面向大规模地形的瓦片调度与实时绘制算法

刘 浩[1,2,3] 曹 巍[4] 赵文吉[1,2,3] 宁方馨[1,2,3] 潘李亮[5]

(首都师范大学资源环境与旅游学院 北京 100048)[1]

(北京市城市环境过程与数字模拟重点实验室-省部共建国家重点实验室培育基地 北京 100048)[2]

(三维信息获取与应用教育部重点实验室 北京 100048)[3]

(中国科学院地理科学与资源研究所 北京 100101)[4] (北京合歌科技有限公司 北京 100085)[5]

摘 要 随着图形硬件性能的提升,大规模地形绘制的主要瓶颈已从绘制能力不足转变为大数据的传输,针对这一问题提出一种支持大规模地形的瓦片调度与实时绘制算法。将超大地形数据以瓦片金字塔形式存储于硬盘,绘制每一帧时只调度当前场景所需的少量瓦片进入显存。首先利用GPU实时计算地形网格点的地理坐标并传回CPU分析可见范围,然后采取瓦片四叉剖分、规则化处理和瓦片合并等一系列操作在所有LOD层中拣选最优瓦片集合并调入内存,在内存中利用一块固定大小的缓存进行管理与更新,并最终以单张纹理的形式传入显存进行采样和绘制。实验表明,该算法节约了大量的显存带宽,有效提升了系统在数据传输方面的执行效率,在大规模地形调度与绘制中取得了较好的效果。

关键词 大规模地形绘制,动态调度,Virtual Texture,瓦片选取,瓦片管理,GPU

中图法分类号 TP391 **文献标识码** A

Dynamic Scheduling and Real-time Rendering Method for Large-Scale Terrain

LIU Hao[1,2,3] CAO Wei[4] ZHAO Wen-ji[1,2,3] NING Fang-xin[1,2,3] PAN Li-liang[5]

(College of Resources Environment and Tourism, Capital Normal University, Beijing 100048, China)[1]

(Urban Environmental Processes and Digital Modeling Laboratory, Beijing 100048, China)[2]

(Laboratory of 3D Information Acquisition and Application, Beijing 100048, China)[3]

(Institute of Geographic Sciences and Natural Resources Research, Beijing 100101, China)[4]

(Beijing Heegle Co. Ltd, Beijing 100085, China)[5]

Abstract Along with the development of the graphics hardware, the transmission of large-scale data has replaced the lack of rendering capacity as the new bottleneck of large scale terrain rendering. Aiming at this problem, we present a new approach for dynamic scheduling and real-time rendering for large-scale terrain. Terrain data are stored as tile-pyramid in hard disk, in real-time rendering phase, only a few of tiles are streamed into the video memory every frame. First, GPU computes and transfers geographic coordinates of the current scene into CPU in real time, CPU analyses and calculates the visible region through these coordinates, and then selects the optimum combination of visible tiles(visible tiles set) among all the layers by quad-tree generating, tiles regularization and tiles merging. The visible tiles set are streamed into CPU, managed and updated in a fix-sized buffer, and finally transferred into GPU as a texture for sampling and rendering. Experiments show that this algorithm saves a large amount of video memory bandwidth, improves the efficiency of data transmission effectively, and obtaines a good result in the large scale terrain scheduling and rendering.

Keywords Large-scale terrain rendering, Dynamic scheduling, Virtual texture, Tiles determination, Tiles management, GPU

1 引言

大规模三维地形实时绘制在地理信息系统、虚拟现实、三维游戏等领域有着广泛的应用,庞大的地形、纹理数据和系统有限的实时处理能力往往成为大规模地形实时绘制的瓶颈,近几十年来,国内外学者针对该问题进行了大量的研究。早期的地形绘制算法主要围绕层次细节方法(LOD)展开[1-3],该类算法专注于削减每一个不必要渲染的三角形,通过减少送入渲染管线的三角形数量来提高帧率,主要解决的是图形硬件有限的绘制能力同大规模数据庞大的数据量之间的矛盾。

本文受国家自然科学基金项目(41001300)资助。

刘 浩(1985—),男,博士生,主要研究方向为三维地理信息系统,E-mail:lihaha1985@126.com;曹 巍(1982—),男,博士,助理研究员,主要研究方向为三维地理信息系统;赵文吉(1967—),男,教授,博士生导师,主要研究方向为遥感技术与应用;宁方馨(1988—),女,硕士生,主要研究方向为遥感技术应用;潘李亮(1981—),男,主要研究方向为虚拟现实、地理信息系统。

随着图形硬件的发展，当今的图形处理器已经能够一次性处理大量的三角形，但是数据的传输速度(特别是外存到内存的数据传输)却没有得到相应的发展，很难与 GPU 的高速吞吐能力相适应，使得 GPU 常常要处于等待状态，因此，超大数据的传输是当前大规模地形实时绘制的主要瓶颈。此外，如何平衡使用 CPU 和 GPU 更加有效地利用系统资源，也是影响地形渲染效率的重要因素。

2 相关研究

超大规模的原始数据无法一次性载入系统内存，因此需要根据具体需求分批次载入，目前最为常用的是基于外存(out-of-core)方法。Pajarola 等[4]最先采用 out-of-core 的思想，将原始地形数据存储于外存，在绘制阶段按需调度，其不足是数据更新困难；Lindstrom 等[5]提出了一个基于外存绘制和管理海量地形数据的通用框架，但其性能依赖于操作系统虚拟内存的大小，缺乏灵活性；戴晨光[6]基于瓦片金字塔模型和目标瓦片搜索算法，实现了局部地形数据的动态更新；李胜等[7]通过构建视线方向上的增量地平线随时更新可见性信息，控制无用数据页面载入，减少了需要调度的数据量；Li[8]设计了一种内存空间分配算法来平衡内存空间和调度时间，使绘制速率能够稳定。这类算法存在两个方面的不足：一是调度的数据量大，导致宽带开销较大；二是 CPU 计算量较大，影响系统的整体绘制效率。

此外，一些优秀的纹理管理方法被引入地形数据的处理，最早由 Willem[9]基于 Mipmap 技术提出了用于地形数据划分的 Geomipmap 技术，该方法对每一个地形纹理创建一系列 Mipmaps，根据视点远近使用不同分辨率的地形纹理，能够一定程度上减少内存的消耗，但其所有层整体载入内存使得驻留内存的数据过大；Losasso 等基于 Clipmap[10]技术提出了 GeoClipmap[11]算法，即在 Mipmap 的基础上选择一个裁剪中心，将裁剪中心周围一定范围内的纹理通过裁剪装入内存；Asirvatham 等[12]对 GeoClipmap 算法进行了改进，使用了顶点纹理技术，将 CPU 中动态增量更新的顶点数据以及顶点索引计算工作移植到 GPU，减轻了 CPU 的计算负载。Clipmap 的优点在于不需要把整个影像金字塔装入内存，而是通过裁剪装载更少的数据，但驻留于内存和显存的数据仍然较大。Megatexture[13]和 Virtual Texture[14-16]是继 Clipmap 后出现的比较新的超大纹理管理方法，二者的基本思想是将大块贴图分割成均匀的小块，将场景中所有要用到的贴图全部放在一张超大的贴图上存储于硬盘，在绘制每一帧时，只把屏幕上需要显示的贴图动态地从硬盘载入显存进行绘制，以此来最大限度地降低宽带的开销。

源于 Virtual Texture 的思想，本文设计并实现了一套面向大规模三维地形场景的地形瓦片动态调度与实时绘制的框架，并对地形瓦片的选取与调度、地形瓦片的管理以及地形网格采样等关键步骤进行了重点研究和实现。

3 算法描述

本文算法将场景中所需的 DEM 和 DOM 均以纹理的形式组织，并采用同一套机制进行管理和调度。设计思路如图 1 所示，场景中所有要用到的 DEM 和 DOM 数据以瓦片金字塔的形式存储于硬盘。GPU 将实时生成的一张包含当前场景地形坐标的纹理传回 CPU，在 CPU 中求取可见区域，进而设计一种瓦片选取机制，选取结果为一张地形瓦片列表，将列表中的瓦片从硬盘载入到内存，并利用一块固定大小的 CPU 缓存来存储、管理及更新该瓦片集合，然后将缓存中的瓦片集合转换为一张纹理(缓存纹理)送入 GPU 进行采样，同时从 CPU 传入的还有瓦片集合中每一个瓦片在世界空间中的地理坐标和范围以及每一个瓦片在缓存纹理中的偏移信息。在 GPU 中着色器将根据这些信息计算每一个地形网格点应该在哪一个瓦片中采样并计算最终的纹理采样坐标，完成地形的绘制。地形采样网格为预先生成的具有连续 LOD 的圆形网格，渲染时从硬盘载入显存。

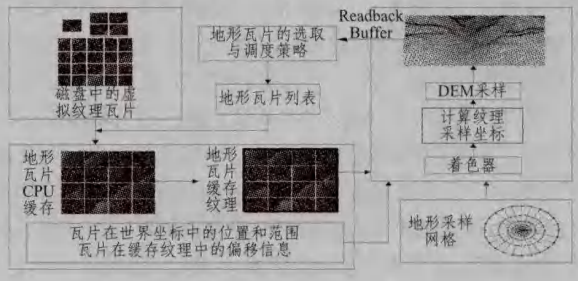

图 1 本文算法流程图

3.1 地形瓦片的选取与调度

本文算法结构中的一些独特性使得现有地形调度算法不能很好地适应，需设计新的与之匹配的调度策略。随着三维地形场景中视点的更新，系统将通过一定的拣选策略，从所有 LOD 层中找出当前渲染所需的瓦片，并判断瓦片是否已经存在于缓存中，若不存在则需要从硬盘中载入，利用所有拣选的瓦片对缓存纹理进行更新，然后将该缓存纹理送入 GPU 进行采样与渲染。该调度算法中，相对于磁盘中海量的瓦片数据，每一帧绘制只调入当前屏幕渲染所需的少量瓦片，因此瓦片拣选的效率、选取结果的质量是影响最终渲染效率的一个重要因素，本文将瓦片选取的整个过程分为 4 步完成。

第 1 步 可见区计算。地形瓦片的选取是以当前渲染窗口中的可见区域为范围来判断需要哪些瓦片，因此第 1 步是获取当前可见区的范围。传统的方法是在 CPU 中利用视锥体与地形求交，这会对 CPU 的计算负载造成很大压力，因此本文采用在 GPU 的像素着色器中以离屏渲染的方式(Render To Texture)将当前地形场景中可见区域内每一个地形顶点的地理坐标(x,y)直接渲染到一张浮点纹理上(见图 2)，然后将该纹理传回 CPU 中进行分析。

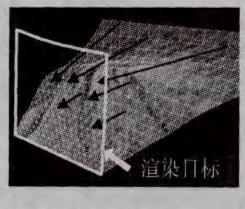

图 2 通过 RTT 获取地形坐标

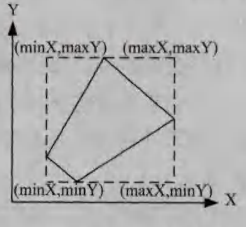

图 3 可见区最小包围框

在 CPU 中，通过循环判断求出该纹理中存储的所有地理坐标的最值，包括 $\min X$，$\min Y$，$\max X$，$\max Y$ 4 个值，本文在

计算可见区时采用以最小包围框代替真实可见区的方法，如图3所示，最小包围框由(min$X$, min$Y$)，(max$X$, min$Y$)，(max$X$, max$Y$)，(min$X$, max$Y$) 4个顶点构成。事实证明该方法是可行的，主要原因有：(1)大大简化了可见区的计算，进而有效简化了瓦片可见性的判断；(2)最小包围框中未在真实可见区内的瓦片，恰好可以作为地形漫游时的缓冲数据，能够避免因载入不及时造成的卡顿现象；(3)渲染时进行视锥裁剪，多余的数据并没有绘制，因此不影响绘制帧率。该方法充分利用了GPU的资源，能够减轻CPU的计算负载，有效简化了可见区的计算。

第2步　瓦片四叉剖分。求出可见区范围后，并不是将瓦片金字塔中所有可见区内的瓦片全部载入，而是离视点越近选取分辨率越高的瓦片(见图4)。瓦片的四叉剖分这一步即是在每一层中根据一定的阈值条件判断哪些瓦片需要进一步细分，逐层完成所有层的瓦片剖分，并将所有进行剖分的瓦片添加到瓦片列表，该列表即为初步筛选出的所有LOD层中的可见瓦片集合。具体操作时，从第0层即最低精度层开始逐层往下判断，根据阈值条件对该层中可见瓦片进行四叉剖分，如果一个瓦片没有被选中，那么它的所有子节点瓦片都不会被选中，因此只需对当前层中满足阈值条件的瓦片向下剖分，如此循环，直到剖分至最高精度层，其过程如图5所示。

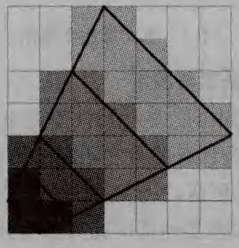

图4　多分辨率瓦片选取

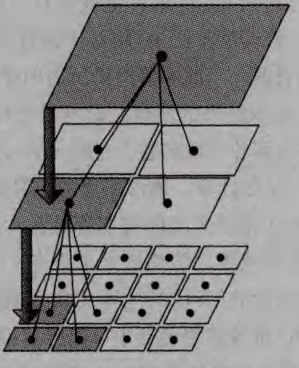

图5　瓦片四叉剖分

第3步　规则化处理。经过第一步瓦片四叉剖分得到的瓦片中，存在很多父节点与子节点覆盖范围完全重叠的瓦片，父节点和4个子节点均被选取。由于在采样时覆盖范围重叠的区域只会采样一次，即只采样精度最高的瓦片，因此当父节点与4个子节点均被选取时，父节点完全是多余的，所以采取将父节点删去的方法来避免冗余，称此过程为瓦片的规则化处理，如图6所示。

由于缓存中仅保存当前渲染所需的少量瓦片，根据设定的缓存大小可计算出缓存中最多能容纳的瓦片数，因此最终选取的存入缓存的瓦片数必须小于或等于MaxofTiles。当完成第1步四叉剖分和第2步瓦片规则化处理后瓦片数仍然超过时，需要在保证满足渲染需求的前提下进一步减少瓦片的数量，本文采取的措施为合并最远处瓦片。

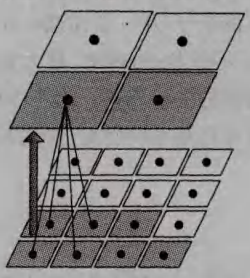

图6　规则化处理

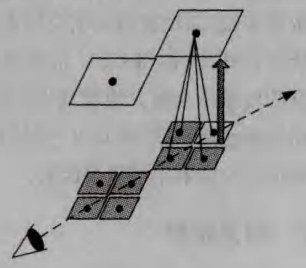

图7　合并最远处瓦片

第4步　合并最远处瓦片。找出距离视点最远的瓦片，其与关联瓦片(即由同一父节点剖分产生的其余子节点)合并成父节点，即把子节点从列表中删去，把父节点加入(见图7)，由于离视点越远其细节要求越低，因此合并最远处的瓦片对整个场景的渲染并无太大影响。第0层即最低精度层包含了整个场景范围，将其始终驻留于GPU中，以保证任何时刻都不会出现数据空缺区。

合并操作不涉及到第0层，通过循环判断找出列表中离视点最远的瓦片，然后计算其关联瓦片，在列表中其关联瓦片的个数可能为0～3个，但不管关联瓦片是否存在，该合并操作将使瓦片数始终朝着减少的趋势发展。合并过程一直持续到某一次合并后瓦片数小于或等于MaxofTiles时为止，则地形瓦片的选取完毕。此处应注意的一点是，当完成一次最远处瓦片合并之后，瓦片集合的结构发生改变，可能再次出现父节点与4个子节点同时被选中的情况，因此需要在每次合并完成之后进行一次规则化处理。

3.2　地形瓦片的管理

地形瓦片的管理是在CPU中开辟一个固定大小的缓存中完成的。缓存一般根据系统的需要预先设定为方形或矩形，边长为瓦片大小的整数倍，与磁盘中的瓦片一样，缓存同样被切分为$m \times n$等分，每一等分均与瓦片大小一致，以保证任意一个瓦片都可以存储于缓存中的任意一个等分中。瓦片装载到CPU缓存时不需要排序，只需按照列表的顺序依次装载即可。

在CPU中要完成的一个重要任务是怎样管理一个按照任意顺序装载的瓦片集合，这将直接关系到缓存纹理传入GPU后纹理采样的计算。考虑到每一个地形瓦片在世界空间中都有自己的位置和范围，并且可以通过瓦片的LOD层级和行列号计算得到，而瓦片在缓存纹理中的偏移坐标可在瓦片装载时记录，从而形成瓦片地理坐标与缓存纹理中偏移坐标的一一对应关系。该对应关系同缓存纹理一起传入GPU，着色器在进行地形网格采样时根据瓦片的地理范围来决定地形网格点应该在哪一个瓦片进行采样，并同时根据该瓦片的偏移信息计算最终的纹理采样坐标。该方法能够简化缓存中瓦片的管理，并且提高地形网格采样的效率。

图8(a)为瓦片在世界坐标中的地理位置和范围，图8(b)为瓦片在缓存纹理中的偏移位置和范围，此处缓存纹理的坐标是归一化的坐标，即范围为[0,1]。瓦片地理位置的计算是基于瓦片的命名方式"*LOD*_*RowID*_*ColumnID*"的，瓦片左下角地理坐标和瓦片宽高的计算公式如下：

$$\begin{cases} w=h=D(lod) \\ X_{pos}=RowID\times w \\ Y_{pos}=ColumnID\times h \end{cases} \tag{1}$$

式中，$D(lod)$为该层中瓦片边长所代表的实际距离。

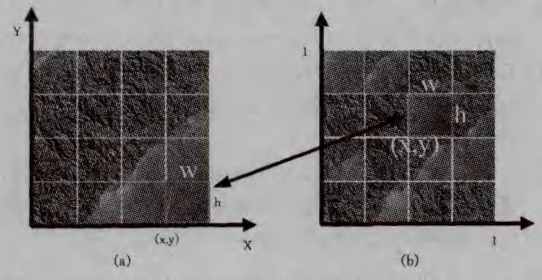

图 8　瓦片的地理坐标与偏移坐标的对应关系

根据瓦片装载时记录的瓦片行列号(f_x, f_y)和瓦片在缓存纹理中归一化处理后的宽高值(w_{tw}, h_{tw})，可得瓦片在缓存纹理中的偏移坐标计算公式如下：

$$\begin{cases} w_{tw}=h_{tw}=D_{virtual} \\ X_{offset}=f_x\times w_{tw} \\ Y_{offset}=f_y\times h_{tw} \end{cases} \tag{2}$$

$D_{virtual}$为缓存纹理坐标经归一化处理后瓦片的边长。

在将瓦片位置和偏移信息传入 GPU 之前，有一个很重要的步骤，就是对这两组信息进行一个排序，将精度高的瓦片放前面，这样做的好处是能够保证当不同 LOD 层的瓦片覆盖范围重叠时，我们选取的是精度最高的瓦片进行采样，也只有这样做才符合本文算法设计的初衷。

3.3　地形网格采样

我们采用一个圆形的地形平面网格进行采样，圆形网格的中心位于相机的位置，离视点越近网格越密提供的细节就越多，随着距离的增加，网格顶点的间距不断增大，而且从视点位置出发，各个方向上的网格疏密变化程度是一致的，因此漫游时即使视线方向任意改变，绘制的帧数都比较稳定，并且连续的网格可避免地形裂缝的产生。

DEM 和 DOM 的纹理采样分别在顶点着色器和像素着色器中完成，首先是通过地形网格点的水平位置判断该点落在哪一个地形范围内，然后在缓存纹理的瓦片集合中检索该覆盖范围的最高精度瓦片，接下来最重要的一步是计算网格点的纹理采样坐标。

如图 9(a)所示，当检索到所需瓦片后，根据采样点的水平位置以及从 CPU 传入的瓦片左下角坐标、瓦片宽和瓦片高，可得到缓存纹理中归一化处理后，采样点在瓦片内部的纹理坐标(X_{in}, Y_{in})的计算公式如下：

$$\begin{cases} X_{in}=(X_{pos}-X_{Tile})/w_{Tile}\times w_{tw} \\ Y_{in}=(Y_{pos}-Y_{Tile})/h_{Tile}\times h_{tw} \end{cases} \tag{3}$$

式中，w_{tw}和h_{tw}为缓存纹理坐标归一化处理后的瓦片宽高，由 CPU 传入。

由图 9(b)可得到，地形网格点最终的纹理采样坐标(X_{tw}, Y_{tw})计算公式如下：

$$\begin{cases} X_{tw}=X_{offset}+X_{in} \\ Y_{tw}=Y_{offset}+Y_{in} \end{cases} \tag{4}$$

式中，X_{offset}，Y_{offset}为缓存纹理中瓦片的偏移坐标，由 CPU 传入。

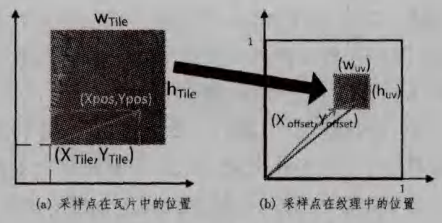

图 9　纹理采样坐标的计算

DEM 和 DOM 纹理采样的计算方法一致，体现了本算法对地形数据和纹理数据统一管理和渲染的技术特点。

4　实验结果与分析

实验程序是在 VS2010 环境下使用 C++和 Direct3D 完成的，使用 HLSL 进行 GPU 编程。硬件配置为 IntelE8500，2 G RAM，Nvidia Geforce 9800CTX。

实验数据为四川省部分区域的 DEM，分辨率为 30m，像素尺寸为 8000×8000，数据大小为 1.16G，DOM 采用 SPOT4 遥感影像数据制作，分辨率为 10m，像素尺寸为 24000×24000，原始大小为 1.8G。在预处理阶段将 DEM 和 DOM 一起进行瓦片金字塔处理，将 DEM 和 DOM 的瓦片大小对于渲染帧率的影响进行了测试，即为 128×128，DOM 瓦片大小为 384×384，金字塔共 7 层。在 CPU 中开辟一个 6×6 的 DEM 瓦片大小的缓存，即边长为 128×6=1024 像素的缓存。

该实验的实测数据折线图如图 10 所示。由于地形场景绘制采用圆形的地形网格模型，从视点位置出发各个方向上的网格疏密变化程度是一致的，因此漫游时即使视线方向任意改变，绘制的帧数都比较稳定，并随着视点位置和观察方向不断地调整自身，使三角地形网格始终满足观察的需要，帧率保持在 90fps 左右。CPU 由于要完成瓦片的管理以及外存与内存、内存与显存的数据交换，因此在需要加载新的瓦片时数据调度会使 CPU 占用率呈现一个较高的值，但是由于避免了 CPU 实时生成地形网格的计算，使得 CPU 平均占用率处于一个较低的水平。

图 11 是漫游过程中的一个场景截图。整个漫游过程中，外存与内存、内存与显存之间能够实时快速交换数据，画面平滑流畅，能够很好地满足大规模地形场景的实时绘制与漫游的需求。

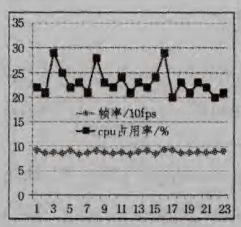

图 10　实测数据折线图

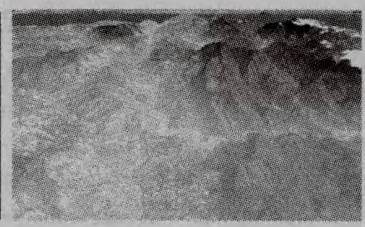

图 11　场景截图

由表 1 可以看出，在大规模地形场景绘制应用中，本文算法的宽带开销非常低，帧率要比传统 CPU 算法和 Geoclipmap 算法高很多。

表 1　本文算法与传统 CPU[17]算法和 Clipmap[18]算法的对比

算法	宽带开销/帧	平均帧率
CPU	712.45k	32fps
Geoclipmap	348.36k	62fps
本文	132.24k	91fps

结束语 本文针对当前大规模地形绘制遇到的大数据传输的问题，提出一种支持大规模地形数据的动态调度和绘制算法。利用 GPU 的实时计算和 CPU 的动态分析，在绘制每一帧时只载入当前渲染所需的极少部分瓦片，并利用地理数据的特性简化了地形块的管理和采样。算法简化了数据的交换并且改进了内存的管理，能够有效减少宽带的开销，对于大数据的传输具有很好的执行效率。目前该算法的改进方向是把更多的计算任务交由 GPU 来完成，以释放更多的 CPU 资源。我们下一步工作的重点一是把瓦片的管理移植到 GPU 进行，二是利用 GPU 完成瓦片的选取。

参 考 文 献

[1] Duchaineauy M. ROAMing Terrain: Real-time Optimally Adapting Meshes[C]// Proceedings of IEEE Visualization'97. 1997: 81-88

[2] Rottger S. Real-time generation of continuous levels of detail for height fields [C]// Proceedings of W-SCG'98. 1998: 315-322

[3] Turner B. Real-Time Dynamic Level of Detail Terrain Rendering with ROAM[EB/OL]. http://www.gamasutra.com/features/20000403/turner_01.htm, 2007-04

[4] Pajarola R. Large Scale Terrain Visualization Using the Restricted Quadtree Triangulation[C]// Proceedings of IEEE Visualization'98. 1998: 19-26

[5] Lindstrom P, Pascucci V. Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization [M]. IEEE Transaction on Visualization and Computer Graphics. 2002

[6] 戴晨光，张永生，邓雪清. 海量地形数据实时可视化算法[J]. 计算机辅助设计与图形学，2004，16(11)：1603-1607

[7] 李胜，冀俊峰，刘学慧，等. 超大规模地形场景的高性能漫游[J]. 软件学报，2006，17(03)：535-545

[8] Li L, Li F, Huang T. Smooth Schedule of Large-Scale Terrain Visualization from External Memory[J]. Journal of Software. 2007, 18(sup): 26-34

[9] de Boer W H. Fast Terrain Rendering Using Geometrical Mipmapping[EB/OL]. http://www.connectii.net/emersion, 2000

[10] Tanner C C, Migdal C J, Jones M T. The Clipmap: A Virtual Miamap[C]// Proceedings of the ACM SIGGRAPH'04. 1998: 151-158

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

[12] Asirvatham A, Hoppe H. Terrain rendering using GPU-based geometry clipmaps, GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation [M]. Boston, MA: Addison-Wesley Professional, 2005: 14-30

[13] Foundation W. MegaTextures[EB/OL]. Retrieved October 10, from http://en.wikipedia.org/wiki/megatextures, 2006

[14] Barrett S. Sparse virtual textures [EB/OL]. Game Developer Conference, San Francisco, CA. http://silverspaceship.com/src/svt, 2008

[15] Neu A. Virtual texturing[D]. CoRR, abs/1005.3163, 2010

[16] van Waveren J M P. id tech 5 challenges-from texture virtualization to massive parallelization[EB/OL]. http://s09.idav.ucdavis.edu/talks/05-JP_id_Tech_5_Challenges.pdf, 2009

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

[18] 白皓，龚光红，丁莹. 基于 Clipmap 的大规模地形可视化技术研究[J]. 中国体视学与图像分析，2009，14(2)：202-208

(上接第 89 页)

了本算法的高斯加权方式的合理性和正确性。但是高斯加权只是一种简单的加权方式，还可能会存在更好的加权方法，并且引入高斯加权改进后增加了算法的时间复杂度，因此下一步还需要对新算法进行更深入的探索和研究，以更好地解决新算法的时间复杂度和加权改进方式等问题。

参 考 文 献

[1] Kennedy J, Eberhart R C. Particle Swarm Optimization [A]// Proceedings of the IEEE International Conference on Neural Networks [C]. 1995: 1942-1948

[2] Eberhart R C, Kennedy J. A New Optimizer Using Particle Swarm Theory [C]// Sixth International Symposium on Micro Machine and Human Science. 1995: 39-43

[3] Shi Y H, Eberhart R C. Empirical Study of Particle Swarm Optimization [A]// Proceeding of Congress on Evolutionary Computation [C]. Piscataway, NJ: IEEE Service Center, 1999: 1945-1949

[4] Shi Y H, Eberhart R C. A Modified Particle Swarm Optimizer [C]// IEEE International Conference on Evolutionary Computation. Anchorage, Alaska, May 1998: 69-73

[5] Ratnaweera A, Halgamuge S. Self-organizing hierarchical particle swarm optimizer with time 2 varying acceleration coefficients [J]. IEEE Trans Evolutionary Computation, 2004, 8(3): 240-255

[6] Robinson A, Rahamat-Samii Y. Particle Swarm Optimization in Electromagnetics[J]. IEEE Trans. Antennas Propag., 2004: 397-407

[7] Shi Y H, Eberhart R C. Parameter Selection in Particle Swarm Optimization[C]// Annual Conference on Evolutionary Programming. San Diego, March 1998: 591-600

[8] Liu Jin-yang, Guo M Z, Deng C. GeesePSO: An Efficient Improvement to Particle Swarm Optimization[J]. Computer Science, 2006, 33(11): 166-168

[9] Xiao Z, Yuan Y, Li P Y. Learning Algorithm for Multimodal Optimization[C]// Proceedings of the ELSEVIER International Conference on Computer and Mathematics with Applications. Zhengzhou, China, June 2009: 2016-2021