ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, Supplement, December 2007, pp.26−34
© 2007 by *Journal of Software*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种大规模地形可视化的平滑调度算法*

李立杰[1+], 李凤霞[1], 黄天羽[2]

[1](北京理工大学 计算机科学技术学院,北京 100081)
[2](北京理工大学 软件学院,北京 100081)

Smooth Schedule of Large-Scale Terrain Visualization from External Memory

LI Li-Jie[1+], LI Feng-Xia[1], HUANG Tian-Yu[2]

[1](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)
[2](School of Software, Beijing Institute of Technology, Beijing 100081, China)
+ Corresponding author: Phn: +86-10-68914776, Fax: +86-10-68914776, E-mail: leahero@163.com

Li LJ, Li FX, Huang TY. Smooth schedule of large-scale terrain visualization from external memory. *Journal of Software*, 2007,18 (Suppl.):26−34. http://www.jos.org.cn/1000-9825/18/s26.htm

Abstract: The paper proposes a smooth schedule algorithm to render terrain data out-of-core. The terrain data is organized as a quad-tree with Z-order filling curve, which preserves the data continuity and improved the schedule efficiency. A schedule strategy basing memory allocation is designed to support the stable frame rate terrain rendering. Controllable schedule area, instead of the view frustum culling, is applied to achieve the balance between the in-core memory requirements and scheduling time. A pre-estimated approach is adopted to smooth the data exchange. The algorithm achieves a smooth schedule and successfully avoides the slow or jerky phenomena, usually caused by inefficient and unsmooth schedule. By compared with traditional methods, the algorithm proposed is feasible and efficient in out-of-core terrain visualization.
Key words: terrain rendering; level of detail; out-of-core; filling curve; stable frame rate

摘 要: 提出一种适于大规模地形绘制的光滑调度算法.采用四叉树分块结构和 Z 型填充曲线组织地形数据,利用地形数据的局部连续性提高调度效率;设计一种内存空间分配算法调度地形数据,实现对恒定帧速率绘制算法的支持;通过可控调度区实现调度优先级计算,在内存空间需求和调度时间需求之间取得平衡;采取预估调度的策略实现平滑调度并有效减少绘制中的缺块现象.算法实现了地形场景漫游中的数据平滑调度,有效地避免了因内外存大数据量交换而引起的显示延迟和跳跃现象.实验结果表明,利用平滑调度算法,数据调度的准确性和稳定性有了较大提高,在地形漫游过程中取得了很好的效果.
关键词: 地形绘制;层次细节;外存;填充曲线;恒定帧速率

Large-Scale terrain rendering is a challenging research and indispensable job of virtual outdoor environment visualization, as widely used in virtual battlefield environments, driving simulators, GIS (geographic information system), and flight applications. Due to the huge scope of the terrain, there are two issues: (1) the quantity of terrain

* Supported by the National Defense Technology Foundation of China under Grant No.B2220061084 (国防基础研究基金)
Received 2007-04-30; Accepted 2007-11-23

data exceeds the rendering capability of computer, which causes rendering too slowly for interactive rates; (2) the rendering process suffers from the limited size of memory and part of data have to be stored in external memory (for example, hard disk and other storage out-of-core), which results in slow data accessing and unacceptable rendering effect.

LOD (Level of details) is a primary technique of in-core rendering for large terrain. By this way, the rendering is speed up with acceptable quality. But in most applications, large-scale terrain results in frequent data exchanges between the fast memory and slow external storage. Therefore the rendering efficiency suffers from the unreasonable schedule algorithm, which has become the bottleneck of large-scale data processing.

There are several solutions [1−3] to deal with out-of-core visualization, such as the vertex-based LOD rendering and the batch-based LOD rendering. Stable frame rate rendering [4] is also a key research to guarantee high-quality roaming. To reach this target there are two conditions: stable output of in-core rendering, and smooth exchange of out-of-core scheduling. To achieve better interaction and quality, the in-core LOD simplifies terrain data to speed rendering, and out-of-core scheduling provides data for LOD rendering.

As main contributions of this paper, we propose a smooth schedule algorithm supporting stable frame rate rendering and batch LOD simplification. The scene rendering speed and effect can be strengthened by following ideas: to improve the schedule efficiency, batch terrain tiles are deal as a schedule unit and re-organized in Z-order; memory allocation is adopt to terrain schedule instead of threshold-based schedule; a pre-estimation strategy is applied to alleviate tile-lacking phenomena and smooth data exchanges.

The paper proceeds with a presentation of related work in Section 2. Section 3 introduces the terrain data pre-processing and organization. Section 4 describes the memory schedule algorithm in details, including memory allocation, priority computation and memory schedule with pre-estimation. In Section 5, comparison experiments are provided and the statistical results are analyzed. We make a discussion and present future work in Section 6.

1 Related Work

For large-scale terrain rendering, in-core LOD strategy and out-of-core data exchanges are respectively necessary to adjust a multi-resolution display and realize the smooth schedule. In this section, the research on large-scale terrain rendering is expatiated in two aspects: terrain rendering in-core and terrain scheduling out-of-core.

1.1 LOD in-core

Traditional LOD simplification algorithm[5,6] was view-dependent LOD, which controlled the LOD levels based on viewpoint and reaches extra refinement. However, the triangulation computing was time-consumed and the capability of GPU accelerations wasn’t fully utilized. Batch LOD simplification algorithms[7−11] divided huge terrain into small tiles or batches. Triangles belong to the same tile were uniformly simplified and rendered. The method combined the virtues of static LOD and dynamic LOD. Methods above compute LOD by comparing with an error threshold. So they can’t satisfy the requirements of interactive frame rate rendering.

The stable frame rate rendering[4,12−14] uses a heuristic to simplify by evaluating the complexity of scene and the capability of graphics hardware. There is no constant error threshold to control the LOD refinement. In rendering, the refinement of LOD levels can be adjusted adaptively. Thus, we can get constant frame rate rendering, which is independent of the software and hardware environment, scene complexity, roaming manners, etc. Therefore, the stable frame rate rendering is more likely to guarantee the immersion when roaming, and it is a potential research field.

1.2 Out-of-Core scheduling

Classical out-of-core visualization algorithms[1,2,15,16] were proposed with vertex-based rendering method. Lindstorm[1,2] applied file mapping of Windows OS to setup the relations between files and memory, which cost too much resources. Bao[15] stored terrain LOD information in a special page structure, which improved utilization and reduced the tile-lacking. Pajarola [16] adopt the dynamical scene management and progressive mesh to realize out-of-core rendering. The methods mentioned above improve the memory usage. However, theses algorithms are not fit for batch-based rendering.

The schedule algorithms based on batch [3,14,17−20] improved the render quality greatly as well as the memory utilization. Ulrich[3] proposed an asynchronous batched-based rendering method for out-of-core terrain data. Pouderoux[14] adopt a square area to make schedule for data. For this method need load the most refined data, accessing the great deal of data made it unsuitable for fast or flight roam. Other algorithms[17−20] directly calculate the terrain errors, and perform scheduling by comparing with a threshold. For the sake of the fixed threshold value, it is difficult to realize smooth schedule and stable frame rate rendering.

2 Data Pre-processing and Organization

In our approach, the large-scale terrain is organized as levels of pyramid, in which each level data is sampled from more refined level. Each level is partitioned into the same pixel-dimension tiled patches, and organized by quad-tree. As shown in Fig.1, each node of quad-tree refers to a piece of terrain tile. The quad-tree structure of terrain improves rendering efficiency, speeds data schedule and reduces rendering computation.

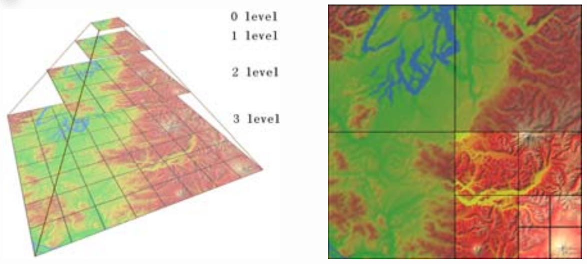


Fig.1 A quad-tree organization of terrain data

The terrain tiles are saved as implicit quad-tree, namely, all the nodes are saved in array instead of father-children pointers. According to the character of complete quad-tree, the relations between father and children can be fast computed. To improve the data continuity, terrain tiles are stored in Z-order, shown in Fig.2. In this manner, all the neighbor data can be accessed at one time. The Z-order filling curves preserve the local continuity, and reduce accessing time.

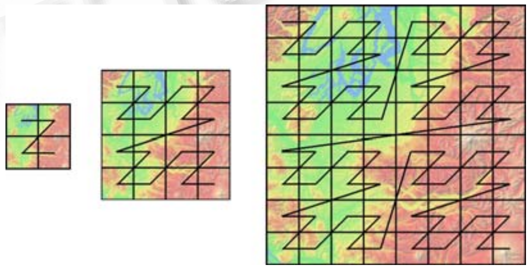


Fig.2 The Z-order filling curves for terrain data storage

In pre-process there are some variables to be computed, such as bounding box, object screen errors, normal vectors, etc. These variables are used in both scheduling process and rendering process. The static priorities, used to estimate the object space errors for terrain tile, are computed by

$$staticPriority_i = \begin{cases} \sum_{t \in triangle(i)} area_t, & i \text{ havenot children} \\ \sum_{t \in child(i)} staticPriority_t, & i \text{ have children} \end{cases} \tag{1}$$

The static priorities, presented by triangle area, are in direct ratio with terrain gurgitation and terrain level.

3 Memory Schedule Out-Of-Core

This section introduces an algorithm of memory schedule out-of-core in three aspects, including: (1) memory allocation strategy——which decides the principle of data exchanging; (2) priority computation---which decides data supposed to be loaded in memory with high priority; (3) pre-estimation schedule---which decides data candidate for exchanging.

3.1 Memory allocation strategy

In an ideal schedule algorithm, the data remaining in memory should contain both current rendering data and buffer data. In a roaming, to ensure proper levels of LOD can be loaded and rendered when need, there are two conditions required: (1) the data remaining in memory should be an enclosure of rendering data in different LOD; (2) extra data need to be rendered in few seconds should be included.

General schedule algorithms[18] directly calculate the terrain errors, and perform scheduling by comparing with a threshold. They are not suitable for stable frame rate rendering for the sake of the fixed threshold value. Another kind of algorithms[14] uses a square or round as viewpoint-centered schedule area. Ignoring terrain gurgitation, it is difficult for such area to cover all required data. Further more, inaccurate schedule data will cause serious memory shortage.

To improve the scheduling process, we propose a memory allocation schedule algorithm, in which the error threshold is unknown but the influence by terrain space error is considered. We can achieve accurate memory schedule without a fixed threshold.

In the memory allocation algorithm, a terrain tile in-core or out-of-core is determined by the available memory size. In a limited memory space, the terrain tile with highest priority, namely with most possibility to be rendered, is selected to be loaded in-core. Thus the schedule area will be an enclosure of the rendering area. The strategy is as following. First, all the available memory $C_{targetMem}$ is allocated to the root of quad-tree, and the root node allocate them to its children recursively. Secondly, make a judge for each node whether the allocated memory is enough to hold its data. If the allocated memory is not enough, the related terrain tile will not be loaded in memory. Else the related terrain tile will be loaded, and the rest memory will be allocated to its children. The allocate pseudo code can be described as follows:

```
Algorithm allocate_memory(node,availMem)
    if ($availMem<tileNeedMem$)
        if ($node\_is\_resident\_in\_memory(node)$)
            $put\_node\_in\_outTileSet(node)$;
        ruturn;
    if ($node\_is\_not\_resident\_in\_memory(node)$)
        $put\_node\_in\_scheduleTileSet(node)$;
    $availMem=availMem-tileNeedMem$;
    for (each $childNode_i \in child\_of(node)$)
        $childMem_i=allocate\_in\_child(availMem,childNode_i)$;
        allocate_memory(childNode_i, childMem_i);
    return;
```

The rest memory of each node is allocated to its children nodes according to their priority. The principle of allocation can be described as

$$allocMem_i = availMem_i \times \frac{priority_i}{\sum_{t\in sib(i)} priority_t} \tag{2}$$

Here, *allocMem*$_i$ is the memory allocated for node *i*, and *availMem*$_i$ is the available memory of its father, *sib*(*i*) is the sibling of node *i*, and *priority*$_i$ represents the schedule priority.

3.2 Priority computation

For the sibling nodes, it is necessary to compute the priorities to determine the memory allocation. Priority of node manifests the importance of the terrain in view. It is computed as following:

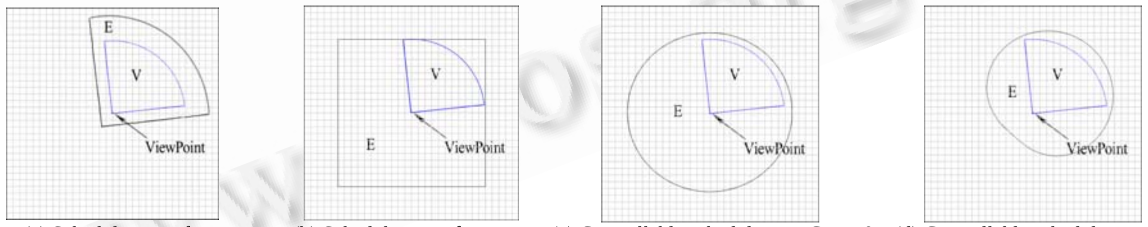
$$priority_i = \frac{staticPriority_i \times angleScale_i}{distance_i^{\,2}} \tag{3}$$

Here, *distance*$_i$ is defined as the distance from patch *i* to viewpoint. *staticPriority*$_i$, computed in pre-process, is object error independent of viewpoint. *angleScale*$_i$ is a variable about the angle between terrain tiles and the sight.

$$angleScale_i = \mathrm{MAX}(1 + (\overrightarrow{view}\cdot\overrightarrow{P-V}) \times C_{angle}), 0) \quad (C_{angle} \ge 0) \tag{4}$$

Here *P* is the center of the terrain, *V* is the viewpoint, *P*-*V* is the unit direction from viewpoint to terrain, and *view* is the direction of view. C_{angle} is a weight to determine the shape of schedule area.

In the data schedule, we adopt direction angle to decide the shape of schedule area. Comparing with the view frustum (Fig.3(a)) or square area (Fig.3(b)) with the center of viewpoint, the direction angle strategy has two advantages: In one hand, the schedule data includes those terrain tiles out of the view frustum with high priority. When the view direction changes, There will not exist frequent data exchanges. And in another hand, the shape of schedule area is controlled by C_{angle}. It can be adjusted dynamically according to the roaming routes. When C_{angle} is close to zero, the schedule area presents a round shape, which no need make any schedule when direction varying, shown in Fig.3(c). When C_{angle} increases, the schedule area will contain more accurate terrain data of the view frustum, fully using of memory space, illustrated by Fig.3(d). In controllable scheduling area, the terrain has higher resolution when closer to the viewpoint. With the distance from viewpoint increases, the resolution will become low. Along with the view direction, the resolution decreases slowly, but it goes faster with larger direction angle.



(a) Schedule area of view frustum (b) Schedule area of square (c) Controllable schedule area C_{angle}=0 (d) Controllable schedule area C_{angle}=0.5

Fig.3 Rendering and scheduling area. *V* is the rendering area, and *V*+*E* is the scheduling area

3.3 Pre-estimated schedule

With the free moving of viewpoint, the terrain data in memory will change constantly. When a terrain tile, need to be rendered, is unavailable in memory, we call it tile-lacking phenomenon. When tile-lacking happens, rendering process will send a message to the scheduling process to perform data exchange. We call such schedule as tile-lacking schedule.

There are two ways to deal with tile-lacking. One way is the synchronous schedule. The rendering process waits the scheduling process reading data out-of-core, then renders. This method guarantees the rendering quality instead of the rendering speed. Another way is asynchronous schedule. The rendering process draws the father tile with low-resolution display after sending the tile-lacking request, so as to ensure the rendering speed. These two approaches cause the jerk phenomenon or low quality. To avoid the tile-lacking, a pre-estimated schedule is proposed in this paper.

Pre-estimated schedule is a method to pre-estimate the data to be rendered after *dt* time, and to smoothly load in memory in time. It guarantees all data is available in memory when needed after *dt* time. In this way, the scheduling process reduces the tile-lacking phenomena and keeps smoothly.

The pre-estimated view range can be obtained from current viewpoint moving-speed and angle-speed. We use *dn* to replace *dt*, in order to compute the viewpoint after *dn* frames as the pre-estimated viewpoint. The schedule is performed at each frame, so as to get smooth schedule procedure within *dn* frames. For a new pre-estimated viewpoint, the data for exchanging is denoted by $scheduleTileSet = memTileSet_{new} – memTileSet_{last}$. In order to realize smooth schedule, $scheduleTileSet/dn$ of data, which need to be scheduled, is loaded averagely at each frame. The schedule order is determined by the needed urgency of tiles, denoted by

$$urgency_i = \frac{availMem_i}{tileNeedMem} \quad (5)$$

Pre-estimated schedule grantees that data in-core of each level contains data requiring to be rendered. There are two exceptions to cause tile-lacking schedule: (1) the schedule area is too small to contain all the rendering area; (2) *dn* is so large that the schedule area can’t cover all data need to be rendered in *dn* frames. If tile-lacking schedule happens in this occasion, we call this kind of schedule as urgent schedule. The scheduling process put lacking tiles in the schedule set and assigns them highest urgency priorities. This strategy ensures the priority of lacking tiles, and the same time smoothes the scheduling process.

The set of terrain tiles, which need to be changed out of the memory after *dn* frames, is defined as $outTileSet = memTileSet_{last} – memTileSet_{new}$. However, *outTileSet* may be still within the rendering area in next *dn* frames, so these tiles can’t be changed out in one time. To avoid changing wrong data, LRU (Least Recently Used) strategy should be applied. The releasing order of tiles is determined by their rendering time lately. The searching set in improved LRU is confined in a smaller range *outTileSet*, so the algorithm is performing faster. The memory releasing is synchronous to the memory allocating, so as to avoid the second-time tile-lacking.

Pre-estimated algorithm makes real-time revisal for schedule data when roaming in the whole scene. While the moving route departures from the pre-estimated route, the scheduling process will cancel last operation and make a new estimation. Using pre-estimated schedule can avoid tile-lacking phenomenon, on the other hand it realized smooth data-exchange. And in the mean time, the reduced computation cost help improve schedule efficiency.

4 Experiments and Results

We have implemented the smooth schedule algorithm with C++ and OpenGL, and performed on an Intel PIV 2.8GHz computer with 1G RAM, 128M display memory. Our experiments were tested on the terrain data——Puget Sound area, which contains 16385^2 grids, and each terrain tile are 17^2.

Fig.4 shows the path in the terrain for a fly-through. By the stable frame rate in-core rendering algorithm[4], 2049^2 grids were rendered with stable output given by Fig.5.

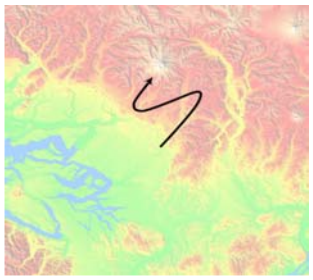


Fig.4 Path for a fly-through

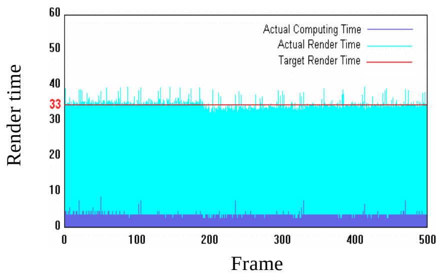
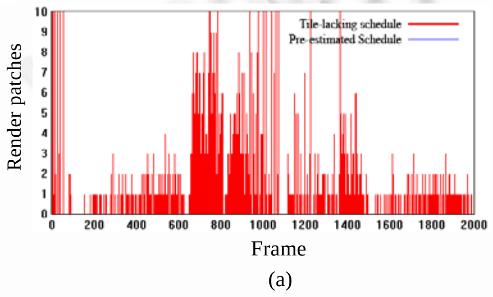


Fig.5 The statistics of render time

First the comparison was made between the traditional schedule algorithm [3] and the proposed pre-estimated schedule algorithm to prove the high efficiency and quality.

Fig.6 (a) shows the number of scheduled tiles at each frame of 2000 frames. Fig.6 (b) illustrates the total number of scheduled tiles while tile-lacking happens. Fig.7 (a) and Fig.7 (b) is the statistical results of our proposed schedule method with C_{angle} =0.1, $C_{targetMem}$ =2000 tiles. As show in Fig.7 (a), since the pre-estimation is adopted, the scheduling is smoother and the number of scheduled tiles at each frame is less and keeping stable. In Fig.7 (b), the red curve presents the tile-lacking schedule, and the blue curve presents the pre-estimated schedule. From this figure, we can see that the lacking tiles are reduced greatly. By the pre-estimated schedule method, it is ensured that the rendering quality is keeping as high as possible and the rendering speed is guaranteed.



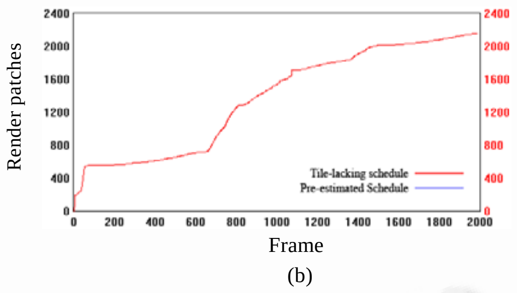
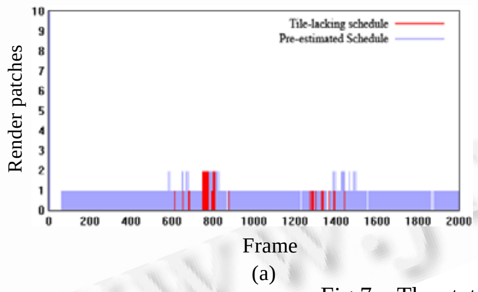


Fig.6 The statistics of unsmooth schedule



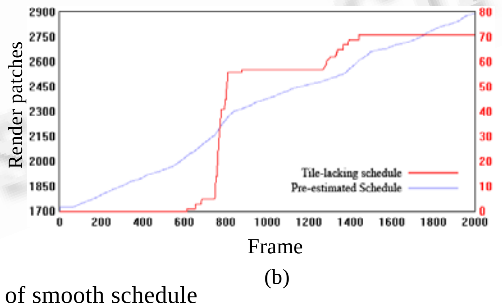
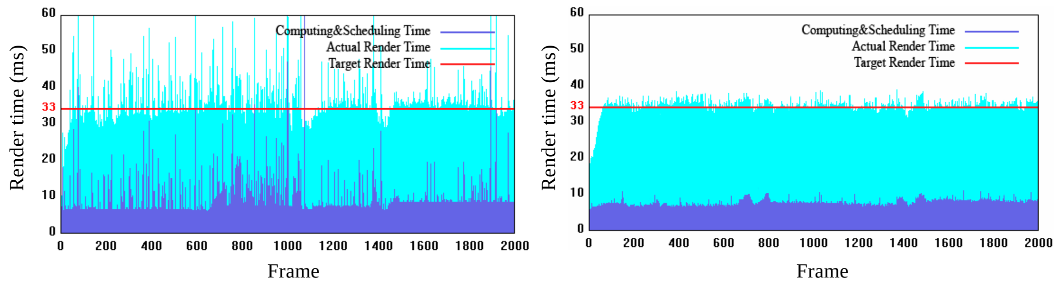


Fig.7 The statistics of smooth schedule

We made frame rate statistics for unsmooth scheduling and smooth scheduling proposed, respectively illustrated by Fig.8(a) and (b). In Fig.8(a), the rendering quality and speed affected by unsmooth scheduling. We applied the schedule with pre-estimation to ensure the stable rendering with 30 fps in Fig.8(b).

Normally the in-core simplification can get stable output (shown in Fig.5), however, affected by unsmooth terrain schedule, the rendering frame rates become unstable. Using proposed algorithm, we can get balanced rendering and scheduling with stable data supplied by schedule strategy.



(a) (b)

Fig.8 The frame rate statistics of unsmooth scheduling and smooth scheduling

5 Conclusion and Discussion

We demonstrated a pre-estimated schedule algorithm to achieve smooth exchanging and rendering large-scale terrain data. Comparing with other methods, the main contributions are following: (1) Efficient data structure. All terrain tiles are organized in quad-tree and stored as the manner of Z-order filling curves. The terrain data can be accessed efficiently and exchanged easily. (2) Memory allocation schedule. It supports stable rendering algorithm with unfixed error threshold. (3) Controllable schedule area. It gets the balance between the in-core memory requirements and scheduling time. (4) Smooth data exchange. A pre-estimated approach was adopted to smooth the data exchange. It guarantees the real-time rendering and interaction.

Our presented work supplies an effective and smooth schedule algorithm for rendering terrains out-of-core. In the future work, we are looking forward to self-adjust the schedule parameters dynamically, so as to achieve more accurate and faster terrain rendering.

References:

[1] Lindstrom P, Pascucci V. Visualization of large terrains made easy. In: Proc. of the IEEE conf. on Visualization 2001. 2001. 363–370.

[2] Lindstrom P, Pascucci V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. IEEE Trans. on Visualization and Computer Graphics, 2002,8(3):239–254.

[3] Ulrich T. Rendering massive terrains using chunked level of detail control. In: Proc. of the ACM SIGGRAPH 2001. 2002.

[4] Li LJ, Li FX, Huang TY. A time-controlling terrain rendering algorithm. In: Proc. of the 12th Int'l. Conf. of Interactive Technologies and Sociotechnical Systems. 2006. 328–337.

[5] Mark D, Murray W, *et al*. Roaming Terrain: Real-Time Optimally Adapting Meshing. In: Proc. of the IEEE Conf. on Visualization 1997. 1997. 81–88.

[6] Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proc. of the IEEE Conf. on Visualization 1998. 1998. 35–42.

[7] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R. BDAM——Batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum, 2003,22(3):505–514.

[8] Cline D, Egbert PK. Terrain decimation through quadtree morphing. IEEE Trans. on Visualization and Computer Graphics, 2001,7(1):62–69.

[9] Larsen BD, Christensen NJ. Real-Time terrain rendering using smooth hardware optimized level of detail. Journal of WSCG, 2003,11(1).

[10] Hwa LM, Duchaineau MA, Joy KI. Adaptive 4-8 texture hierarchies. In: Proc. of the IEEE Conf. on Visualization 2004. 2004. 219–226.

[11] Wagner D. Terrain geomorphing in the vertex shader. In: ShaderX2, Shader Programming Tips and Tricks with DirectX 9. Wordware Publishing. 2004.

[12] Funkhouser TA, S'equin CH. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. Computer Graphics, 1993,27:247−254.

[13] Lakhia A. Efficient interactive rendering of detailed models with hierarchical levels of detail. In: Proc. of the 2nd Int’l Symp. on 3D Data Processing, Visualization, and Transmission. 2004. 275−282.

[14] Pouderoux J, Marvie J. Adaptive streaming and rendering of large terrains using strip masks. In: Proc. of the 3rd Int’l Conf. on Computer Graphics and Interactive Techniques in Australasia and South East Asia. 2005. 299−306.

[15] Bao X, Pajarola R. LOD-Based clustering techniques for efficient large-scale terrain storage and visualization. In: Proc. of the SPIE Conf. on Visualization and Data Analysis 2003. 2003. 225−235.

[16] Pajarola R. Large scale terrain visualization using the restricted quadtree triangulation. In: Proc. of the IEEE Conf. on Visualization 1998. 1998. 19−26.

[17] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R. Planet-Sized batched dynamic adaptive meshes (p-bdam). In: Proc. of the IEEE Conf. on Visualization 2003. 2003. 147−154.

[18] Reddy M, Leclerc Y, Iverson L, Bletter N. Terravision II: Visualizing massive terrain databases in VRML. IEEE Computer Graphics and Applications. 1999,19(2):30−38.

[19] Schneider J, Westermann R. GPU-Friendly high-quality terrain rendering. Journal of WSCG, 2006,14(1):49−56.

[20] Wu JZ, Liu XH, Wu EH. Real-Time rendering for out-of-core terrain model. Journal of Computer Aided Design & Computer Graphics, 2005,17(10):2196−2202 (in Chinese with English abstract).

附中文参考文献:

[20] 吴金钟,刘学慧,吴恩华.超量外存地表模型的实时绘制技术.计算机辅助设计与图形学学报,2005,17(10):2196−2202.

LI Li-Jie was born in 1977. He is a Ph.D. candidate at the School of Computer Science and Technology, the Beijing Institute of Technology. His current research interests include virtual reality and scene simulation.

HUANG Tian-Yu was born in 1979. She is a docent at the School of Software, the Beijing Institute of Technology. Her research areas are 3D human animation and simulation.

LI Feng-Xia was born in 1953. She is a professor at the School of Computer Science and Technology, the Beijing Institute of Technology. Her research areas are virtual reality and computer simulation.