了较大提高。

5 结 论

结合算法设计与试验分析结果,本文地形可视化方法的特点总结如下:

1)效率更高。由于将使用的 GPU Buffer 降到了更低,进一步降低了 CPU 负荷以及其与 GPU 交换数据的频率,使得二者的工作耦合性更低。

2)对 GPU 更友好的设计。CPU 只需要指定相应的信息,GPU 能完成全部的 LOD 计算,并且完成全部的网格生成过程。

3)存在一定局限性。本文方法对 GPU 要求更高,必须获得支持 OpenGL 4.0(Direct 3D 11)以上版本的显卡支持。另外每个 Block 的边界细分并不是完全自由的,因此边界上的误差控制存在一定局限性。

全球地形可视化的趋势是更加面向 GPU 设计与优化,系统将把更多的 CPU 资源用于更多其他的有助于提升用户体验的方面,如网络优化、预判调度等。未来将进一步研究如何在保证渲染效率的同时更加精密的计算地形 LOD,并研究在更高地形精度(如 LIDAR 数据)的全球地形渲染中的性能与优化问题。

参考文献(References)

[1] Hoppe H. View-dependent refinement of progressive meshes [C]//Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press/Addison-Wesley Publishing Co, 1997: 189-198.

[2] Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering [C]//Proceedings of the Visualization'98. Los Alamitos: IEEE Computer Society Press, 1998: 35-42.

[3] Lindstrom P, Koller D, Ribarsky W, et al. Real-time, continuous level of detail rendering of height fields [C]//Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. New York: ACM Press, 1996: 109-118.

[4] Lindstrom P, Koller D, Ribarsky W, et al. An integrated global

GIS and visual simulation system, GIT-GVU-97-07 [R]. Atlanta, USA: Georgia Institute of Technology, 1997.

[5] Heidrich W, Slusallek P, Seidel H P. Real-time generation of continuous levels of detail for height fields [J]. Journal of WSCG, 1998: 315-322.

[6] Duchaineau M, Wolinsky M, Sigeti D E, et al. ROAMing terrain: real-time optimally adapting meshes [C]//Proceedings of the 8th Conference on Visualization'97. Los Alamitos: IEEE Computer Society Press, 1997: 81-88.

[7] De Boer W H. Fast terrain rendering using geometrical mipmapping [EB/OL]. [2014-06-20]. http://www.flipcode.com/archives/article_geomipmaps.pdf 2000

[8] Erikson C, Manocha D, Baxter Iii W V. HLODs for faster display of large static and dynamic environments [C]//The 2001 symposium on Interactive 3D Graphics. New York: ACM Press, 2001: 111-120.

[9] Ulrich T. Rendering massive terrains using chunked level of detail control [C]//Proceedings of the SIGGRAPH Course Notes. New York: ACM Press, 2002: #35.

[10] Cline D, Egbert P K. Terrain decimation through quadtree morphing [J]. IEEE Transactions on Visualization and Computer Graphics, 2001, 7(1): 62-69.

[11] Cozzi P, Ring K. 3D Engine Design for Virtual Globes [M]. Natick, Massachusetts: AK Peters Limited, 2011.

[12] Zhang L, Yang C, Liu D, et al. A web-mapping system for real-time visualization of the global terrain [J]. Computers & Geosciences, 2005, 31(3): 343-352.

[13] Youbing Z, Ji Z, Jiaoying S, et al. A fast algorithm for large scale terrain walkthrough [J]. Journal of computer aided design and computer graphics, 2002, 14(7): 624-628.

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

[15] Asirvatham A, Hoppe H. Terrain rendering using GPU-based geometry clipmaps [J]. GPU gems, 2005, 2(2): 27-46.

[16] Petterson F W L, Hugues H. Terrain rendering using nested regular grids: U. S, 7436405 [P]. 2008-10-14.

[17] Wang X, Yang X, Wang Z M. Adaptive Refinement for Terrain Rendering on GPU [J]. Journal of Computer Aided Design&Computer Graphics, 2010, 22(10): 1741-1749. [王旭, 杨新, 王志铭. 在 GPU 上实现地形渲染的自适应算法 [J]. 计算机辅助设计与图形学学报, 2010, 22(10): 1741-1749.]

[18] Yu Z, Liang X H, Ma S, et al. A real-time rendering method for large terrain including details in different resolutions[J]. Journal of Computer Research and Development, 2010, 47(6): 988-995. [于卓, 梁晓辉, 马上, 等. 一种支持大规模多种精度地形的实时绘制算法 [J]. 计算机研究与发展, 2010, 47(6): 988-995.]

[19] Zhang H J, Lu Y H, Liu S H. A terrain model simplification method based on adaptive areas division [J]. Journal of Computer Research and Development, 2010, 47(1): 53-61. [张慧杰, 吕英华, 刘淑华. 一种基于自适应区域分割的地形模型简化方法 [J]. 计算机研究与发展, 2010, 47(1): 53-61.]

[20] Li S L, Zheng L P, Zhang J, et al. Correction method of single-precision floating-point error for digita earth rendering[J]. Journal of Image and Graphics, 2014, 19(1): 119-125. [李尚林, 郑利平, 张静, 等. 数字地球渲染中单精度浮点数误差改正 [J]. 中国图象图形学报, 2014, 19(1): 119-125.] [DOI: 10.11834/jig.20140115.]

[21] Zheng L P, Chan B, Wang W P, et al. Remote visualization based on distributed rendering framework[J]. Journal of Computer Research and Development, 2012, 49(7): 1438-1449. [郑利平, 陈斌, 王文平, 等. 基于分布式渲染架构的远程可视化研究 [J]. 计算机研究与发展, 2012, 49(7): 1438-1449.]

[22] Li Q, Dai S L, Zhao Y J, et al. A block LOD real-time rendering algorithm for large scale terrain [J]. Journal of Computer Aided Design&Computer Graphics, 2013, 25(5): 708-713. [李钦, 戴树岭, 赵永嘉, 等. 分块 LOD 大规模地形实时渲染算法 [J]. 计算机辅助设计与图形学学报, 2013, 25(5): 708-713.]

[23] Tatarchuk N. Dynamic Terrain Rendering on Gaps Using Real-time Tessellation [M]. Newton: Engel W. (ed.) Charles River Media, 2009.

[24] Segal M, Akeley K. The opengl graphics system: a specification version 4.0, [R]. Technical report, OpenGL. org, 2010.

[25] Mu X, Niu X, Zhang T, et al. Terrain rendering using GPU tessellation [M]. Advances in Image and Graphics Technologies. Berlin: Springer. 2013: 269-276.

[26] Lee H, Jeong Y, Lee S. Recursive tessellation [C]//Proceedings of the SIGGRAPH Asia 2013 Posters. New York: ACM Press, 2013: #16.

[27] Jang H, Han J. Feature-preserving displacement mapping with graphics processing unit (GPU) tessellation [J]. Computer Graphics Forum, 2012, 31(6): 1880-1894.

[28] Kang H, Jang H, Cho C S, et al. Multi-resolution terrain rendering with GPU tessellation [J]. The Visual Computer, 2014: 1-15.

[29] Boubekeur T, Alexa M. Phong tessellation [J]. ACM Transactions on Graphics, 2008, 27(5): 141-144.

[30] Dupuy J. Phong Tessellation for Quads [EB/OL] [2014-06-20]. http://liris.cnrs.fr/Documents/Liris-6161-phong_tess.pdf.

[31] Reddy M, Cohen J, Varshney A, et al. Level of Detail for 3D Graphics [M]. Burlington, Massachusetts: Morgan Kaufmann, 2003.

[32] Livny Y, Kogan Z, El-Sana J. Seamless patches for GPU-based terrain rendering [J]. The Visual Computer, 2009, 25(3): 197-208.

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