高程与影像的动态匹配研究与实现*

王 柏，胡谷雨，罗健欣
（解放军理工大学 指挥自动化学院，南京 210007）

摘 要：为了解决传统大规模地形渲染算法中高程与影像数据单一匹配、投影方式固定且对数据完整性容忍不高的问题，提出一种支持任意数据分块、投影方式多样以及容忍非完整数据的高程影像动态匹配方案。该方法通过一系列的坐标变换以及对数据块作相应处理，实现多种影像与高程数据的准确匹配，提出一种新型数据加载机制（子承父高法），解决了在相邻块暂无精细高程数据，而周围场景使用较为粗糙的高程数据时出现的陡峭问题；提出高程数据加载与地形块更新模块异步执行的新方法，提高了系统性能。最后通过自主开发的数字地球系统验证了该方案的可行性与有效性。

关键词：数字地球；细节层次；四叉树；墨卡托投影；子承父高
中图分类号：TP311 **文献标志码**：A **文章编号**：1001-3695(2012)06-2360-04
doi:10.3969/j.issn.1001-3695.2012.06.096

Dynamic matching of altitude and image

WANG Bai, HU Gu-yu, LUO Jian-xin
(*Institute of Command Automation*, *PLA University of Science & Technology*, *Nanjing* 210007, *China*)

Abstract: In order to solve the problems in traditional terrain rendering algorithms, such as altitude and image data must be one-to-one, and the projection has to be ortho-projection, also the data should be integrated, this paper proposed a dynamic matching method of image and altitude. It designed some coordinate changing and data block managing to match the image and altitude accurately, and used a new method (inherit parent's altitude) to solve the craggedness problem, which happened when there was no high-level altitude data in close scene, but around displayed low-level terrain; the loading of altitude data was aparted from the update of the whole terrain block to improve the system performance. The experimental result approves the method is feasible and valid.

Key words: digital earth; level of detail; quad tree; Mercator projection; inherit parent's altitude

大规模地形的渲染一直是图形学领域中的热点问题。随着数字地球[1]和全球信息网格等概念的相继提出，建立全球规模的虚拟地形场景受到越来越多的关注。本文在细节层次[2]（level of detail，LOD）技术基础上提出一种高程数据动态的匹配影像数据的方案，主要解决了三个方面的问题，使不同的影像数据均可正确地匹配同一种高程数据，大大减少了不必要的预处理过程：a）不同影像与高程数据集的分割方式不同而导致的两者数据块并非一一对应的问题（图 1）；b）不同的影像块与高程数据集的投影方式不同导致网格创建不匹配问题（图 2）；c）高程数据不完整导致的最终显示出现陡峭问题（图 3）。

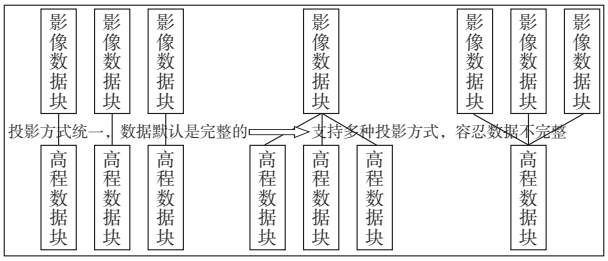
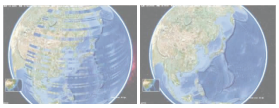


图1 传统影像高程匹配与本文的改进对比

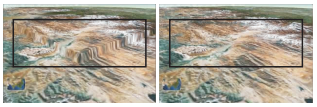
1 相关技术研究

LOD 技术就是物体在实时表现的时候运用的一种模型简化和优化技术。这种技术有静态和动态两种方案。静态的方案就是处理程序预先将不同分辨率的模型建立好，当在绘制物体时，根据一定的阈值条件，选择一定分辨率的模型对物体进行绘制。动态的方案是指在程序的运行过程中，根据某些触发条件，实时调节模型分辨率。



(a) (b)

图2 投影方式不匹配导致的渲染失败与改进后的效果对比



(a) (b)

图3 陡峭现象消除前后效果对比

基于 LOD 技术的大规模地形渲染研究越来越多。文献[3]在 LOD 基础上提出一种适合于 GPU 实现四叉树场景分割和渲染的算法，利用纹理和像素着色器实时构建四叉树，使用几何着色器实现 GPU 对四叉树的遍历和场景分割。文献[4]提出了以 GPU 直接处理 DEM 的算法，该算法的优点在于不需要复杂的场景分割等步骤，效率比较高，但缺陷在于不是多边形算法，所以应用范围受到限制。文献[5]提出了一种新的地形渲染算法，其主要特点是将地形高程数据组织成单通道纹理存放于显存，用于支持视点跟随的 LOD 地形渲染。文献[6]是通过对顶点曲度、感知强度、视点移动速度这三个因素进行量

收稿日期：2011-10-02；**修回日期**：2011-11-21 **基金项目**：国家自然科学基金资助项目(60303023)

作者简介：王柏(1987-)，男(回族)，硕士研究生，主要研究方向为计算机图形学(372882017@qq.com)；胡谷雨(1963-)，男，教授，主要研究方向为网络管理；罗健欣(1984-)，男，博士研究生，主要研究方向为计算机图形学.

化,并把它们融合进节点评价函数中,进而提出的一种新算法。文献[7~9]是最近的关于 GPU 友好的地形渲染研究,使用的是光栅化与光线投射相结合的混合渲染方式。文献[10]也是较新的大规模地形渲染以及网格等相关技术的研究,通过类似阴影投射的方法将纹理图集映射到高程图集上以完成在 GPU 上的渲染。

以上算法大多针对已有数据进行处理,且假设采用的投影方式都是正交投影,并没有过多地考虑引言中提出的问题。但是,现实中使用的影像数据与高程数据不一定是一一对应的,影像与高程的数据集可能不统一。例如 NASA 提供的影像数据集有多个:Blue Marble、LandSat7 等,它们的分块大小都不一样; Google 的 Google Earth、Microsoft 的 Bing Map 等软件也分别提供不同分块的影像数据集;采用的地图投影方式也多种多样,如正交投影、墨卡托投影、高斯投影、兰伯特投影等;客户端从服务器下载数据的延时或者下载线程的阻塞都会导致数据不完整等问题。本文在阅读了相关文献的基础上,提出一种高程与影像数据实时动态匹配方案,实现高程数据的异步加载,解决了提出的三个主要问题。不论采用何种影像数据集与高程数据集,也不论采用何种投影方式均可准确匹配。下面以自主开发的数字地球系统所采用的数据集和投影方式为例,以实际应用为背景进行详细阐述。传统的影像高程匹配与本文改进后的匹配方案比较如图 1 所示。

2　动态匹配方案描述

2.1　数据块的匹配

以自主开发的数字地球采用的影像数据与高程数据为例,两者均可使用四叉树结构来描述和管理。其中,影像数据对应的四叉树结构首层节点个数为 1×1(行×列),而高程数据对应的四叉树结构首层节点个数为 9×18,它们并非一一对应。对于指定影像块的指定位置,如何判定与之匹配的高程块,如何获取该位置的高程值是需要解决的问题。

因为一个影像四叉树块可能对应着多个高程数据块,所以根据影像四叉树块的网格精度额外创建一个高程数据块 T,存储影像块每一个网格点的高程值,这些高程值需要从多个真实高程块 t 中读取。在创建地形块的网格时,需要加载影像块对应的高程数据,这个过程因需要读取多个真实高程块获取高程值而变得很耗时,会使得更新线程一直占用该四叉树块,从而阻塞了渲染线程,导致帧速急剧下降。为此,在数据的更新过程中,将加载高程数据分离出来,使其异步进行,并通过缓存队列机制来管理与影像数据对应的高程块 T。高程数据的更新流程如图 4 所示。右半部分为地形块的数据加载线程,其主要功能是为高程数据块 T 加载真实高程数据值,这一功能通过读取 T 覆盖范围内包含的多个真实高程数据块 t 而获得;左半部分为地形块的匹配更新线程,主要功能是根据当前视野需要,读取与影像块对应的高程数据块 T 的值,如果 T 还没有更新,就将该块加入到缓存队列中,供流程右半部分更新(加载和更新细节参见 2.2 节)。

如何根据影像块的范围来确定对应的高程块 T 包含哪些真实高程数据块 t,这就是 2.2 节将要讨论的网格创建需要解决的问题。

2.2　数据网格的创建

影像数据采用墨卡托投影,将平面图像映射到球面上,模拟真实的地球场景。在四叉树的分割过程中,使用全球地理坐标,而高程数据则采用经纬度点来记录某一位置的高程值,这时两者就出现了不匹配现象。因此在创建网格时,影像数据某一网格点对应的高程值的获取就需要使用墨卡托投影公式将全球地理坐标转为经纬度坐标,然后使用该经纬度坐标去获取该点的高程值。匹配失败的原因是在计算影像块的网格中点时没有使用墨卡托投影将中心点的地理坐标转换为经纬度坐标,而是直接使用四个角的经纬度取中间值作为中心点的经纬度,此时的中心点与其实际的中心点在南北方向上有偏移,在东西方向上可以正确匹配。这种不匹配的网格导致纹理在南北方向上有拉伸和挤压的效果(效果对比如图 2 所示)。

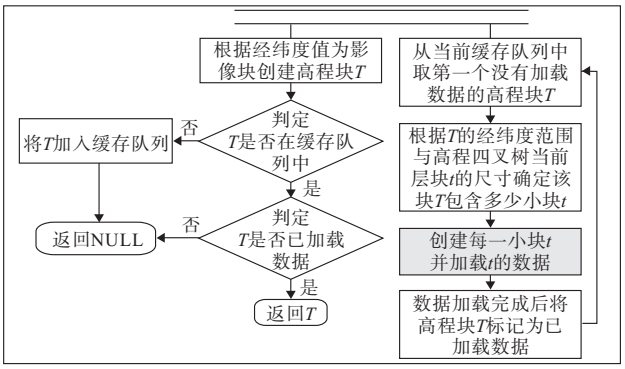


图4　高程数据的更新流程

公式 1　墨卡托投影正反解公式(注:标准纬度 B_0、原点纬度 0、原点经度 L_0、K 为比例因子、e 为第一偏心率、e' 为第二偏心率)。

正解:经纬度坐标转为地理坐标 $(B,L)\rightarrow(X,Y)$

$$X_N = K\ln\left[\tan\left(\frac{\pi}{4}+\frac{B}{2}\right)\times\left(\frac{1-e\sin B}{1+e\sin B}\right)^{\frac{e}{2}}\right]$$

$$Y_E = K(L-L_0)$$

$$K = N_{B_0}\times\cos(B_0) = \frac{a^2/b}{\sqrt{1+e'^2\times\cos^2(B_0)}}\times\cos(B_0)$$

反解:地理坐标转为经纬度坐标 $(X,Y)\rightarrow(B,L)$

$$B = \frac{\pi}{2}-2\arctan\left(\exp\left(-\frac{X_N}{K}\right)\times\exp\left(\left(\frac{e}{2}\right)\ln\left(\frac{1-e\sin B}{1+e\sin B}\right)\right)\right)$$

$$L = \frac{Y_E}{K}+L_0$$

其中:exp 为自然对数底,纬度 B 通过迭代计算很快就收敛了。然后异步地执行下面两个步骤(结合图 4 所示的高程数据更新流程):

a)根据通过墨卡托投影反解公式(见公式 1)计算出的影像四叉树块的经纬度范围。创建一个新的与该影像四叉树块对应的高程数据块 T,T 的行列数由影像块的网格精度决定。判定 T 是否已经存在于缓存队列 1 中,如果不存在,将 T 加入缓存队列 1,并返回 NULL,表示该块暂时没有加载高程数据,此时视该地形块未更新,不予渲染(渲染线程会渲染该块的父节点)。如果 T 存在于缓存队列 1,则将 T 从缓存队列 1 中取出并判定 T 是否已经加载了数据,是则返回 T,否则返回 NULL。

b)与步骤 a)异步进行,不断地从当前缓存队列 1 中取第一个未加载数据的高程数据块 T,根据 T 的经纬度范围计算 T 包含多少块该层对应的实际高程四叉树块 t,并根据具体的经纬度坐标创建每一小块 t。根据影像网格精度与高程的分辨率计算高程数据的层数(见代码 1),然后根据代码 2 计算 t 在该

层的行列号,判断 t 是否存在于缓存队列 2 中。如果存在,直接使用;否则根据 t 的层行列加载数据文件,并读取 t 中对应位置的高程值填充高程数据块 T 中每一网格点的高程值(代码 3)。然后将 t 放入缓存队列 2 备用。各个小块数据加载完成后,将 T 标记为已经加载了数据。

代码 1 计算所需高程数据的层数

```
samplesPerDegree 影像网格精度(每一度采样点个数)
m_samplesPerTile 高程数据每个数据文件采样点个数
m_levelZeroTileSizeDegrees 第 0 层高程数据每一块覆盖的经纬度范围
for(int i = 0; i < m_MaxLevels; i++)
{
  //当前层高程数据块每一块的经纬度范围
  double range = m_levelZeroTileSizeDegrees * pow(0.5,i);
  //当前层高程数据的分辨率,每一度的采样点个数
  double temp = m_samplesPerTile / range;
  /* 如果高程分辨率满足了影像数据的精度,则采用当前层的高程;否则判断下一层是否满足要求 */
  if(samplesPerDegree <= temp)
  {
    m_TargetLevel = i;
    break;
  }
}
```

代码 2 计算所需高程的行列号

```
//当前层的每一数据块大小(度数表示)
TileSize = m_levelZeroTileSizeDegrees * pow(0.5,TargetLevel);
//当前数据纬度为 lat(-90,90),则行号为
Row = (lat+90)/TileSize;
//当前数据经度为 lon(-180,180),则列号为
Col = (lon+180)/TileSize;
```

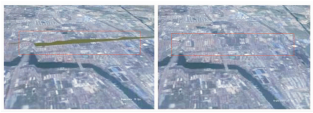
代码 3 在高程数据文件中取对应点高程值

```
// m_North,m_West 为高程数据文件的边界值
// latitude,longitude 为目标点经纬度
// SamplesPerTile 为数据文件采样点个数
// TileSizeDegrees 为数据文件经纬度范围
//计算目标点在数据文件对应的位置
double deltaLat = m_North - latitude;
double deltaLon = longitude - m_West;
double df2 = (m_SamplesPerTile-1)/ m_TileSizeDegrees;
float lat_pixel = (float)(deltaLat * df2);//目标点位置
float lon_pixel = (float)(deltaLon * df2);
//周围四点插值获得目标点高程值
//更加准确地描述该点的高程值
int lat_min = (int)lat_pixel;
int lat_max = (int)ceil(lat_pixel);
int lon_min = (int)lon_pixel;
int lon_max = (int)ceil(lon_pixel);
float delta = lat_pixel - lat_min;
float westElevation =
m_ElevationData[lon_min+lat_min*m_SamplesPerTile]*(1-delta)+
m_ElevationData[lon_min+lat_max*m_SamplesPerTile]*delta;
float eastElevation =
m_ElevationData[lon_max+lat_min*m_SamplesPerTile]*(1-delta)+
m_ElevationData[lon_max+lat_max*m_SamplesPerTile]*delta;
delta = lon_pixel -lon_min;
float interpolatedElevation =
westElevation*(1-delta)+eastElevation*delta;
```

无论是 T 的缓存队列 1 还是 t 的缓存队列 2 都有大小限制,因为不可能无限制地将所有浏览过的数据块均放入缓存队列。因此,在数据块加入缓存队列时,记录当前程序运行时间,当缓存队列满时,根据当前视野删除最不相关的数据块,为操作方便,可将最先加入该队列的数据块视为最不相关,找到当前队列中最先加入的数据块,将其移出缓存队列,并释放掉该块所占用的内存。根据获取的与影像块网格精度对应的高程块 T,可以精确地绘制出该地形块对应的网格,通过贴图影像纹理即可获得良好的地形显示效果,如图 5 所示。

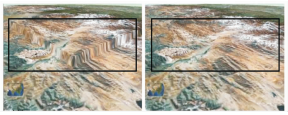
2.3 出现的问题及其解决方案

当相邻两地形块有一块没有高程数据时,在交界处会出现相对较大的裂缝。本文采用传统的垂直边缘法[12](vertical skirt)来消除地形裂缝,如图 6 所示。但是当该裂缝出现在海拔高的地区,而所需要的高程数据还未下载或没有该地区的高程数据时,边界处会出现很陡峭的悬崖(如图 3(a)所示,方框中一个 V 字形斜坡)。



(a) (b)

图 5 裂缝消除前后效果对比



(a) (b)

图 6 陡峭现象消除前后效果对比

该问题的解决办法形象地称之为子承父高法。即在高程数据更新模块中加载数据时(如图 4 高程数据更新的灰色块部分),当前块的高程数据如果不存在,则使用其父(祖)节点在该区域的高程值,并做好标记,隔一段时间再次更新时,当前块的高程数据依然要进行一次更新,以查看该高程数据是否已经存在(可能刚刚下载完成)。代码 4 为该方法实现的示例。陡峭现象修正后的效果如图 3(b)所示。

代码 4 在高程数据文件中取对应点高程值

```
while((m_TargetLevel>0)&&(fileExist! =0))//封装文件中不存在
{
//依当前地形块属于其父节点的哪个子节点来定义新经纬度范围
if (m_Row%2==0)//southchild
    {
        m_North = m_North + (m_North-m_South);
    }
    else//northchild
    {
        m_South=m_South - (m_North-m_South);
    }
    if (m_Col%2==0)//westchild
    {
        m_East=m_East+(m_East-m_West);
    }
    else//eastchild
    {
        m_West=m_West-(m_East-m_West);
    }
    //计算当前地形块的父节点层行列号
    m_TargetLevel--;
    m_Row=(int)(m_Row/2);
    m_Col=(int)(m_Col/2);
    m_UseParentData=true;//做好标记,实时检测更新
    fileExist=a_FPS→SQueryFileExisted(this);
    }
```

3 实验分析

本文数字地球系统测试硬件配置为酷睿双核 E7400 2.8 GHz,1.96 GB 内存,ATI Radeon HD 3450 显卡,软件环境为 Windows XP,客户端VC++6.0,服务器端 Eclipse5.0,使用 OpenGL 图形程序接口与 ActiveX 实现数字地球在浏览器上的显示。所使用的数据如表 1 所示。

表 1　实验数据说明

数据类型	文件类型	文件平均大小/KB	最大层数	数据总量
影像	JPG	15	19	6 TB
高程	ZIP	30	11	500 MB
矢量	SHP	60	5	40 MB
地名	XML	5	6	20 MB

表 1 中,影像数据采用的是 Google 数据,而高程数据使用的是 SRTM 数据。两者是不匹配的,通过本文提出的动态匹配算法实现了地形的正确渲染。

地形块的更新包括影像数据的加载和影像网格的创建。其中影像网格的创建需要使用高程数据,而高程数据的加载过程也比较耗时,故另外启动一个线程来完成高程数据的加载,供网格创建时使用。在 WinAPI 中函数 GetTickCount( )可以获取程序从开始执行到此处时的时间(单位:ms),通过在影像块更新、高程加载以及影像块渲染前后添加该语句,并计算每个操作前后两次返回时间差,即可获取执行一遍相应操作的耗时(示例见代码 5),统计多次取平均值可有效降低由运行环境导致的误差。各个线程针对一个影像四叉树块的操作所需要的时间统计均值如表 2 所示。

代码 5　统计程序操作耗时示例

```
#if def _DEBUG
    double time1,time2,updatetimecost;
    time1 = GetTickCount();//获取更新前程序运行时间
#end if
//执行影像块的更新操作
UpdateWithPriority(g_pGlobalRenderlist, RENDER_PRIORITY_IMAGE);
#if def _DEBUG
    time2 = GetTickCount();//获取更新后程序运行时间
    updatetimecost = time2 - time1;//计算时间差,即操作耗时
#end if
```

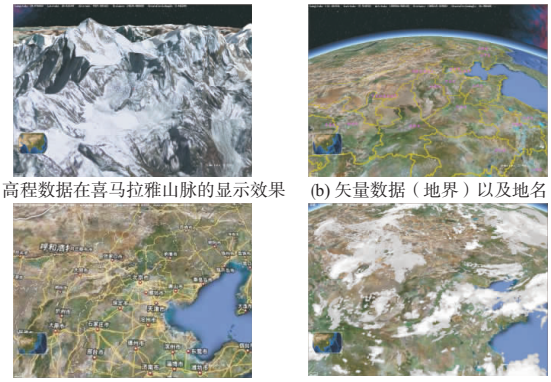
表 2　程序各个模块耗时统计　　/ms

操作类型	耗时 1	耗时 2	耗时 3	平均耗时
影像块更新	40	47	52	46.3
高程加载	318	323	359	333.3
影像块渲染	15	16	21	17.3

由实验数据可见,影像块的渲染耗时比较稳定,保持在 30 ms以下,使用独立的线程加载高程数据后,影像块的更新过程大大缩短,并且并不是每一次渲染均需要更新影像块。即当前场景更新后,会多次渲染,所以此时更新线程并不会阻塞渲染线程。

为提高系统性能,本文将数字地球的渲染频率定为30 ms/次,渲染线程流畅,帧速始终控制在 30 ~ 32 fps。影像块按照墨卡托投影方式,四叉树模型首层仅有 1 个节点(即 1 个四叉树模型),高程数据、矢量地界、地名均按照正交投影方式,高程和矢量地界都包含 9 × 18 个首层节点,地名包含 5 × 10 个首层节点。这些四叉树模型的构建方式是统一的,只需在构建不同数据四叉树模型时指定投影方式以及首层四叉树节点行列数,从而体现了各种数据与影像数据的动态匹配。通过指定两个构建参数,随时可以为影像层匹配其他数据层,如 Google 提供的矢量图片层(与影像层格式一样)。此实验中仅使用 NASA 提供的高程数据,通过正交投影,共有 9 × 18 个四叉树模型,如有需要,在程序中可指定投影方式和四叉树首层节点行列数,为影像层匹配其他不同格式的高程数据。

这种动态匹配方案具有通用性,不同的数据层通过该匹配方案均可准确地匹配影像层,系统效果如图 7 所示。



(a) 高程数据在喜马拉雅山脉的显示效果　(b) 矢量数据(地界)以及地名

(c) Google提供的图片格式的矢量数据　(d) 云层效果(云层是不断流动的)

图7 系统展示效果

4　结束语

本文提出了一种影像数据与高程数据动态匹配的方法,即使在两者使用不同投影方式建模四叉树块且非一一对应时也能够准确加载并正确绘制。考虑到了当高程数据不存在时,该层地形块使用父层(或祖层)节点的高程数据,从而达到良好的渲染效果。通过将耗时的高程数据加载从地形块的更新模块中分离出来异步进行,提高了系统的效率,改善了渲染的流畅性。该方法已经应用到了自主开发的数字地球系统中,对于影像及其匹配的高程绘制,能够满足用户实时浏览的需要,并表现出了良好的性能。本文可以进一步改进的地方:除影像数据外,其他数据量不是很大,系统效果仍可以改善;程序的各个模块还可以进一步完善,从而缩短耗时,使程序运行更加高效。这些是笔者下一步努力的方向。

参考文献:

[1] GORE A. The digital earth understanding our planet in the 21st century[R]. [s. l.]:Open GIS Consortium,1998.

[2] LINDSTROM P,KOLLER D,RIBARSKY W,*et al*. Real-time,continuous level of detail rendering of height fields[C]//Proc of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. New York:ACM,1996:109-118.

[3] 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.

[4] POLIEARPO F,OLIVEIRA M M,COMBA J L D. Real-time relief mapping on arbitrary polygonal surfaces [C]//Proc of Symposium on Interactive 3D Graphics and Games. New York:ACM,2006:155-162.

[5] 张桀宁,李帅. 一种基于顶点纹理的 LOD 地形渲染算法[J]. 系统仿真学报,2008,20(7):1758-1764.

[6] 石祥滨,武凯化,王越等. 一种视点相关的 LOD 地形生成算法[J]. 小型微型计算机系统,2010,31(2):312-316.

[7] LUO Jian-xin,NI Gui-qiang,CUI Ping,*et al*. Quad-tree atlas ray casting:a GPU based framework for terrain visualization and its applications[C]//Proc of the 24th International Conference on Computer Animation and Social Agents. 2011:74-85.

[8] GOSWAMI P,MAKHINYA M,BOSCH J,*et al*. Scalable parallel out-of-core terrain rendering[C]//Proc of Eurographics Symposium on Parallel Graphics and Visualization. 2010:63-71.

[9] AMMANN L,GÉNEVAUX O,DISCHLER J M. Hybrid rendering of dynamic heightfields using ray-casting and mesh rasterization[C]//Proc of Graphics Interface. Toronto,Ont:Canadian Information Processing Society,2010:161-168.

[10] LUO Jian-xin,NI Gui-qiang,HU Gu-yu,*et al*. Spherical projective displacement mesh[C]//Proc of the 12th International Conference on CAD/Graphics. 2011:111-118.

[11] ULRICH T. Rendering massive terrains using chunked level of detail control[C]//Proc of SIGGRAPH. 2001.