An Effective Visualization Method for Large-scale Terrain Dataset

HANG QIU[1,2,3] LEI-TING CHEN[1,3] GUO-PING QIU[2] HAO YANG[1,2,3]
[1] School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu, CHINA
[2] School of Computer Science, University of Nottingham, Nottingham, UNITED KINGDOM
[3] Provincial Key Laboratory of Digital Media, Chengdu, CHINA
qiuhang@uestc.edu.cn

Abstract: -Visualization of very large digital elevation models plays an important role in many applications ranging from computer games to professional Geographic Information System (GIS). However, the size of terrain datasets is much larger than the capacity of main memory, which brings up serious challenges for the interactive visualization of massive terrain. In this paper, an effective visualization method for large-scale terrain is proposed. In preprecessing stage, the full resolution terrain dataset is divided into several uniform blocks. For each block, a LOD hierarchy, which contains different resolution of elevation data, is constructed based on quadtree-split. In order to save the external memory, a non-redundant storage method is put forward. Different from the traditional pyramid model, we stored only the newly added vertices in each level. The data quantity during real-time rendering decreased effectively due to the sight-line-dependent screen error computation, view frustum culling and detailed culling scheme. To ensure the continuity of navigation, a pre-loading scheme, which based on the view frustum extension, is presented and implemented. Experiments show that compared with traditional methods, our method has reduced the external memory cost by 60% and achieved good visual effect with high rendering speed.

Key-Words: - Levels of Detail (LOD), Model Error, Static Error, Multi-resolution model, Quadtree

1 Introduction

In recent years, with the development of remote sensing, digital photography and other measuring technologies, we can get digital elevation models (DEMs) and the corresponding photo textures with high resolution easily. These original datasets bear a great application value in many domains, including geological disaster assessment [1], flight simulation [2], military simulation [3] and visualization of complex environments [4]. However, due to the ever-increasing size of the resolution of scanned DEMs and the corresponding photo textures, visualization of terrain is facing more and more problems of dealing with large-scale datasets. Massive data of terrain can easily exceed the capacity of main memory. Thus, the efficiency of terrain rendering is affected seriously due to the data exchange between main memory and the external memory, which is a common bottleneck of large-scale data processing.

Extensive works made efforts to manage the complexity of huge terrain model by using view-dependent level-of-detail (LOD) rendering algorithms [5]. LOD algorithms apply a hierarchy of mesh refinement operations to adapt the surface tessellation. Thus, it could reduce the complexity of the terrain model and keep a nice visual result. Lindstrom et al. [6] proposed a dynamic multi-resolution terrain algorithm, which used regular grid to describe a continuous level of detail model and define the maximum error of projected image by a variable threshold in screen space. Duchaineauy et al. [7] presented ROAM (Real-time Optimal Adaptive Meshes) algorithm, which adopted a binary tree over the set of triangles and two queues are maintained for splitting and merging operations. Zhang et al. [8] put forward a novel terrain model simplification method based on terrain features, using the idea of discrete particle swarm algorithm. Losasso et al. [9] introduced geometry clipmaps, which is a multi-resolution, fixed memory size scheme for efficient representation and rendering of large terrains, and it provides good rendering performance, but the representation does not lend itself to editing or modification of the terrain. In 2009, Livny et al. [10] gave a new approach for GPU-based terrain rendering. Some other LOD algorithms and more specific surveys can be found in [11]. The methods based on view-dependent LOD are widely applied that one can represent less important areas with fewer amounts of polygons. They effectively reduce the total number of primitives to be rendered while achieving good visual quality of terrains.

Nevertheless, when the size of simplified model also exceeds the capacity of main memory, and then the out-of-core schemes must be considered. Many researchers have concentrated on the data structures and algorithms of out-of-core terrain visualization. These strategies focused on the cache management of data, which resolves the bottleneck that takes much time when massive data is transferred from the external memory to the main memory. Lindstrom et al. [12, 13] proposed out-of-core visualization of large-scale terrain relies on an approach of view-dependent simplification. The works in [14-18] adopted pyramid model, which linearizes multi-resolution terrain data and stores them into the external files. However, the above-mentioned algorithms have numerous redundant data in the terrain model of large regions and increase the cost of the external memory.

In this paper, we proposed a visualization method for large-scale terrain dataset. The major contributions can be listed as follows.

(1) We present a method of non-redundant storage that reduced the cost of the external memory.

(2) To improve the rendering speed, new methods of error metric and dynamic construction of continuous LOD model are achieved.

(3) A pre-loading scheme based on view frustum extension is proposed, which implements the real-time loading of datasets.

The rest of the paper is organized as follows. Section 2 discusses the construction of out-ofcore multi-resolution model. Section 3 gives the details of real-time rendering of block-based multi-resolution terrain. Section 4 proposes scheduling and pre-loading scheme based on view frustum extension. The experiments and conclusion of this paper are given in Section 5 and Section 6 respectively.

2 Out-of-Core Multi-resolution Model

2.1 Terrain Segmentation

Obviously, it is impossible to pre-load all the datasets of large-scale terrain into the main memory for visualization. Thus, we divide the original terrain data with the resolution of $(2^n+1)\times(2^m+1)$ vertices into a set of uniform blocks in preprocessing stage. As shown in Fig.1, each block has $(2^k+1)\times(2^k+1)$ vertices, and the whole terrain is divided into $2^{n-k}\times 2^{m-k}$ blocks.

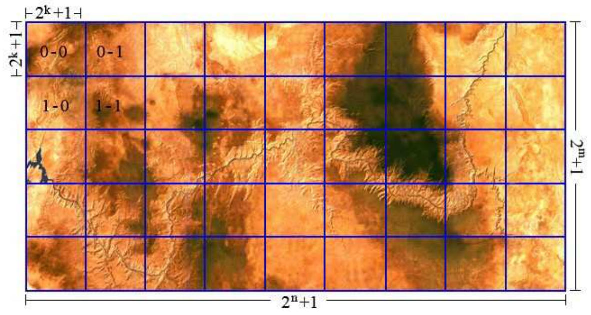


Fig. 1 Terrain Segmentation

We denote each of the block at row *i*, column *j* by Row_iCol_j. To facilitate the storage and the index of data, based on the spatial location of each block in the original terrain, a grid index file is built by which all the blocks are effectively organized in the external memory.

2.2 Multi-resolution Construction

For each block, a LOD hierarchy, which contains different resolution of elevation data, is constructed based on quadtree-split. Fig.2 shows the subdivision of quadtree hierarchy.

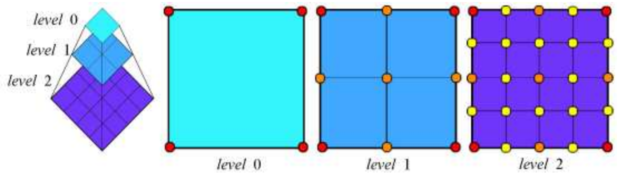


Fig. 2 Quadtree subdivision of terrain blocks

The main idea is to start with a coarse terrain dataset *level-0*, and it recursively constructs finer level by splitting node into four child nodes from top to bottom, until level $i > L$, where L is the maximum of level. Each node is composed of vertices.

In order to store the hierarchical multi-resolution data, pyramid model [14-17] is the most used approach. Each level of the pyramid corresponds to a resolution of height map, which leads to numerous redundant data. We assume that the current level is i, $i \in [1, L]$, then the number of redundant vertices between adjacent levels are $(2^{i-1}+1)\times(2^{i-1}+1)$. Obviously, with the increase of the depth of quadtree-split, the redundant data will dramatically increase.

To deal with this problem, we adopt a different storage strategy. The main idea of our method is to store the newly added vertices in each level. As shown in Fig.3, the lowest resolution data is taken as *level-0*, which has four vertices (0~4). Quadtree subdivision constructs *level-1* based on *level-0*, five vertices (4~8) are generated. In traditional pyramid model, all the vertices in *level-1* must be stored.

Therefore, more and more redundant vertices are reproduced with the increase of levels. In contrast to pyramid model, we store only the newly added vertices in each level. Then a linear sequence is constructed, e.g. in *level-1* we just store five vertices (4~8). Thus, there are no redundant data in block and the cost of the external memory is reduced.

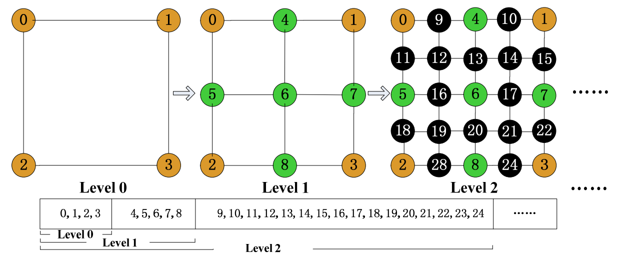


Fig. 3 Block non-redundant data storage method

We apply Equation (1) and Equation (2) to calculate the start offset and the end offset of elevation value in terrain files:

$$Offset(n)_{start} = \begin{cases} 0, & n = 0 \\ (2^{n-1}+1)\times(2^{n-1}+1), & n \ge 1 \end{cases} \quad (1)$$

$$Offset(n)_{end} = (2^n+1)\times(2^n+1)-1, n \ge 0 \quad (2)$$

To access hierarchical data efficiently, *level index file* and *order index file* are built, which are list structure and reside in the main memory. Level index file records the hierarchy number that vertices belong to, and order index file records the location offset of each vertex in a level.

In preprocessing stage, elevation values of vertexes are constructed into a linear sequence by adopting non-redundant storage method. Then the level and the order number of each vertex are calculated, and the values are written into the corresponding index files. In real-time rendering stage, following steps are performed:

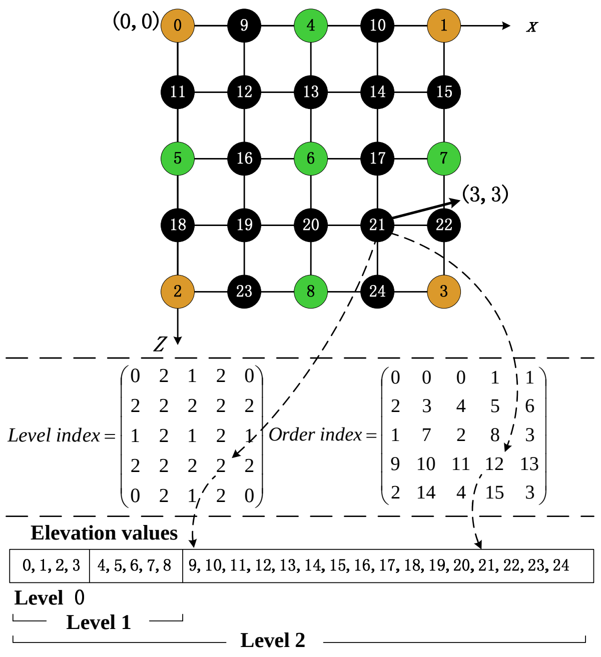
Step 1. Get the coordinate value (*X*, *Z*) of current vertex in block.

Step 2. Get level number of current vertex in block by querying the level index file.

Step 3. Get order number of current vertex in the corresponding level by querying the order index file.

Step 4. Get elevation value by calculating the offset.

Let us take a 5×5 terrain block as an example. As shown in Fig.4, for the vertex numbered 21, its coordinate is (3, 3). Firstly, by querying the index files, we know the corresponding level number of vertex 21 is 2, and the order number is 12. Secondly, using Equation (1) to calculate its start offset ($offset(2)_{start}$ = 9). Thus, the location of the elevation value in linear sequence is 21($offset(2)_{start}$+12).



$$Level\ index = \begin{pmatrix} 0 & 2 & 1 & 2 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 0 & 2 & 1 & 2 & 0 \end{pmatrix} \quad Order\ index = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 2 & 3 & 4 & 5 & 6 \\ 1 & 7 & 2 & 8 & 3 \\ 9 & 10 & 11 & 12 & 13 \\ 2 & 14 & 4 & 15 & 3 \end{pmatrix}$$

Fig. 4 Mapping relationship among vertex coordinate, index file and elevation value

2.3 Static Error

Static error is the difference of elevation values, i.e., the vertical distance among corresponding vertices in the original and the approximate mesh. It is independent of the viewpoint. To obtain error metric as fast as possible in rendering stage, in this paper, we calculate static error in pre-processing.

The static error of parent nodes is not always larger than their children's, which leads to an imprecise approximation of terrain surface. To tackle with this problem, an error function $StaticE(\delta_i^l)$, denoted as Equation (3), is applied to ensure that the error of parent node is larger than its child nodes.

$$StaticE(\delta_i^l) = \begin{cases} H(\delta_i^l) & l = l_{\max} \\ \max\{H(\delta_i^l), \max\limits_{\substack{l' \in [l+1,\ldots l_{\max}] \\ i' \in [1,4]}} \{StaticE(\delta_{i'}^{l'})\}\} & l \ne l_{\max} \end{cases} \quad (3)$$

where $H(\delta_i^l)$, defined in Equation (4), is the static error of node *i* in level *l*. It is the maximal value of six vertices error, including four midpoints of node's edges and two midpoints of diagonal lines.

$$H(\delta_i^l) = \max_{n=1\ldots6}(\delta_n) \quad (4)$$

$$\underset{n=1..6}{\delta_n} = |f(v_n) - f'(v_n)| \quad (5)$$

$f(v_n)$ is the original elevation value of vertices v_n, $f'(v_n)$ is the elevation value calculated by linear operation.

3 Real-Time Rendering of Block-based Multi-resolution Terrain

3.1 View Frustum Culling

In fact, due to the limitation of view field of frustum, most of the elevation data are not involved in rendering of the final terrain image. Therefore, view frustum culling is an efficient way to increase the rendering speed. Generally, view frustum culling needs to test six planes of the view frustum. To reduce the computational complexity, we adopt a fast culling simplification algorithm based on projection of view frustum.

As shown in Fig.5, we discard three of the six planes by ignoring the near clipping plane, top clipping plane, bottom plane and simply projecting the view frustum to *x*-*z* plane.

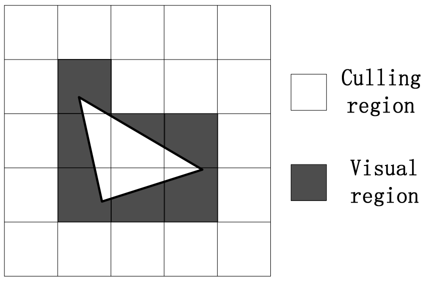


Fig. 5 Simplification of frustum culling

The culling is implemented by intersection test based on blocks of the terrain. The blocks, which intersect with projection triangle, construct the visual region. The rendering of the rest blocks is unnecessary.

3.2 Error Metric

Error metric is measurement of how well the coarse mesh approximates the full resolution mesh, and it is a criterion for the selection of which level to be rendered. Classical error metric has only considered with the static error that related to the roughness of surface and the distance from viewpoint to the center of current terrain node, which may lead to unnecessary splitting. We put forward a sight-line-dependent method to calculate screen error. To improve clarity in the following, we first introduce some parameters.

As shown in Fig.6, α is the vertical field angle of the viewpoint. d is the distance from viewpoint to screen. H stands for the width of screen. θ represents the angle between the sight line and terrain. D is the distance from viewpoint to the node. δ and ρ represent static error and screen error respectively. δ has been calculated in preprocessing stage and ρ is a dynamic value which is calculated in the rendering stage.

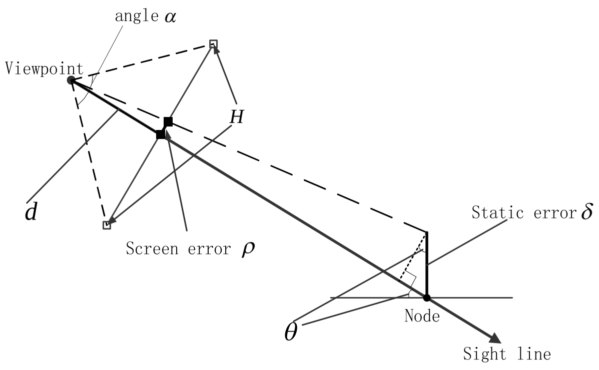


Fig. 6 Error metric

Firstly, considering angle θ between sight line and terrain, the larger the angle θ, the smaller the screen error of the node. Therefore, the equation of screen error is as follows:

$$\rho = \frac{H\delta\cos\theta}{2D\tan\frac{\alpha}{2}} \tag{6}$$

Secondly, the relationship between sight line and node need to be considered. As shown in Fig. 7, angle θ_2 is smaller than angle θ_1, that is to say, node 2 is closer to the direction of sight line than node 1, and then it should have higher precision.

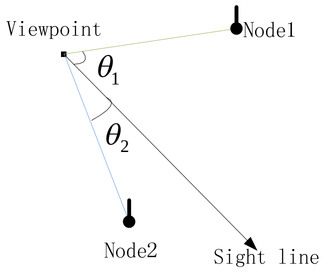


Fig. 7 Relationship between sight line and node

Thus, suppose that the angle between direction of sight line and current node is β, the final equation for calculating the screen error can be expressed as:

$$\rho = \frac{H\delta\cos\theta}{2D\tan\left(\frac{\alpha}{2}\right)(1+\sin\beta)} \tag{7}$$

where τ is the threshold value, and the node needs to be split when $\rho \le \tau$.

Our error metric method not only considers with the roughness of terrain surface and viewpoint distance, but also introduces sight line as an important factor, so that the result is more reasonable.

3.3 Dynamic Construction of Continuous LOD Model

For the blocks in visual region, the continuous LOD model needs to be constructed. In general, we could adopt Equation (7) to calculate screen error ρ . Whether blocks will be split depends on the values of ρ and threshold τ . However, it is unnecessary to construct continuous LOD model for the whole region, because only the data in projection triangle area need to be rendered. To improve efficiency and reduce redundant triangles, we adopt bounding circle to determine the visibility of quadtree nodes and combine with error metric method to implement the dynamic construction of continuous LOD model. As shown in Fig.8, we present the algorithm as follows:

Step 1. Split the current node that intersects with view frustum by quadtree-split.

Step 2. Set the diagonal of child node as diameter, and construct bounding circle of each child node.

Step 3. Test intersection between bounding circle of each node and the projection triangle of view frustum. Ignore the non-intersected nodes, and the intersected nodes go to step 4.

Step 4. Use Equation (7) to calculate the corresponding screen error ρ . The nodes that satisfy $\rho > \tau$ are split recursively. For each child node, repeat above step 2 to step 4, until current node is leaf node or $\rho < \tau$.

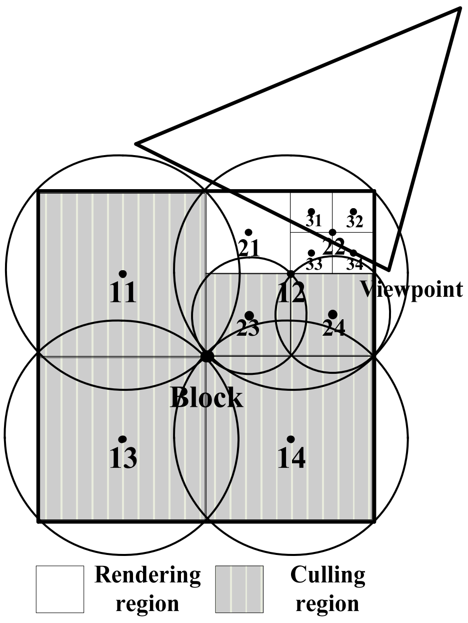


Fig. 8 Construction of continuous level of detail

4 Scheduling and Pre-loading Scheme based on View Frustum Extension

For the massive terrain data, it is difficult to load all the datasets in main memory. Thus, developing effective scheme to decrease the time-consuming data loading from the external memory becomes the principal task for interactive large-scale terrain visualization.

In terrain visualization systems, the viewpoint is always navigating in the scene. If we can predict the data that required in the following times and load them in advance from the external memory, the time for data exchange between the main memory and the slower external memory in rendering stage will be eliminated effectively. Based on this idea, we propose a scheduling and pre-loading scheme for terrain data, as shown in Fig.9.

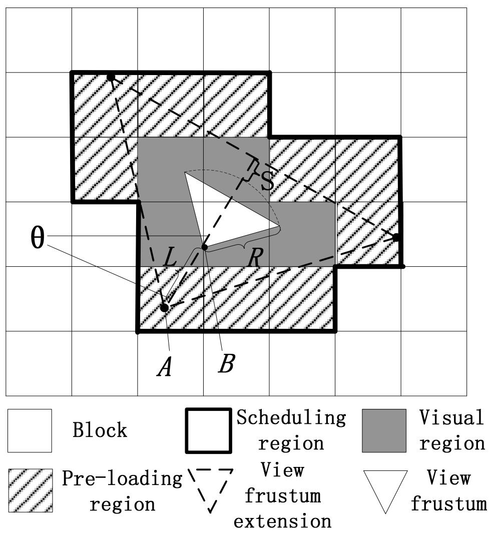


Fig. 9 View frustum extension-based pre-loading scheme

The terrain coverage is divided into view frustum region, view frustum extension region, visual region, scheduling region and pre-loading region. View frustum region is defined by view frustum, and it depicts the area user can see. Visual region consists of terrain blocks that intersected with view frustum. *M* is a set of blocks in visual region, which is used to construct continuous LOD model. View frustum extension and view frustum are two similar triangles. The terrain data that must be loaded is dependent on the size of view frustum extension. Scheduling region consists of terrain blocks, which intersected with view frustum extension. *N* is a set of blocks in scheduling region, which need to be loaded in the main memory. The set $N - M$ depicts terrain blocks in pre-loading region, which is the predicted data of next frame.

As shown in Fig.9. *R* is the length of two legs of isosceles triangle of view frustum. The distance from vertex A to vertex B is L. With the increase of L, the requirement of viewpoint for high-speed rotation motion is more able to meet, but the cost of scheduling is higher at the same time. Generally, we define L has the same value as the length of a block boundary. Distance from vertex A to the hypotenuse is $L+R+S$. S is variable, which is determined by motion speed and the direction of viewpoint. We can use S to adjust the range of view frustum extension dynamically.

As shown in Fig.10, for the translational motion of viewpoint, when the angle between motion direction and the direction of sight line in $[0, \pi/2)$, S is proportional to the velocity component in sight line direction. For the other translation and rotation, S is 0. Assume that the current motion speed is V_i, the angle between motion direction and direction of sight line is α, and the frame rate is f. S_i can be calculated as follows:

$$S_i = \begin{cases} V_i \cos\alpha / f, & \alpha \in [0, \frac{\pi}{2}) \\ 0, & others \end{cases} \qquad (8)$$

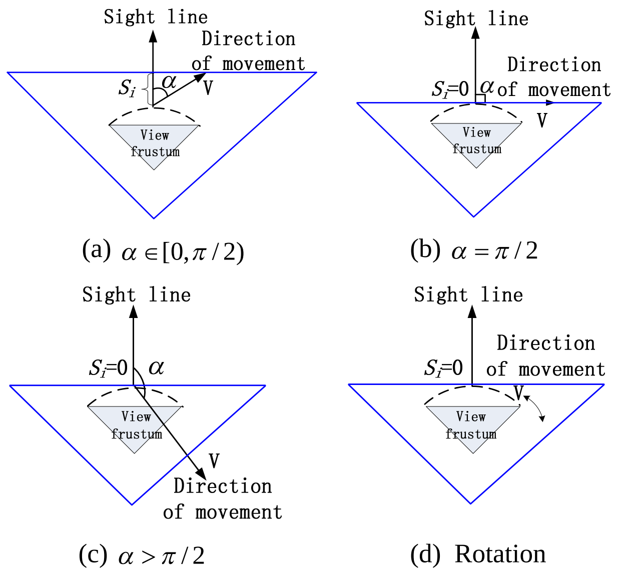


Fig. 10 The calculation method for S

We apply multi-threading mechanism to speed up scheduling and rendering process.

The main functions of main thread include calculation of view frustum and view frustum extension, choosing terrain blocks for data loading and unloading, updating linked list and rendering terrain data. Main thread creates a sub-thread for each terrain block that need to be fetched or unloaded. The main functions of sub-thread are calculation the levels of detail of current block, loading or unloading terrain data level-by-level, sending signal if a block need to be rendered.

The procedure of main thread is as follows:

Step 1. Get the position of current viewpoint, calculate view frustum and view frustum extension by the speed and direction of motion.

Step 2. Perform intersection tests between projection view frustum and blocks. Construct linked list for the blocks that intersected with view frustum and set the value of n, which records the number of blocks that need to be rendered.

Step 3. Create a sub-thread for each block in linked list respectively to implement data scheduling.

Step 4. Check the value of n. If n=0, shutdown all the sub-thread, and go to step 5. Otherwise, determine whether there are signals from terrain blocks. If no signal has arrived, then go on to wait. Otherwise, scan linked list, render the terrain block that sent signal, set n=n-1, and return to step 4.

Step 5. Finish.

The procedure of sub-thread is as follows:

```
Step 1. Test whether current level of detail is the suitable one for the current terrain block. If it is true, go to step 1.1. Otherwise, go to step 1.2.
    Step 1.1 Test whether current block need to be rendered. If it is true, go to step 1.1.1. Otherwise, go to step 1.1.2.
        Step 1.1.1 Send signal, go to step 2.
        Step 1.1.2 Go to step 2.
    Step 1.2 Load or unload the data by one level, and return to step 1.
Step 2. Finish
```

5 Experiments and Discussions

We use OpenGL on C++ platform to implement the method presented in this paper, and carry out related experiments on PC. The computer configuration used for experiments is Windows XP operating system, 2G memory, and NIVDIA 9800GT graphics card.

Grand Canyon (Arizona, USA), Puget Sound (Washington State, USA) and Southeast of China are tested as the data sources. The size of Grand Canyon dataset is about 16M with the resolution of 4097×2049, which covers a rectangle area with width 112km and length 224km. The original size of Puget Sound dataset is about 512M, with the resolution of 16385×16385 grid of 16bit heights covering a square region of length about 160km. The original size of Southeast of China dataset is about 2.01G, with the resolution of 36006×30006.

In preprocessing stage, we construct block-based multi-resolution structure. The size of each block is 1025×1025. Mipmapping technique is applied to construct the corresponding textures of the terrain.

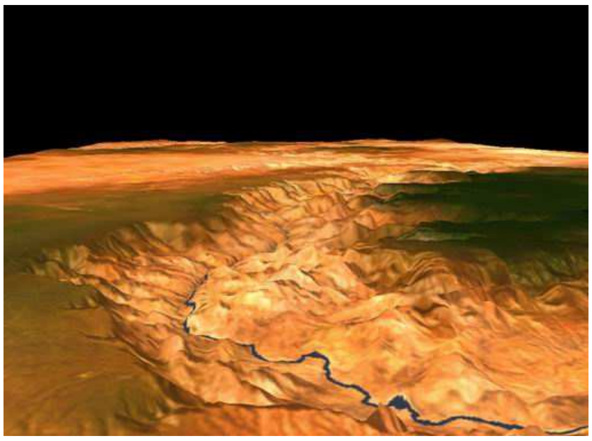
The textures are compressed to DXT1 format. Table 1 shows the size of data before and after treatment.

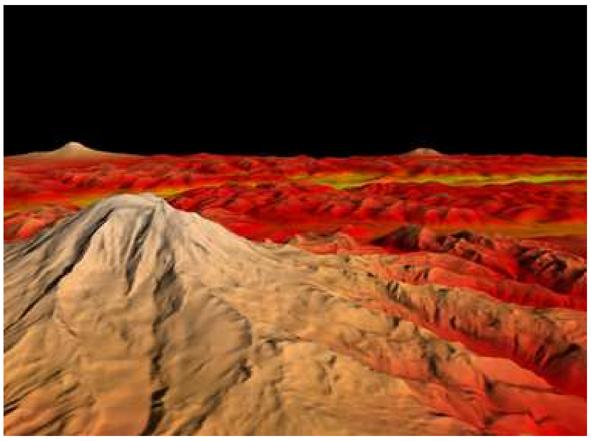
Table 1. Experimental Data

Dataset	DEM data		Texture data	
	Original size	After Processing	Original size	After Processing
Grand Canyon	16.04M	17.3M	24M	5.33M
Puget Sound	512M	555M	768M	170M
South-east of China	2.01G	2.18G	3.01G	685M

We have compared our external memory cost with some pervious works. The Puget Sound dataset, for example, in [12] and [16-17], the cost are 5G and 1.5G. Our method has reduced the external memory cost by 90% and 60%.

When thresholds τ is 2, the visualization results of large-scale terrain by our method are shown in Fig.11, which indicate that our method can render large-scale terrain of the fine details and it has little perceivable loss in image quality.


(a) Grand Canyon


(b) Puget Sound

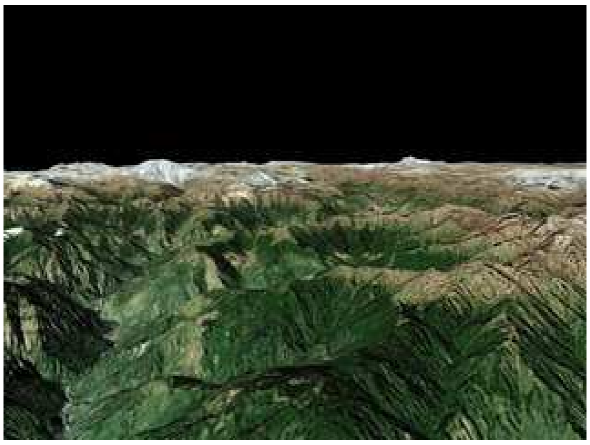
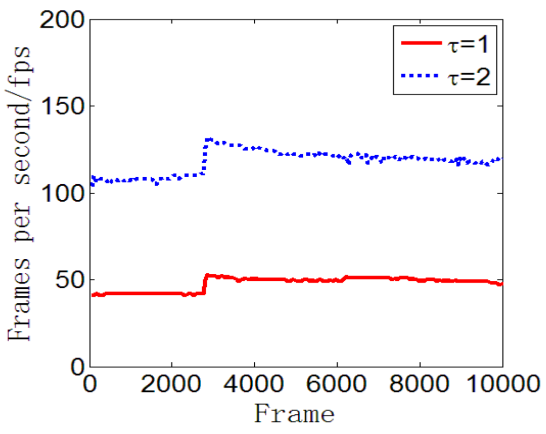
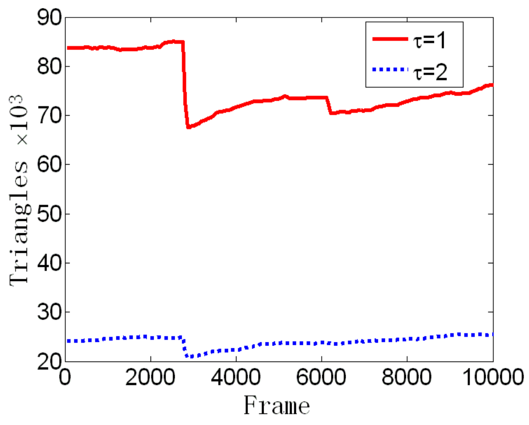

(c) Southeast of China

Fig. 11 Visualization results of large-scale terrain

To testify the rendering efficiency of our method, we rendered three terrain datasets to a 1024×768 view port respectively. The fly-through path is predefined, and the rendering mode is terrain geometry with texture. We recorded the frame rate and triangles per frame with different thresholds of screen error. The results are shown in Fig.12-Fig.14.

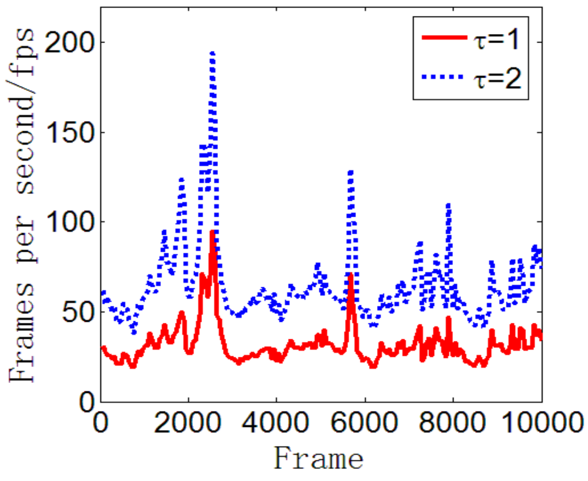


(a) Frame rate

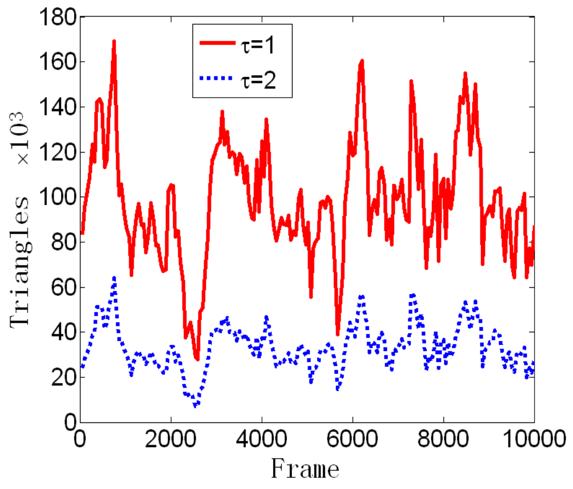


(b) Triangles per frame

Fig. 12 Experiment of Grand Canyon with 10000 frames

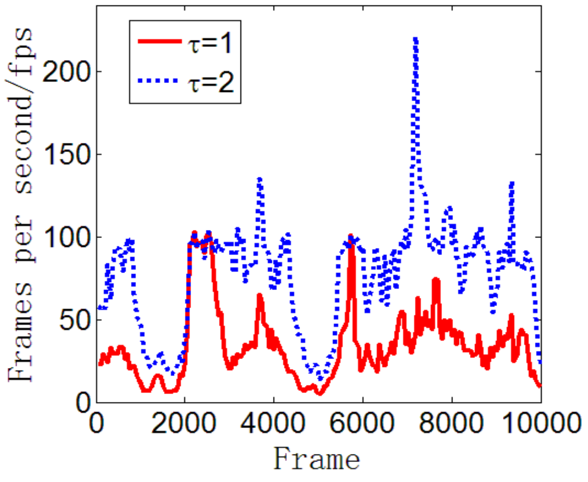


(a) Frame rate

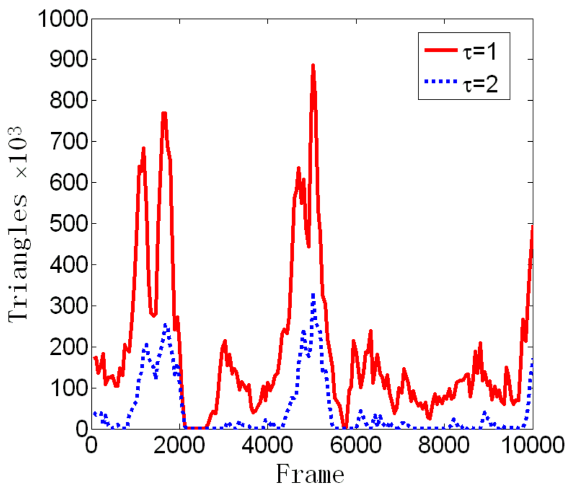


(b) Triangles per frame

Fig. 13 Experiment of Puget Sound with 10000 frames



(a) Frame rate



(b) Triangles per frame

Fig. 14 Experiment of Southeast of China with 10000 frames

At the same time, we made statistical analysis of the average frame rate, triangles per frame and main memory cost while navigation. Table 2 shows the results.

Table 2. Statistics on frame rates, rendering complexity and peak value of memory cost while navigating

Dataset	τ	Average Triangles per frame	Average Frames per second(fps)	Peak value of memory cost (M)
Grand Canyon	1	77,350	48	61
	2	24,390	110	54
Puget Sound	1	98,230	40	102
	2	33,430	67	85
Southeast of China	1	184,880	34	142
	2	42,310	65	92

Southeast of China is large enough to show the visualization efficiency of our method for massive datasets. Puget Sound and Grand Canyon represent medium and small datasets respectively. The outcome of experiment indicates our method can achieve a high frame rate, which can meet the basic requirement of the interactive visualization.

We also carried out an experiment to testify the efficiency of scheduling scheme. In this experiment, threshold τ is set to 2. The results are as shown in Table 3. Experimental results show that the I/O time is relatively long in the first frame. This is because all the needed data are loaded from external memory. However, benefited from pre-loading scheme, in following frames, the total quantity of real-time loading data is reduced significantly and the scheduling efficiency is increased.

Table 3. Results of loading data and scheduling time

Frame series number	Grand Canyon			Puget Sound			Southeast of China		
	Needed data (M)	Real-time loading data (M)	I/O time (ms)	Needed data (M)	Real-time loading data (M)	I/O time (ms)	Needed data (M)	Real-time loading data (M)	I/O time (ms)
1	1.193	1.193	20.2	1.307	1.307	21.3	1.425	1.425	30.9
162	1.578	0.376	3.88	1.975	0.032	0.59	0.390	0.012	0.24
388	1.297	0.094	1.18	2.445	0.034	0.97	1.975	0.032	0.90
4509	1.372	0.372	3.38	1.284	0.007	0.19	1.472	0.085	0.07
5330	1.446	0.024	0.67	1.719	0.007	0.16	1.769	0.125	1.10
7374	1.508	0.094	0.70	1.113	0.008	0.23	1.267	0.126	1.15
8208	1.384	0.366	3.31	2.267	0.024	0.30	0.272	0.024	0.46
10000	1.203	0.019	0.41	1.132	0.014	0.29	1.132	0.113	1.09

6 Conclusion

In this paper, we present an effective out-of-core large-scale terrain visualization method. To realize the multi-resolution storage, a non-redundant storage method is adopted, which can effectively reduce the cost of external memory. Moreover, in order to increase the rendering speed, new error metric and dynamic constructions of continuous LOD models methods are introduced. At the same time, to ensure the continuity of navigation, a scheduling and pre-loading scheme based on view frustum extension is presented. We analyzed the performance of our method. The results showed that our method could provide solution for interactive rendering terrain scene with large dataset.

There also exists some limitations in our method. We apply Mipmapping technique to deal with the textures of terrain, which may increase extra memory cost, although compression approach is used. Furthermore, in our method, we assume the terrain is non-spherical structure; therefore, our work currently can deal with only planar terrain.

In the near future, we expect to improve our method to visualize spherical terrains [19, 20]. Moreover, the challenging issue of large-scale dynamic terrain [21, 22] is also one of our future research topics.

Acknowledgement:
The authors wish to thank the anonymous reviewers for their thorough review and highly appreciate the comments and suggestions, which significantly contributed to improving the quality of this paper. Also, we would like to thank Dr. Yugang Liu who spent time on reading the earlier drafts of this paper.

This work is supported by National High Technology Research and Development Program of China (No. 2012AA011503), Project on the Integration of Industry, Education and Research of Guangdong Province (No. 2012B090600008) and pre-research project (No. 51306050102).

References:
[1] C. J. Yang, F. Q. Zhang, S. Wu, 3D Geographic information system on Wenchuan earthquake, *Journal of Remote Sensing*, Vol.12, No.6, 2008, pp. 839-899. (in Chinese)
[2] H. Q. He, Y. Y. Xing, T. B. Wang, Real-time rendering system of large-scale terrain in flight simulation: design and implementation, *In Proc. CGIV'09*, Tianjin, China, 2009, pp. 180-185.
[3] X. Y. Li, M. Li, W. Cai, The research of 3D terrain generation and interactivity realization techniques in virtual battlefield environment, *In Proc. ICIG'09*, Xi'an, China, 2009, pp. 668-671.
[4] H. Qiu, L. T. Chen, 3D Visualization of radar coverage considering terrain effect, *Journal of Electronic Measurement and Instrument*, Vol.24, No. 6, 2010, pp. 528-535.(in Chinese)
[5] J. WU, Y. F. Yang, S. R. Gong, et al, A new quadtree-based terrain LOD algorithm, Journal of Software, Vol.5, No.7, 2010, pp.769-776.
[6] P. Lindstrom, D. Koller, W. Ribarsky, et al, Real-time, continuous level of detail rendering of height fields, *In Proc. SIGGRAPH'96*, New Orleans, USA, 1996, pp. 109-118.
[7] M. Duchaineau, M. Wolinsky, D. Sigeti, ROAMing terrain: real-time optimally adapting meshes, *In Proc. Visualization'97*, Phoenix, USA, 1997, pp. 81-88.
[8] H. J. Zhang, Y. H. Lv, S. H. Liu, A terrain model simplification method based on adaptive areas division, *Journal of Computer Research*

and Development, Vol.47, No.1, 2010, pp. 53-61.(in Chinese)

[9] F. Losasso, H. Hoppe, Geometry clipmaps: terrain rendering using nested regular grids, *ACM Transactions on Graphics*, Vol.23, No.3, 2004, pp. 769-776.

[10] Y. Livny, Z. Kogan, J. Elsana, Seamless patches for GPU-based terrain rendering, *The Visual Computer*, Vol.25, No.3, 2009, pp. 197-208.

[11] R. Pajarola, E. Gobbetti, Survey on semi-regular multiresolution models for interactive terrain rendering, *The Visual Computer*, Vol.23, No.8, 2007, pp. 583-605.

[12] P. Lindstrom, V. Pascucci, Visualization of large terrains made easy, *In Proc. Visualization'01*, San Diego, USA, 2001, pp. 363-370.

[13] P. Lindstrom, V. Pascucci, Terrain simplification simplified: a general framework for view-dependent out-of-core visualization, *IEEE Transactions on Visualization and Computer Graphics*, Vol.8, No.3, 2002, pp. 239-354.

[14] B. S. Deng, R. H. Yu, F. Y. Qin, et al, Research on seamless rendering of large scale terrain, *Application Research of Computers*, Vol.29, No.1, 2012, pp. 369-372.(in Chinese)

[15] S. Zhao. Y.P. Lu, Multi-resolution quadtree based algorithm for real-time visualization of massive terrain dataset, *In Proc. ICMT'2011*, Hangzhou, China, 2011, pp. 5934-5938.

[16] S. X. Shi, X. Z. Ye, S. Y. Zhang, Partition based on real-time rendering method for large-area terrain data, *Journal of Zhejiang University (Engineering Science)*, Vol.14, No.12, 2007, pp.2002-2006.

[17] Y. Q. Lu, Study of the real-time rendering for large-scale terrain dataset, *PhD Thesis*, Zhejiang Univsersity, 2003.(in Chinese)

[18] X. H. Long, Y. P. Jin, Y. L. Song, Research on realistic rendering technology for large-scale terrain, *Computer Engineering*, Vol.38, No.7, 2012, pp. 260-262.(in Chinese)

[19] M. Clasen, H. C. Hege, Terrain rendering using spherical clipmaps, *Eurographics/ IEEE-VGTC Symposium on Visualization'06*, Lisbon, Portugal, 2006, pp. 91-98.

[20] R. Westerteiger, A. Gerndt, B. Hamann, Spherical terrain rendering using the hierarchical HEALPix grid, *In Proc. IRTG'11*, Kaiserslautern, Germany, 2011, pp. 13-23.

[21] V. R. Kamat, J. C. Martinez, Large-scale dynamic terrain in three-dimensional construction process visualizations, *Journal of Computing in Civil Engineering*, Vol.19, No.2, 2005, pp. 160-171.

[22] D. Wang, Y. N. Zhang, P. Tian, et al, Real-time GPU-based visualization of tile tracks in dynamic terrain, In Proc. International Conference on Computational Intelligence and Software Engineering, Wuhan, China, 2009, pp. 1-4.