Send Orders for Reprints to reprints@benthamscience.ae

Large Scale Terrain Real-Time Rendering on GPU Using Double Layers Tile Quad Tree and Cuboids Bounding Error Metric

Ying Yang*, Chunfang Wang, Yuyu Gao and Jingwei Xing

LiRen College of Yanshan University, Qinhuangdao, Hebei, 066004, China

Abstract: Improving terrain tile data selection efficiency, real-time loading of visible tile data and building GPU-based continuous Level of Details (LOD) are the key technologies for large scale terrain rendering on GPU. In this article, in order to reduce terrain tile data selection time, we build double layers tile quad tree for massive terrain data and organize tile data by designing Z-order space filling curve. According to the visible region coordinates obtained by GPU offscreen render to texture, we realize real-time loading of visible tile data from CPU to GPU. Map visible tile quad tree into two-dimensional texture on GPU making full use of the characteristics of GPU multi-channel parallel processing. In order to execute tile error metric computation and LOD selection on GPU, we design GPU-based cuboid bounding unsaturated error metric, reduce CPU computational burden and enhance the terrain rendering performance. Experiments show that our algorithm can improve the utilization rate of GPU in the terrain rendering and achieve a good visual effect and a high frame rate.

Keywords: Terrain, double layers, tile quad tree, cuboid, error metric, GPU.

1. INTRODUCTION

The real-time rendering of large scale terrain is an important topic in the virtual reality and geographic information system, and also is an important underlying supporting technology in emergency city planning, disaster prevention and control and virtual battlefield simulation. The scale of terrain data seriously affects the real-time and authenticity of terrain visualization. With the improvement of GPU processing capacity, reducing CPU computational burden and the frequency of CPU-GPU interactions, large scale terrain rendering on GPU is a hot area of computer graphics research issues. The main work of improving terrain rendering performance contains the following two aspects：1. Massive terrain data organization based on out of core, reduces terrain tile data selection time and the load bandwidth. 2. Terrain mesh simplification on GPU, designing reasonable error metrics make full use of the characteristics of GPU multi-channel parallel processing in order to implement terrain view dependent multi-resolution representation.

With GPU become faster and more powerful, fine-grained LOD, which processes a triangle as a simplified unit, has been replaced by coarse-grained LOD. Coarse-grained LOD, which processes a set of triangles as simplified unit, can greatly improve the utilization rate of GPU and reduce CPU computational burden. Since GPU developed and accelerated, coarse-grained LOD methods achieve high performance, and are more suitable for large scale terrain rendering on GPU. Block triangle binary tree [1], tile quad tree [2] and GeoClipmap [3] are the three major methods based on GPU coarse-grained LOD. Tile quad tree has became the mainstream method based on GPU coarse-grained LOD with the characteristics of grid LOD and texture LOD corresponding each other, data index simply and selection quickly. Ulrich used Tile Quad tree based on chunked LOD to manage terrain data and realize terrain real-time rendering [4]. Li Sheng constructed the incremental horizon dynamically to pre-extracted potential silhouette of each chunk for real-time walkthrough of the large scale terrain environments [5]. Dick presented a geometry compression scheme for restricted quad tree meshes [6], built the triangle strip by using only triangle types, windings and height values. Li Baiyun presented a GPU based implementation of quad tree for terrain rendering and used texture to store quad tree data, and pixel shaded to construct quad tree in real-time [7]. Vanek divided terrain into tiles of different resolutions according to the terrain's complexity, and it stored each tile as a mip-map texture [8]. Liu Hao presented an approach for dynamic scheduling and terrain real-time rendering [9]. The algorithms above most implemented based on single layer quad tree. As terrain's size increase, the quad tree become deeper, and tile selection time will grow as factor of 4n. It will break the limit of real-time. Nie Junlan presented Multilevel Tile Load Map (MTLM) algorithm accelerated frame rate. But the maximal size processed by single texture (4096×4096) limit the MTLM algorithm scalability and tile data indexes should be improved [10]. Michael Bader organized tile data by designing space filling curve, as a result, improved the speed of terrain rendering [11]. HyeongYeop Kang realized

1874-4443/14 2014 Bentham Open

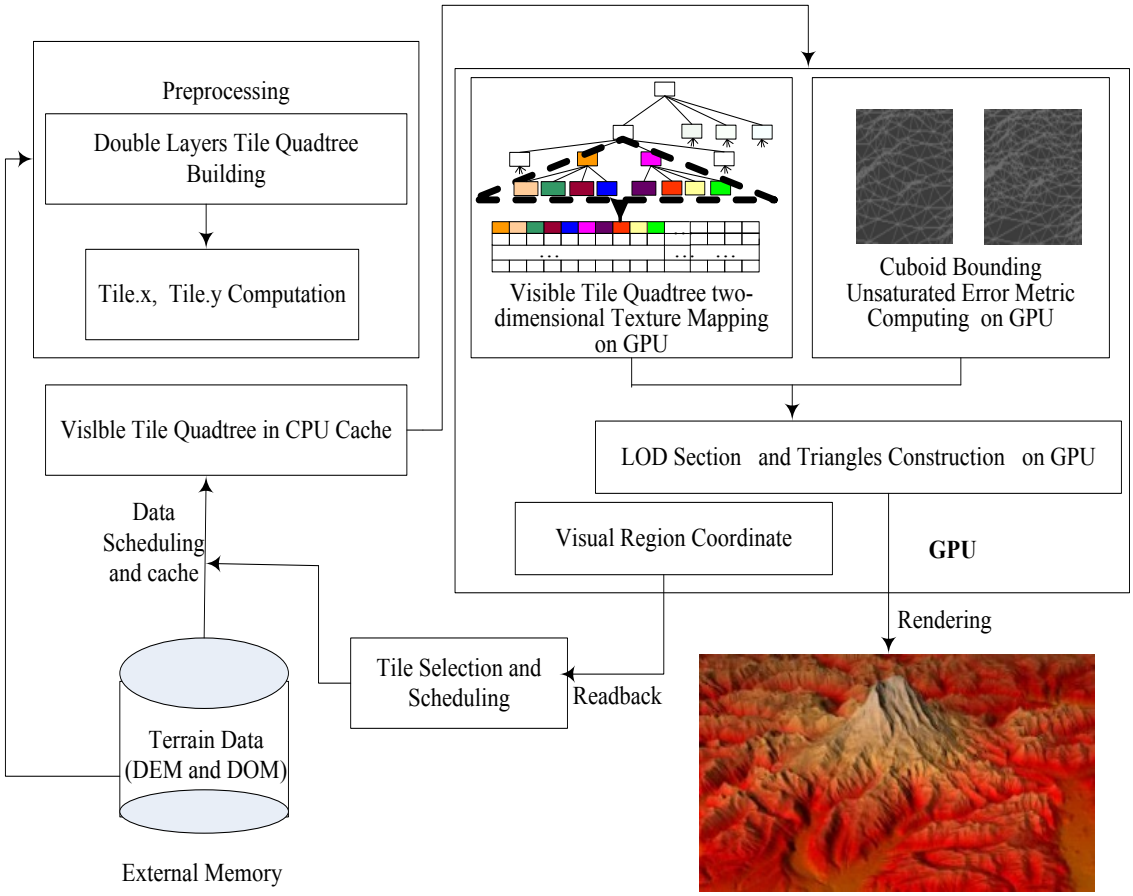


Fig. (1). Terrain rendering algorithm based on double layers tile quad tree and cuboids bounding unsaturated error metric.

GPU tessellation based on tile quad tree using Direct3D 11 [2].

In this article, the terrain regular grid data are split into tiles. Building double layers tile quad tree for massive terrain data and organizing tile data by designing Z-order space filling curve, optimize the tree modeling and improve the terrain rendering speed.

Reasonable error metric is one of the important criterions for terrain simplification. With GPU shades and powerful parallel computing capability development, it is possible to execute tile error metric computation and LOD selection on GPU. The research on GPU terrain simplified error metric is another hot topic of terrain rendering. Peter Lindstrom presented saturated error metric based on nested object space error and nested ball bounding to realize terrain rendering [12]. Lu Yanqing used unsaturated cuboid error metric based on triangle binary tree, decreases the amount of triangles in terrain rendering [13]. The algorithms above most implemented based on CPU. With the development of GPU powerful parallel computing capability, it is possible to execute tile error metric computation LOD selection on GPU. Li Sheng applied a new error metric constrained normal cone to view-independent simplification based on silhouette preserving and shading preserving criteria [5]. Lindstrom designed reasonable error metric to implement terrain data compression and terrain rendering [1]. Fu Wei presented a terrain rendering method with dynamic error metric and made error computation and triangulation executed by GPU. The method improved the rendering efficiency of GPU [14]. But the saturated error metric increased the amount of triangles, and the radius of nested ball will be very big when traveling several levels along the quad tree. This will be more serious when the difference of the adjoining terrain nodes' height value changes rapidly.

In this article, design cuboid unsaturated error metric, using GPU multi-channel parallel processing, in order to execute tile error metric computation and LOD selection by GPU acceleration.

2. GPU TERRAIN RENDERING ALGORITHM PRODUCEDURE

The GPU terrain rendering algorithm based on double layers tile quad tree and cuboid bounding unsaturated error metric is presented in this article. The detailed algorithm is shown in Fig. (**1**).

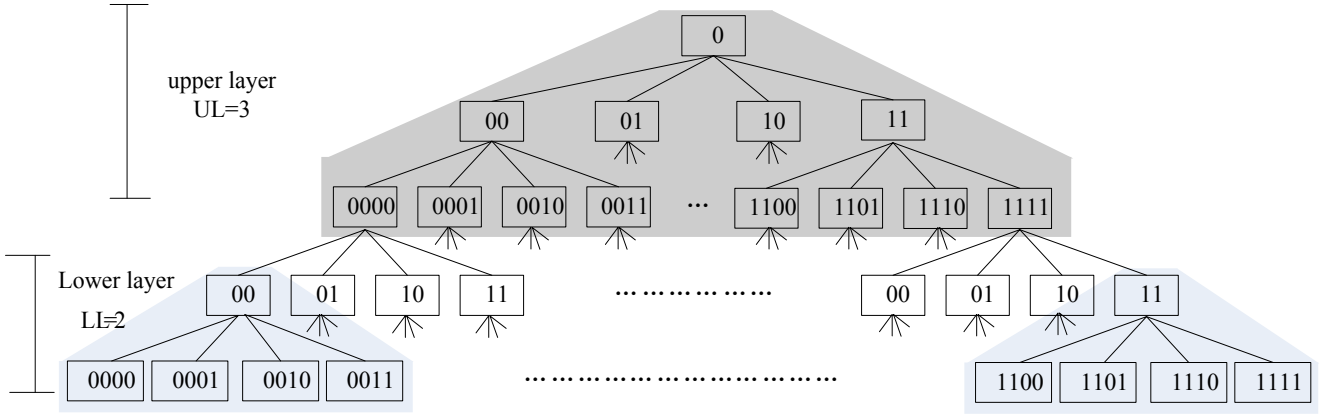


Fig. (2). Double layers tile quad tree.

All used terrain data are stored on the external memory in the form of tile pyramid. In the preprocessing, the terrain regular grid data are split into tiles. Double layers tile quad tree is built for massive terrain data and the tile data are organized by designing Z-order space filling curve. In the real-time rendering, visible region coordinates obtain by GPU Render to Texture and read back to CPU. CPU designs a tile selection mechanism, and uses a fixed size of CPU cache to store, manage and update the tile data. CPU selects the visible tile quad tree and real-time loads the visible tile data from CPU to GPU. Visible tile quad tree is mapped into two-dimensional texture computed on GPU. Error metric and cuboid unsaturated error metric computation and level-of-detail selection are executed by GPU acceleration. According to the error metric, tiles needed rendering are selected and triangulated. The real-time and authentic large scale terrain environment has been implemented. Two-dimensional texture mapping and error metric computation make full use of the characteristics of GPU multi-channel parallel processing, can save CPU run time and enhance the terrain rendering performance.

3. DOUBLE LAYERS TILE QUAD TREE BUILDING

Large scale terrain data are usually too large to load the whole terrain data sets at one time. As a means to combat this problem, it is necessary to design terrain data modeling and data organization. Tile quad tree has became the mainstream method based on GPU coarse-grained LOD with the characteristics of grid LOD and texture LOD corresponding each other, data index simply and selection quickly.

3.1. Tile Quad Tree

Tile quad tree is a static and precomputed level quad tree. The root node of tile quad tree represents a low resolution of the whole terrain. Quarter the root node into four children nodes, northwest (NW), northeast (NE), southwest (SW) and southeast (SE). Every one children node represents one quarter of the whole terrain. The resolution of children nodes is higher than the resolution of the root node. In the same way, quarter the root's children nodes into their children nodes. Repeat the steps until generate the left nodes. Every tile node has own code. The codes of the tile nodes are defined, in this order: NW is 00, NE is 01, SW is 10, SE is 11. The level of the tile nodes in tile quad tree can be obtained according to the codes of the tile nodes.

Tile quad tree has became the mainstream method in large scale terrain rendering and the data organization based on tile quad tree is more situable to GPU terrain rendering. At present, the terrain rendering algorithms most implemented based on single layer tile quad tree. As terrain's size increase, the quad tree will become deeper, invalid tiles selection time will grow quickly and the speed of terrain rendering will improve largely. Therefore, building double layers tile quad tree is urgently needed in order to optimize the tree modeling and improve the rendering speed.

3.2. Double Layers Tile Quad Tree Building

Double layers tile quad tree is made of two layers, upper layer tile quad tree and lower layer tile quad tree. An example of double layers tile quad tree of five levels is shown in Fig. (**2**). The level of upper layer tile quad tree is 3, the level of lower layer tile quad tree is 2.

The whole single layer tile quad tree is divided into double layers tile quad tree. It means that one single deeper layer tile quad tree is divided into many shallower tile quad trees. When traversal the double layers tile quad tree top-down, we judge whether the node is in the view frustum or not. If the node is not in the view frustum, the node and its children are ignored. If the node is in the view frustum, there were children need to judge, go on, top-down recursively until found all tiles which should be visible.

In this article, double layers tile quad tree reduces the depth of the tree, avoids selecting invalid tiles and calculation redundancy and decreases the computation time. As a result, the speed of terrain rendering will improved largely.

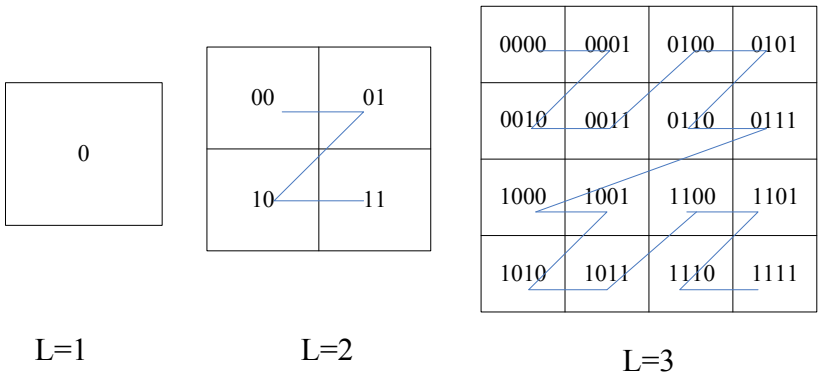


Fig. (3). Tile data organization using Z-order space filling curve.

3.3. Tile data Organization by Z-order Space Filling Curve

To improve data access efficiency, linear memory layouts should be care. We use Z-order Space filling curve at every level in double layers tile quad tree, and linke them end to end. Every tile data in double layers tile quad tree has own code. An example of codes and organization of 3 levels tile quad tree is shown in Fig. (**3**).

The coding regulation of tile data in double layers tile quad tree is defined as follow. The code of the root of the upper layer tile quad tree is 0. The codes of the four children of the root are defined, in this order: NW is 00, NE is 01, SW is 10, SE is 11.When the level is added 1, the codes of the children are added two bits binary codes after their fathers' codes. The added two bits binary codes are defined in the same order: NW is 00, NE is 01, SW is 10, SE is 11. Repeat the steps until generate the left nodes of the upper layer tile quad tree. The code of the root of the lower layer tile quad tree is 00. The codes of the four children of the root are defined, in this order: NW is 00, NE is 01, SW is 10, SE is 11.When the level is added 1, the codes of the children are added two bits binary codes after their fathers' codes. The added two bits binary codes are defined in the same order: NW is 00, NE is 01, SW is 10, SE is 11. Repeat the steps until generate the left nodes of the lower layer tile quad tree. As an example, the codes of double layers tile quad tree of five levels are shown in Fig. (**2**).

In this article, the scale of the whole terrain is WIDTH×WIDTH. The levels of the upper layer tile quad tree are numbered from 0 to UL. The levels of the lower layer tile quad tree are numbered from 1 to LL. When the level of the tile data is LEVEL, the bits number of the tile binary codes can reach 2×LEVEL and the tile binary encoding is defined as ZINDEX. Define $Level_l = l$, check codes M is 0x01. The Tile.x and Tile.y are computed in the following two cases.

In one case, if the tile data are in the upper layer tile quad tree, l<=UL, Tile.x and Tile.y are computed as follow

$$ZIndex_l = \mathrm{ZINDEX} >> 2(UL - Level_l) \,\&\, M \tag{1}$$

$$ZIndex_l' = \mathrm{ZINDEX} >> [2(UL - Level_l) + 1] \,\&\, M \tag{2}$$

l=1,2, …, UL in (1) and (2). $ZIndex_l$ and $ZIndex_l'$ in (1) and (2) are the intermediate variables used to compute Tile.x and Tile.y.

$$Width_l = WIDTH >> l \tag{3}$$

$$Tile.x = \sum_{l=1}^{UL} Width_l (ZIndex_l \,\&\, M) \tag{4}$$

$$Tile.y = \sum_{l=1}^{UL} Width_l (ZIndex_l' \,\&\, M) \tag{5}$$

$Width_l$ in (3) is the width value of the tile, whose level is l. $Width_l$ is an intermediate variable used to compute Tile.x and Tile.y.

In another case, if the tile data is in the lower layer tile quad tree, l>UL, $ZIndex_l$ and $ZIndex_l'$ obtained by (1) and (2), Tile.x and Tile.y are computed as follow

$$ZIndex1_l = \mathrm{ZINDEX1} >> 2(LL - Level_l) \,\&\, M \tag{6}$$

$$ZIndex1_l' = \mathrm{ZINDEX1} >> [2(LL - Level_l) + 1] \,\&\, M \tag{7}$$

l=1,2, …, LL in (6) and (7). $ZIndex1_l$ and $ZIndex1_l'$ in(6) and (7) are the intermediate variables used to compute Tile.x and Tile.y.

$$\begin{aligned} Tile.x = &\sum_{l=1}^{UL} Width_l (ZIndex_l \,\&\, MX) \\ &+ \sum_{l=1}^{LL} Width_{l+UL} (ZIndex1_l \,\&\, MX) \end{aligned} \tag{8}$$

$$\begin{aligned} Tile.y = &\sum_{l=1}^{UL} Width_l (ZIndex_l' \,\&\, MX) \\ &+ \sum_{l=1}^{LL} Width_{l+UL} (ZIndex1_l' \,\&\, MX) \end{aligned} \tag{9}$$

The computation of Tile.x and Tile.y has been completed in the preprocessing. The Tile.x and Tile.y are used to CPU selects the visible tiles, when visible region coordinates obtain by GPU. The Tile.x and Tile.y are also used to tile error

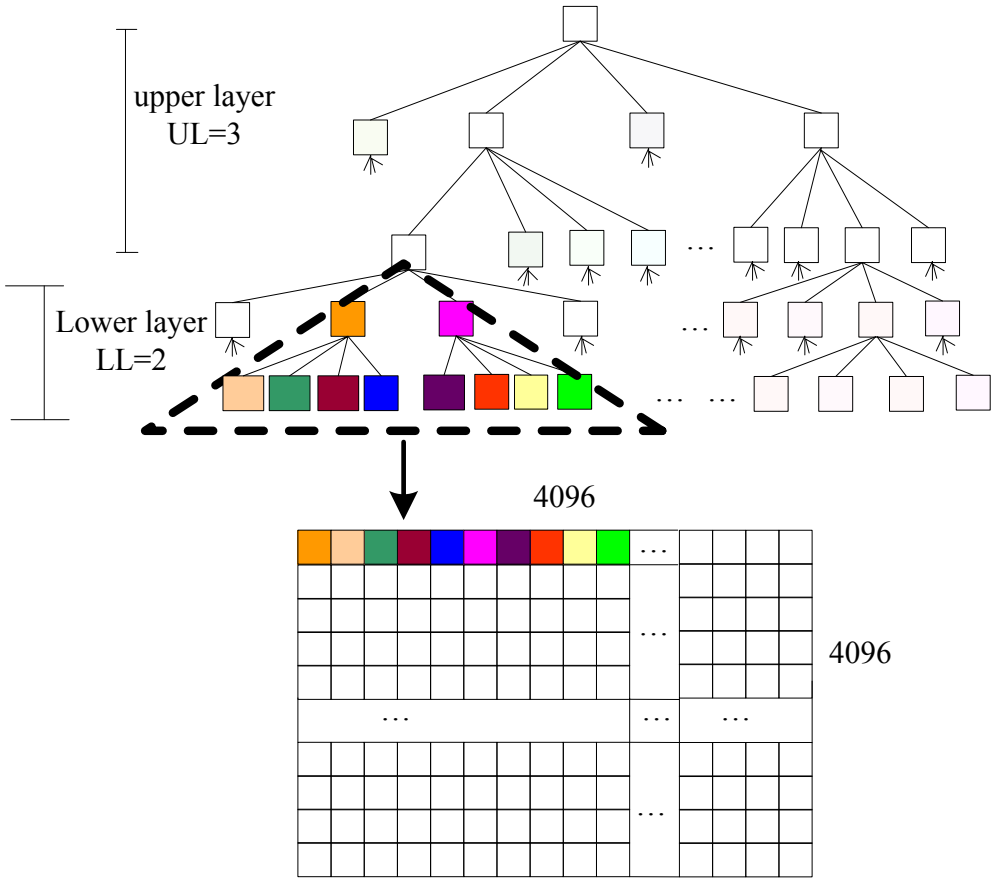


Fig. (4). Visual tile Quad tree two-dimensional texture mapping.

metric computation. While visible tile quad tree has been loaded from CPU to GPU, the Tile.x and Tile.y will be loaded at the same time. We organize the tile data by Z-order Space-filling curve, in order to enhance data access locality and improve tiles' selection efficiency. The computation of Tile.x and Tile.y contains only additive and bit operations. There aren't any complex calculations.

3.4. Visible Tile Quad Tree Two-Dimensional Texture Mapping on GPU

At present, the maximal size processed by single texture on GPU is 4096×4096. It is impossible to map the whole tile quad tree into two-dimensional texture computed on GPU. In this article, we make full use of the characteristics of GPU multi-channel parallel processing, only load visible tile quad tree from CPU to GPU, compute two-dimensional texture mapping and error metric on GPU. The two-dimensional texture mapping of visible tile quad tree is shown in Fig. (**4**).

The thick dotted region represents visible tile quad trees in Fig. (**4**). Map the tiles into the two-dimensional texture according to the order row first, column after. Every row contains 4096 texels. All tiles in visible tile quad tree can be seen as texels to compute on GPU, which can take advantage of GPU multi-channel powerful parallel computing capability and reduce CPU burden. In this article, the R channel respectively stores the level of the whole double layers tile quad tree, the G channels and B channels respectively store coordinate Tile.x and Tile.y. The level, Tile.x and Tile.y could be used to compute the tile error metric, detailed algorithm in Ⅴ.

4. GPU CUBOID BOUNDING UNSATURATED ERROR METRIC

With GPU computing capability become faster and more powerful, GPU-based mesh simplification criterion is another hot research in terrain rendering. Scholars have done a lot of work on GPU error metric. Lindstrom presented saturated error metric based on nested sphere bounding to realize terrain rendering [12]. Lu Yanqing used unsaturated cuboid error metric based on triangle binary tree on CPU, decreased the amount of triangles [13]. Fu Wei presented a terrain rendering method with dynamic error metric, made error and triangulation executed by GPU [14]. But the saturated error metric would increase the amount of triangles, and the radius of nested ball will be very big when traveling several levels along the quad tree. In view of more triangles caused by saturated error metric and the bigger radius caused by nested ball, we design GPU cuboid bounding unsaturated error metric, execute tile error metric computation and level-of-detail selection by GPU acceleration.

4.1. Object Space Error Metric and Cuboid Bounding Design

Error metrics include object space error metric and screen space error metric. Object space error metric is the vertical height difference between the vertex's original height value and the vertex's height value after simplification. By means of longest edge bisection, node *i*'s object space error metric ε_i is the maximum of vertical height difference of the midpoints of hypotenuses of all triangles, which are contained in the tile node *i*. Node *i*'s object space error metric ε_i is computed by using

$$\varepsilon_i = \max_{t \in T_i} \{\delta_{i,t}\} = \left| (Z_l + Z_r)/2 - Z_i \right| \tag{10}$$

In (10), *T* is the set of triangles contained in the tile node *i*. Z_l and Z_r is the height value of the two endpoints of the hypotenuse of triangle t in the triangle set *T*. Z_i is the height value of the midpoint of the hypotenuse of triangle *t* in the set of triangles *T*.

Cuboid bounding is designed to represent the tile node. The bottom of cuboid bounding is the projection of the tile node on a horizontal plane. The area of the bottom of cuboid bounding associates with the level of the tile node. The level of the tile node can be obtained by the R channel on GPU. The area of the bottom of cuboid bounding O_i is computed by using

$$O_i(x,y) = 4^{-(l-1)} \cdot WIDTH^2 \tag{11}$$

In order to include all sample vertexes in tile node, the height value of the cuboid bounding is the maximum of vertical height value difference of all vertexes of triangles, which will be contained in the tile node *i*. In order to save space, the height of the cuboid bounding and the ε_i in (10) is recorded by one data, which is the maximum of the two ε_i obtained by (10) and (12).

$$\varepsilon_i' = \max\{\max\{Z_{C_j} - Z_{C_i}\}, \varepsilon_i\} \tag{12}$$

Ci is the set of the midpoints of hypotenuses of triangles contained in the tile node *i*. *Cj* is the set of the two endpoints of hypotenuses of triangles contained in the tile node *i*. Z_{C_j} is the height value of vertexes in *Cj*. Z_{C_i} is the height value of vertexes in *Ci*.

The cuboid bounding of tile node *i* is computed by using

$$B_i = \{(x,y,z) \mid (x,y) \in O_i(x,y), z \le \varepsilon_i'\} \tag{13}$$

The volume of cuboid bounding designed in this article is much smaller than sphere bounding. The cuboid bounding designed in this article contains less tile sample vertexes. It means that the computation of tile vertexes in the cuboid bounding will became less and easily. As a result, it can reduce the amount of triangles and improve the speed in terrain rendering.

4.2. GPU LOD Selections

Screen space error metric is the error that projects object space error to view-dependent screen space error of each node. Screen space error metric is the important evaluation criterion for tile node. ρ (ε ,B,e) represents the node's screen space error metric, where ε is an object space error metric , and e is the viewpoint, B is the bounding that represents the tile node. Node i's screen space error metric ρ is computed by using

$$\rho(\varepsilon_i', B_i, e) = \lambda \cdot \varepsilon_i' / \left\| B - e \right\| \tag{14}$$

$\lambda = \omega / (2 \cdot \tan(\phi/2))$, where ω is the number of pixels along the field of viewpoint ϕ. $\left\| B - e \right\|$ is the Euclidean distance between the viewpoint e and the cuboid bounding.

The computation of tile node's object space error metric, cuboid bounding and screen space error metric only involves addition and multiplication arithmetic, which correspond to parallel computing requires. For example, the computation of tile node's object space error metric involves only two additions and one multiplication. Therefore, error metric can be computed by GPU. Compared to saturated error metric, GPU unsaturated error metric in this paper can decrease amount of triangles and improve the speed of terrain rendering greatly.

In the terrain rendering, according to viewpoint e, compare ρ against a user-specified screen error metric tolerance τ. We obtain

$$\begin{aligned} split(i) &\Leftrightarrow \rho > \tau \Leftrightarrow (\lambda \cdot \varepsilon_i' / \left\| B - e \right\|) > \tau \\ &\Leftrightarrow \lambda / \tau \cdot \varepsilon_i' > \left\| B - e \right\| \end{aligned} \tag{15}$$

The criterion of LOD selection and tile nodes simplification is according to (15). The computation of the criterion involves only three additions and seven multiplications, which also corresponds to GPU parallel computing requires. In the terrain rendering, Therefore, LOD selection and tile nodes simplification can execute by GPU acceleration.

In the terrain rendering, cracks can produce between different resolutions. We avoid cracks according to [14].

In this article, we make full use of GPU powerful computing capability and implement the computation of error metric and LOD selection on GPU, in order to save the time spending on preprocessing and reduce CPU-GPU bandwidth greatly. The computation of error metric is only aimed at visible tile nodes instead of the whole tile quad tree nodes. The amount of calculation will decrease largely. The cuboid bounding error metric designed in this article base on viewpoint and terrain feature, which can meet terrain rendering authenticity requirement.

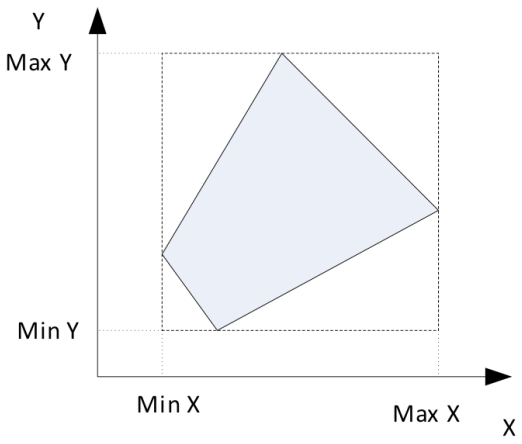


Fig. (5). Visible region coordinates.

5. GPU VISIBILITY ESTIMATION AND CPU DATA SCHEDULING

5.1. GPU Visibility Estimation

The visibility of tile nodes estimated the intersection between cone of viewpoint and terrain surface in traditional methods. The computation complexity of the visibility estimation will increase CPU computational burden. In this article, we map every terrain vertex in current view field into a float texture by way of GPU "Render To Texture" [9]. So every vertex's coordinates(x, y) was saved in the texture. Then put the texture read back to CPU. The visible region is the minimum bounding box instead of the real view field. The minimum bounding box, which represents the visible region, is made up four coordinates (min x, min y), (max x, min y), (max x, max y), (min x, max y). The visible region coordinates is shown in Fig. (**5**).

The visibility estimation by way of GPU "Render To Texture" will simplify the computation of visible region and avoid the computation of redundant data. The visibility estimation makes full use of GPU and reduces the CPU computation complexity.

5.2. CPU Data Scheduling

Large scale terrain is too big to store in the memory all the time. It is necessary to design a tile loading mechanism to schedule terrain data tile from external memory to internal memory dynamically. We use a fixed size of CPU buffer pool to store, manage and update the tile data. The size of CPU buffer pool is designed in integral multiple of the size of terrain tile so that any one tile can be stored in any one CPU buffer pool unit. Buffer pool is managed by CPU. With the viewpoint shifting at any time, we adopt an effective prefetching mechanism and asynchronous operation mode avoiding immediate dispatch and ensuring a smooth frame rate. According to the viewpoint position and movement, we compute the range of the buffer's data for next frame. The range is concerned with the speed of view shifting and the rendering frame.

6. RESULT AND ANALYSIS

Our prototype application is implemented for Windows XP32 and Visual Studio C++ 6.0, OpenGL and using the OpenGL Shading Language (GLSL) for programmable shaders. We use a 3.0GHz Intel Pentium IV PC, with 2G DDR2 of RAM, GeForce 9500GT graphics with 512M of graphics RAM, and SATA 500G disk in our experiments. We test a data set over the Puget Sound area in Washington was used, which is made up of 16384×16384 vertexes at 10 meter horizontal and 0.1 meter vertical resolution. The effectiveness of the proposed algorithm in this article can be demonstrated through the tile data organization performance experiment and error metric performance experiment.

6.1. Tile Data Organization Performance Experiment

The Puget Sound Area 16384×16384 terrain data is interpolated to simulate the 65536×65536 terrain data. Grid-based data set for each tile is 65×65, and thus the level of the double layers tile quad tree is 10. The level of the upper layer tile quad tree is 4, the level of the lower layer tile quad tree is 8 in the experiment.

We compared the data organization performance of CPU SLTQ (CPU single layer tile quad tree), GPU SLTQ (GPU single layer tile quad tree) and GPU DLTQ (our algorithm based on GPU double layers tile quad tree) algorithm. The comparison result is listed in Table **1**.

As seen from Table **1**, the frame rate of algorithms on GPU, which make full use of GPU powerful computing capability, is much higher than the frame rate of the algorithms on CPU. The preprocessing time of our algorithm based on GPU double layers tile quad tree is more than the algorithm based on single layer tile quad tree. But the visible tile selection time of our algorithm is less than the algorithm based on single layer tile quad tree obviously. The reduction of the visible tile selection time of our algorithm has proved that the double layers tile quad tree has an advantage of selecting tile nodes quickly by decreasing the depth of the quad tree. Our algorithm based on double layers tile quad tree in this

Table 1. Comparison of different data organization algorithms.

Algorithm	Preprocessing Time	Visible Tile Selection Time	LOD Rendering Time	Average Frame Rate
CPU SLTQ Algorithm	~27m33s	~4ms	~16ms	42.8
GPU SLTQ Algorithm	~27m55s	~3.8ms	~4ms	124.6
GPU DLTQ Algorithm	~29m40s	<1.6ms	<2.5ms	219.3

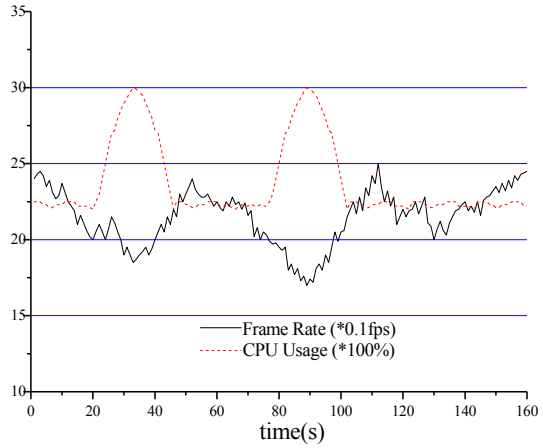


Fig. (6). Recorded frame rate and CPU usage over time.

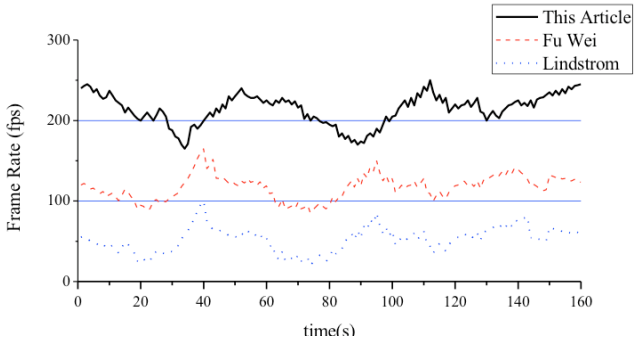


Fig. (7). Performance curves of frame rate.

article uses GPU two-dimensional texture mapping and executes tile error metric computation and LOD selection by GPU acceleration. Therefore, the terrain rendering time of our algorithm is the minimum, and the average frame rate of our algorithm is the maximum.

The size of CPU buffer pool is designed 16×16 tiles in the experiment. The length of one side is 64×16=1024 pixels. The same roaming path rendering performance curves of frame rate and CPU usage are shown in Fig. (**6**).

When the viewpoint shifts, tile data will be needed update and load between external memory and internal memory. Tile data's update and loading can occupy most CPU resources. Therefore, the CPU usage will reach a high level. But the CPU isn't used and CPU usage is still a lower level when the terrain rendering. As seen from Fig. (**6**), our algorithm can reach over 170 fps and satisfy the real-time requirement of terrain rendering.

6.2. Error Metric Performance Experiment

We compared the error metric performance of Lindstrom (CPU nested sphere and saturated error metric) [12], Fu Wei (GPU nested sphere and saturated error metric) [14] and This Article (cuboid bounding unsaturated error metric on GPU). The same roaming path rendering performance curves of the frame rate are shown in Fig. (**7**).

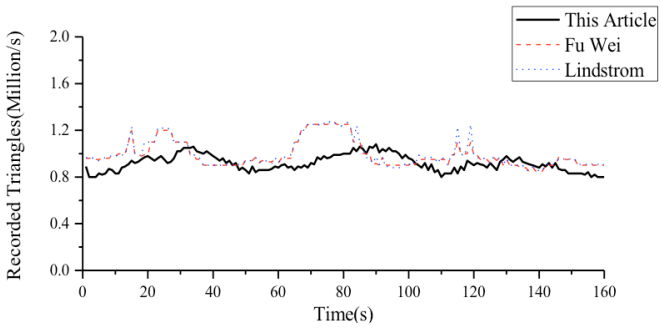
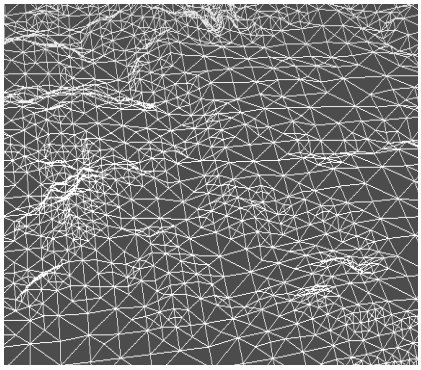
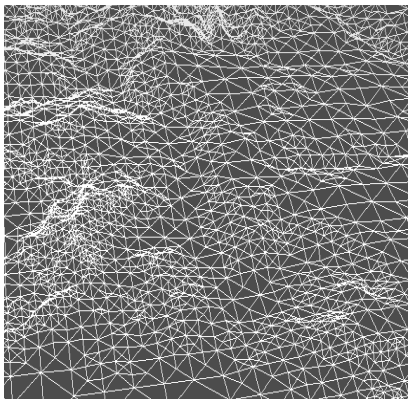


Fig. (8). Performance curves of the amount of triangles.



Unsaturated error metric (594K triangles)



saturated error metric (653K triangles)

Fig. (9). Terrain meshes screenshots of different error metric.

As seen from Fig. (**7**), Lindstrom implemented error metric computation on CPU; the average frame rate of Lindstrom is about 66fps. Fu Wei realized error metric computation on CPU and LOD selection on GPU, the average frame rate of Fu Wei is about 155fps. Our algorithm executes both error metric computation and LOD selection on GPU, the average frame rate of our algorithm is about 219fps. Compared to Lindstrom and Fu Wei, our algorithm based on cuboid bounding unsaturated error metric can reach higher frame rate.

The same roaming path rendering performance curves of the amount of triangles are shown in Fig. (**8**).

The volume of cuboid bounding designed in this article is much smaller than sphere bounding in Lindstrom and Fu Wei. The cuboid bounding contains less tile sample vertexes. It means that the computation of tile vertex in the cuboid bounding will became less and easily. As seen from Fig. (**8**), triangles number of our algorithm based on cuboid bounding unsaturated error metric is less than Lindstrom and Fu Wei obviously.

Compare the error metric of our algorithm based on cuboid bounding unsaturated error metric and Fu Wei based on GPU nested sphere and saturated error metric [14]. Terrain meshes Screenshot of the two algorithms are shown in Fig. (**9**).

As seen from Fig. (**9**), the amount of triangles generated by our algorithm based on cuboid bounding unsaturated error metric is 8.96 percent less than Fu Wei based on GPU nested

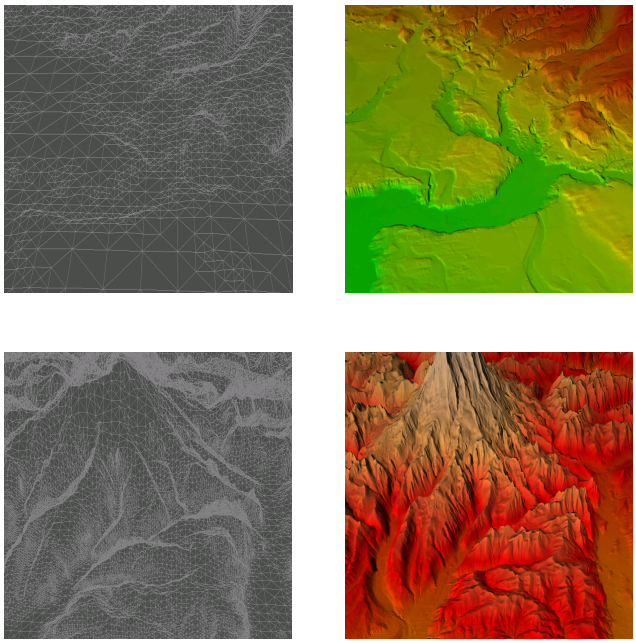


Fig. (10). Terrain meshes and textures screenshots in roaming.

sphere and saturated error metric. From terrain meshes Screenshots displays, our algorithm can not miss the terrain features, can satisfy the authentic requirement of terrain rendering.

Real-time rendering screenshots of terrain meshes and textures are shown as Fig. (**10**).

From Fig. (**10**), it can be seen that our algorithm based on double layers tile and cuboid bounding unsaturated error metric can load tile data quickly between external memory and internal memory, between CPU and GPU. The terrain rendering screenshots displays show smooth and fluent. All experiments' results can verify that our algorithm can greatly improve the speed of the terrain rendering and achieve a good visual effect.

CONCLUSION

According to data organization and LOD building, double layers tile quad tree for massive terrain data is built to reduce the depth of the tree, avoid selecting invalid tiles and calculation redundancy and decrease the computing time. Tile data in double layers tile quad tree are organized by Z-order space filling curve, in order to enhance data access locality and improve tiles' selection efficiency. The visibility estimation based on GPU render to texture reduces the CPU computation complexity. Map visible tile quad tree into two-dimensional texture on GPU makes full use of the characteristics of GPU multi-channel parallel processing and improves largely the speed of terrain rendering. The volume of cuboid bounding designed in this article is much smaller, which reduces the amount of triangles in terrain rendering. Cuboid unsaturated error metric on GPU implement the computation of error metric and LOD selection on GPU, in order to save the time spending on preprocessing and reduce CPU-GPU bandwidth greatly. The cuboid bounding error metric designed based on viewpoint and terrain feature can keep the 3D terrain feature. Experiments show that the algorithm in this article can improve the utilization rate of GPU in the terrain rendering and meet terrain rendering real-time and authenticity requirements. Future work includes researching eliminating cracks efficiently and grid-based compression algorithm to meet the massive complexity of terrain rendering.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

This work is supported by The National Natural Science Foundation, China (No.60970073). This work is supported by HeBei National Natural Science Foundation, China (No.F2012203084). This work is supported by HeBei National Natural Science Foundation, China (No.A2012203124).

REFERENCES

[1]. P. Lindstrom, and J.D. Cohen, "On-the-fly decompression and rendering of multiresolution terrain," In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Washington: ACM. 2010, pp. 65-73.

[2]. H.Y. Kang, H. Jang, C.S. Cho, and J.H. Han, "Multi-resolution terrain rendering with GPU tessellation," *The Visual Computer*, vol. 31, no. 4, pp. 455-469, 2014.

[3]. A. Asirvatham, and H. Hoppe, *Terrain Rendering Using GPU-based Geometry Clipmaps*, GPU Gems, Addison-Wesley, Boston, pp. 27-45, 2005.

[4]. T. Ulrich, "Rendering massive terrains using chunked level of detail control," *Course Notes of ACM SIGGRAPH. San Antonio: SIGGRAPH*, 2002.

[5]. S. Li, J. Ji, X. Liu, E.H. Wu, "High performance navigation of very large-scale terrain environment," *Journal of Software,* vol. 17, no. 3, pp. 535-545, 2006.

[6]. C. Dick, J. Schneider, and R. Westermann, "Efficient geometry compression for gpu-based decoding in realtime terrain rendering," *Computer Graphics Forum*, vol. 28, no. 1, pp. 67-83, 2009.

[7]. B. Li, and C.X. Zhao "A GPU based run-time quad-tree construction method for fast terrain rendering," *Journal of Computer-Aided Design & Computer Graphics*, vol. 22, no. 12, pp. 2259-2264, 2010.

[8]. J. Vanek, B. Benes, A. Herout, and O. Stava, "Large-scale physics-based terrain editing using adaptive tiles on the GPU," *Computer Graphics and Applications IEEE*, vol. 31, no. 6, pp. 35-44, 2013.

[9]. H. Liu, W. Cao, and W. Zhao, "Dynamic scheduling and real-time rendering method for large-scale terrain," *Computer Science*, vol. 40, no. 6, pp. 120-124, 2013.

[10]. J. Nie, D. Guo, L. Kong, and Y. Wang, "Multi-resolution terrain rendering seamlessly by function curve fitting," *Journal of Computational Information Systems*, vol. 7, no. 2, pp. 452-461, 2011.

[11]. M. Bader. "Two motivating examples: sequential orders on quad trees and multidimensional data structures," *An Introduction with Applications in Scientific Computing*, vol. 9, pp. 1-14, 2013.

[12]. L. Peter, and V. Pascucci, "Terrain simplification simplified: a general framework for view-dependent out-of-core visualization," *IEEE Transaction on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 239-254, 2002.

[13]. Y. Lu, "Study of the real-time rendering for large-scale terrain dataset," Ph.D. dissertation Zhejiang University 2003

[14]. W. Fu, Z. Ge, and W. Li, "A terrain rendering method with dynamic error metric for flight simulation," *Journal of Jilin University (Science Edition)*, vol. 50, no. 6, pp. 1175-1178, 2012.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014