Terrain Rendering With View-Dependent LOD Caching

Ming Fan Ueng Jung Hong Chuang
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, ROC
{mfueng, jhchuang}@csie.nctu.edu.tw

Abstract

Real-time and smooth rendering of a large-scale terrain data has been a challenging problem. In this paper, we propose a geometrically continuous view-dependent level-of-detail (LOD) modeling aiming to speed up the generation of terrain mesh and in the meantime achieve a satisfied image quality. The terrain data is subdivided into blocks and each of which will possesses its own LOD mesh that is dynamically determined according to the viewing parameters. Between two adjacent blocks, a dike structure is proposed that aims to provide a smooth blending between two meshes of different levels of detail, and hence remove cracks that usually occur in previous methods. We also propose a mechanism in LOD modeling that caches the LOD of a block for the possible reuse in the following frames after it is generated. Since LOD selection and generation in general requires computation on each node level, such a LOD caching can potentially contribute a considerable saving of computation time.

Key words: LOD, Terrain rendering, Caching

1. Introduction

A rendering system is a kernel for visual simulation and virtual reality applications. In such applications, we are very much concerned about the high-performance and real-time visual capability. This leads to the quest of high resolution, low latency, and high but constant frame rate in the visual display. In the past years, many techniques have been proposed. Among them, we mention fast view and back-facing culling, visibility culling, level-of-detail modeling, hybrid rendering, and image-based rendering.

As a special case of the general rendering system, a terrain rendering system usually takes a terrain grid with high-field values as input, and has found applications in flight simulations, tank simulations, and other GIS applications. Most applications usually cover a very large area, and hence require a large-sized terrain grid. This results in too many polygons to be efficiently rendered by the current hardware. Level-of-detail (LOD) modeling has been proven to be a very effective technique for reducing the number of polygons.

This paper describes techniques for removing cracks that occur between two adjacent blocks of different LOD, and for caching LOD of a block and possible reuse in the following frames. In the following sections, we review previous work, and we describe the dike structure for blending two different LOD models and the cache mechanism of LOD model, and finally we show several experimental results.

2. Related Work

LOD modeling techniques for terrain grid can be classified into two major mesh structures: regular square grid (RSG)[2,3,6,8] and triangulated irregular network (TIN) [4,7,9].

In RSG approach, terrain grid is usually subdivided into blocks to avoid global propagation in dependency checking during LOD construction [2,3,6,8]. Such a block subdivision also provides a good support in the view culling and paging mechanism. In [6], a quadtree structure is used for each block. The quadtree structure is explicitly and hierarchically constructed based on a regular and symmetric triangulation of the grid vertices. This hierarchical structure allows efficient derivation of a LOD model for new viewing parameters. During view dependent navigation, the delta segment projection for a node will be tested to see if the node should be simplified or refined. A block-LOD-reduction scheme is also used to reduce the LOD construction time by alleviating the testing at a huge number of nodes and allowing LOD be determined on the block basis. RSG approach has several advantages. For examples, Delaunay trianglation can be easily maintained for view-dependent selective refinement, switching between levels can be efficient and simple, and fast triangle strip can be easily constructed. It, however, produces for each block a mesh that is usually not optimal, and has cracks between two adjacent blocks of different LOD resolutions..

In TIN approach, vertices can be added and removed, or

connection can be modified in order to obtain a mesh that is better approximating the original shape [4,7,9]. As a result, a reduced mesh with better approximation; but less polygons is generally possible. Comparing to RSG approach, this approach usually requires more computation time, is more troublesome to locally modify a terrain model , and less efficient in performing collision detection.

3. The Proposed Terrain Rendering System

3.1 Overview

The terrain rendering system we implemented takes the RSG approach; that is, we take the terrain triangulation as in [6]. As a preprocessing, we divide the terrain grid into blocks with a dike between each pair of adjacent blocks. In run-time, blocks are first tested for view-volume culling for each new frame, and for each of those blocks intersecting the view volume, we check to see if its cached LOD can be reused in the new frame. That is, the cached LOD can be reused if its projected error with respect to the new viewpoint is within a pre-specified error, or a new LOD should be re-generated if the test fails. The test is block-based: rather than vertex-based, and thus can be very efficient. After the LOD of all blocks within the view volume are ready, we triangulate the dikes such that the LOD models of different resolutions can be smoothly blended.

3.2 Hierarchical Structures

Two hierarchical structures are proposed. A dependency hierarchy is used to facilitate the run-time selective refinement. Moreover, we construct a triangle tree in such a way that triangle mesh can be efficiently derived once terrain vertices are selected for the current LOD without traversing terrain vertices one more time.

According to the triangulation rule in RSG approach, a terrain of $(2^n+1)\times(2^n+1)$ can be simplified to 2n+1 levels; as shown in Fig. 1 for n=2. A triangle tree is a binary tree in which each node represents a triangle in the RSG triangulation. The refinement on each triangle results in two triangles, representing the children of the corresponding parent node. Fig. 2 shows the triangle trees for a 3×3 terrain grid.

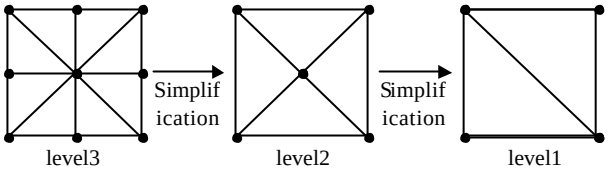


Fig. 1. Levels of LOD model.

While performing the refinement, a triangle in the RSG triangulation is subdivided into two triangles by adding a vertex on the bottom edge of the triangle. We call the top vertex of the original triangle is the mother or father vertex of the newly added vertex. The order that a vertex is selected for and added to the LOD model determines a hierarchy among terrain vertices. Vertices that are new in level 1 constitute the first level of the dependency hierarchy, and vertices that are newly added to level l form the l-th level of the dependency hierarchy. In the hierarchy, each vertex is associated with a father and a mother pointer pointing to its mother and father vertices. Fig. 3(c) is the dependency hierarchy for a 5 ×5 terrain grid shown in Fig. 3(a).

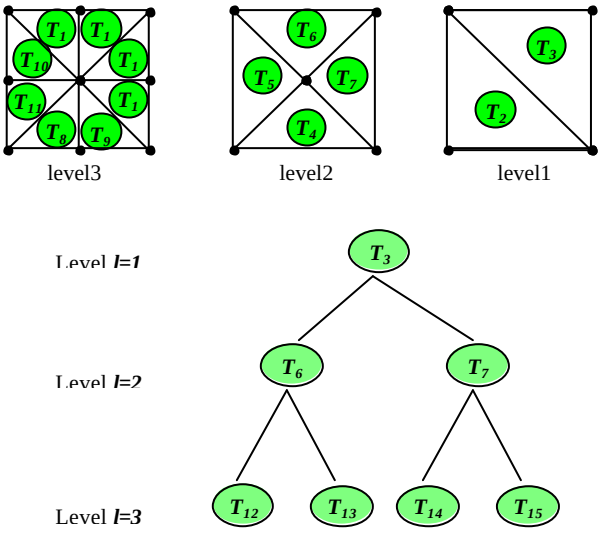
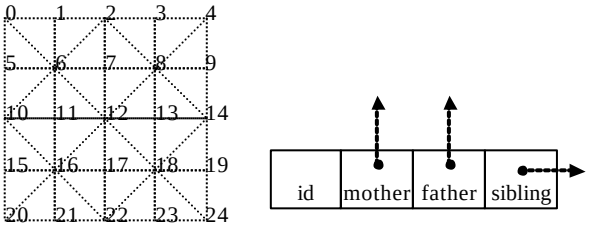
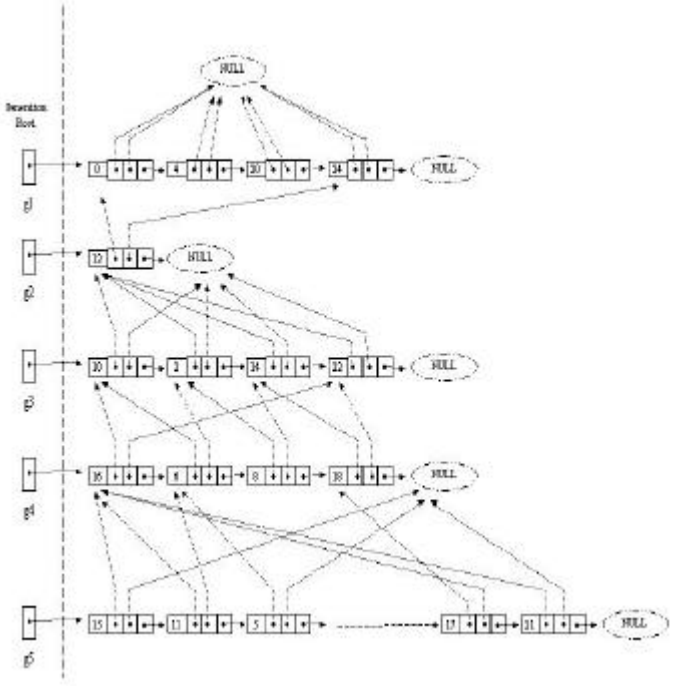


Fig. 2. Triangle binary tree.



(a) Vertex index (b) Data structure for vertex



(c) dependency hierarchy

Fig. 3. Dependency hierarchy.

3.3 Dynamic Selective-Refinement

In navigation phase, the dependency hierarchy is traversed to derive the mesh of a desired resolution. When a node is visited, we do screen-error test to see if the projection of its height difference exceeds a pre-specified tolerance. If so, the vertex is selected, and in the meantime, its parent vertices are locked and selected without the screen-error test.

Each vertex is associated with two more variables, namely *active* and *lock*. The variable *active* is a Boolean recoding the selection state of the vertex. The variable *active* is TRUE when the vertex is selected, and FALSE otherwise. The variable *lock* for a vertex *v* is an integer recording the number of vertices that are children of *v* and are either selected or locked. A nonnegative *lock* means that *v* is locked, and a zero *lock* represents that *v* is not locked.

Two operations are involved in selecting the vertex *v*. *Dependency* operation switches the *active* variable of *v* from FALSE to TRUE while *unlocking* operation does oppositely. In dependency operation, the variable *lock* of parent vertices of *v* must be increased by 1. In case *v* has *lock* value 0, parent vertices of *v* must repeatedly perform dependency operation. In unlocking operation, the variable *lock* of parent vertices of *v* must be decreased by 1. In case *v* has *lock* value 1, parent vertices of *v* must repeatedly perform unlocking operation.

The dependency hierarchy is traversed in a bottom-up fashion. If a vertex is locked, its corresponding triangles are put into the display list. If a vertex is not locked and passes the screen-error test, the *active* variable becomes FALSE and unlocking operation is performed, provided that its *active* variable is TRUE. If a vertex is not locked and fails to pass the screen-error test, its *active* variable becomes TRUE and dependency operation is performed, provided that its *active* variable is FALSE, and its corresponding triangles are put into the display list.

4. Removing Cracks

A dike structure is proposed to remove cracks occurring between two adjacent blocks of different LOD resolutions. See Fig. 4 for illustration.

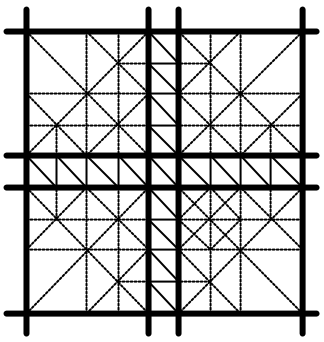


Fig. 4. Dike structure.

After LOD models are obtained for all blocks, we begin to triangulate the dike area one by one without altering the selection state of block's boundary vertices. In our implementation, each dike area is first completely triangulated and then simplified based on edge collapsing guided by the selection status of block's boundary vertices.

5. LOD Caching

The screen-error test mentioned in previous section takes the projection of vertex's height difference into account. As shown in Fig. 5, the height difference of *B*, denoted as $?_B$, is defined as the deviation in z-direction from *B* to Δ AEC.

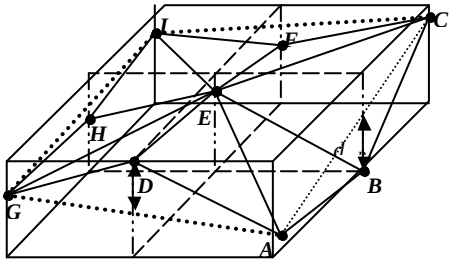


Fig. 5. Height difference on a vertex.

Following the formula in [6], vertex v will pass the screen-error test if

$$?^2_{screen}(\mathbf{e},v) ? \frac{d^2?^2?^2 \left(?e_x ? v_x ?^2 ? ?e_y ? v_y ?^2\right)}{\left(?e_x ? v_x ?^2 ? ?e_y ? v_y ?^2 ? ?e_z ? v_z ?^2\right)} ? ?^2$$

where **e** is the eye point, *d* is the view plane distance, λ is the ratio of the unit length in world coordinate system over the pixel size in the screen coordinate system, δ is the height difference on vertex *v*, and τ is a user-specified error tolerance. The above formula can be viewed differently to define a s called *allowable height difference* of *v* as follow:

$$?^2_{allowable}(\mathbf{e},v) ? \frac{?^2 ?\left(?e_x ? v_x ?^2 ? ?e_y ? v_y ?^2 ? ?e_z ? v_z ?^2\right)}{d^2?^2\left(?e_x ? v_x ?^2 ? ?e_y ? v_y ?^2\right)}$$

As a result, the screen-error test is equivalent to testing if $?_v ? ?_{allowable}(\mathbf{e},v)$.

The LOD caching mechanism aims to cache the LOD of a block for possible reuse in the following frames with the requirement that the screen error is within a user-specified tolerance. We first denote the *projection bound of the delta allowable height difference* as *s* (in pixel unit), and suppose that a cached LOD model can be

reused if, for a new viewpoint, the projected delta allowable height difference of the LOD model is less than or equal to *s*.

Consider a vertex v_i, we have $\delta_{allowable}(\mathbf{e}_0, v_i)$ and $\delta_{allowable}(\mathbf{e}_1, v_i)$, respectively for viewpoints $\mathbf{e}_0$ and $\mathbf{e}_1$. By replacing τ with *s*, we obtain the bound on delta allowable height difference of v_i with respect to $\mathbf{e}_0$ as follows:

$$\delta(\mathbf{e}_0, v_i) \le \frac{s \cdot \left| e_{0x} \cdot v_{ix} \right| + \left| e_{0y} \cdot v_{iy} \right| + \left| e_{0z} \cdot v_{iz} \right|}{d \cdot \sqrt{\left| e_{0x} \cdot v_{ix} \right| + \left| e_{0y} \cdot v_{iy} \right|}}$$

We then claim that the selection state of v_i with respect to $\mathbf{e}_0$ can be preserved while viewing from $\mathbf{e}_1$ is the delta allowable height difference $\Delta\delta_{allowable}(\mathbf{e}_0, \mathbf{e}_1, v_i)$ is less than $\delta(\mathbf{e}_0, v_i)$, where $\Delta\delta_{allowable}(\mathbf{e}_0, \mathbf{e}_1, v_i)$ is $\left\| \delta_{allowable}(\mathbf{e}_1, v_i) - \delta_{allowable}(\mathbf{e}_0, v_i) \right\|$. In such case, we can show that preserving the selection state of v_i results in a projected height difference bounded by $\tau + s$.

Next, we extend the preserving of vertex's selection state to the LOD caching of a block. For the LOD caching of a block, we, in principle, need to check if $\Delta\delta_{allowable}(\mathbf{e}_0, \mathbf{e}_1, v_i) \le \delta(\mathbf{e}_0, v_i)$ for all v_i in the block. This is, however, very time consuming. Since $\delta(\mathbf{e}_0, v_i)$ becomes smaller when the distance between v_i and $\mathbf{e}_0$ gets smaller, it is reasonable to say that $\delta(\mathbf{e}_0, v_i)$ is larger than or equal to the minimum of $\delta(\mathbf{e}_0, v_{lt})$, $\delta(\mathbf{e}_0, v_{rt})$, $\delta(\mathbf{e}_0, v_{lb})$, and $\delta(\mathbf{e}_0, v_{rb})$, provided that $\mathbf{e}_0$ and $\mathbf{e}_1$ are outside the block; see Fig. 6.

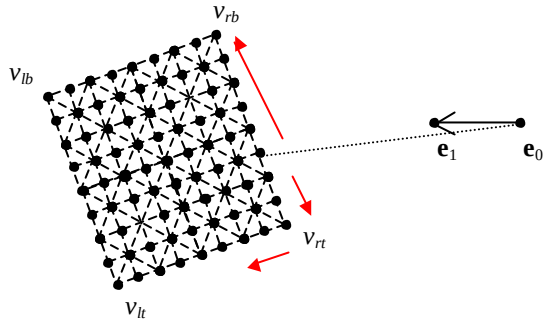


Fig. 6. Variation on tolerable height deviation.

6. Experimental Results

We have implemented the proposed scheme using C language, OpenGL, and GLUT library. Experiments have been performed using terrain data of Dan-Shoei River. Results are obtained on a PC with Pentium III 660Mhz CPU, 128MB Ram, and GeForce 256 3D graphics card.

The terrain data includes an area of 26,400m×26,400m, and is divided into 20×20 blocks, each of which has 33×33 grid vertices. A complete triangulation of this terrain data has 868,488 triangles. We set up a navigation path with height about 1,000m, 40 degrees for field of view, and a display window of 800×800 pixels.

Table 1 depicts the performance of LOD caching mechanism based on several different τ and different *s* for each τ. More detail analysis is shown in Table 2. Using LOD caching, we have seen a 25% to 46% speed-up in frame time and 92% to 98% speed-up in LOD construction time. Note that the LOD models obtained using LOD caching have less number in triangles, ranging from 2.2% to 8.7% in our experiment. Our experience shows that the change ranges from 3% to 4% when $s = 0.1\,\tau$. Figures 7, 8, and 9 show the performance plots for various τ.

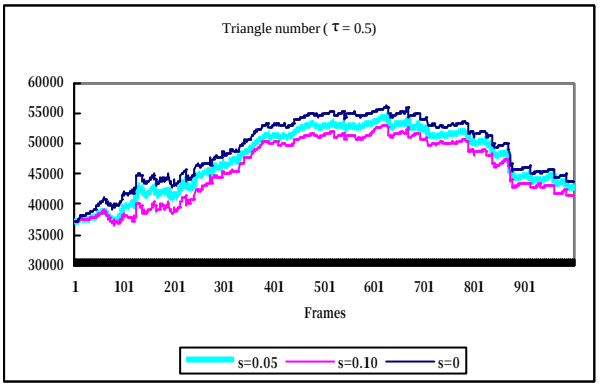
τ (pixel)	*S* (pixel)	Average triangle number	Average LOD generation time (ms/frame)	Average rendering FPS
0.5	0.05	47,487	1.2	6.9
	0.10	45,972	0.7	7.1
	-	49,097	34.8	5.5
1.0	0.05	24,595	1.4	13.7
	0.10	24,025	0.8	14.0
	0.20	22,950	0.6	14.9
	-	25,158	23.7	10.2
2.0	0.10	11,028	0.9	32.1
	0.20	10,770	0.7	32.4
	-	11,277	13.3	22.1

Table 1. Performance of LOD caching - 1.

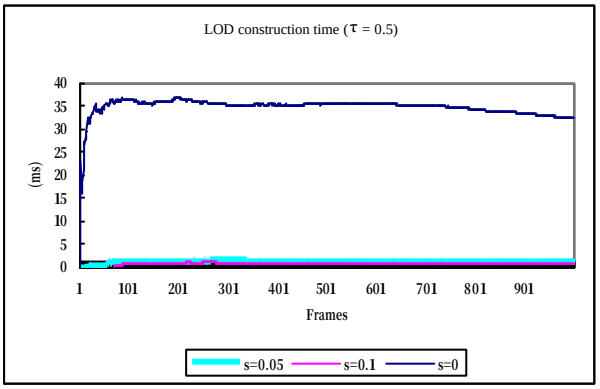
τ (pixel)	*S* (pixel)	Change in average triangle number	Average LOD generation time		(c) Average gain factor on frame time
			(a) Gain factor on LOD generation	(b) Gain factor on frame time (due to LOD caching)	
0.5	0.05	-3.3%	96.7%	90.1%	25.4%
	0.10	-6.4%	98.1%	83.2%	30.3%
1.0	0.05	-2.2%	94.3%	89.2%	34.1%
	0.10	-4.5%	96.5%	84.8%	37.6%
	0.20	-8.8%	97.3%	74.5%	46.0%

2.0	0.10	-2.2%	92.9%	88.6%	45.5%
	0.20	-4.5%	94.5%	90.0%	46.7%

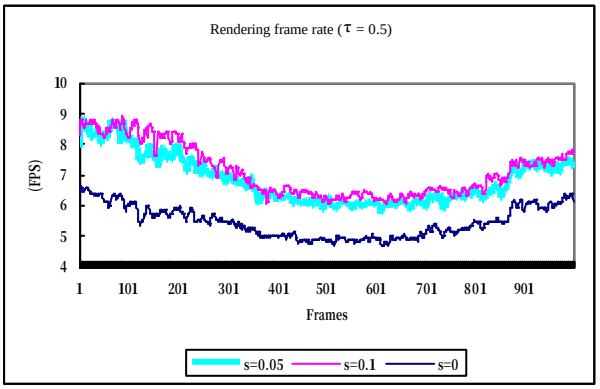
Table 2. Detail analysis on performance of LOD caching.



(a)

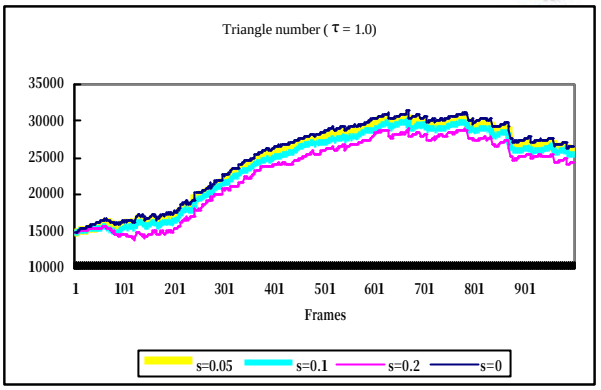


(b)

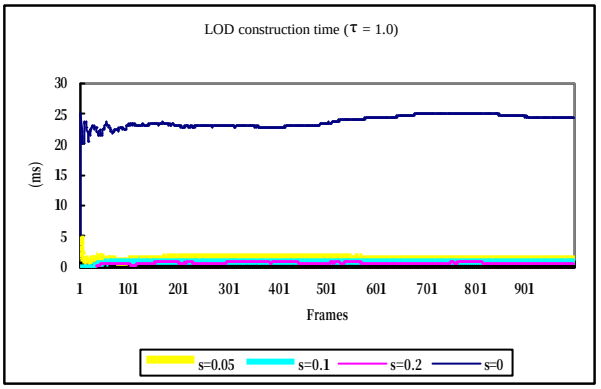


(c)

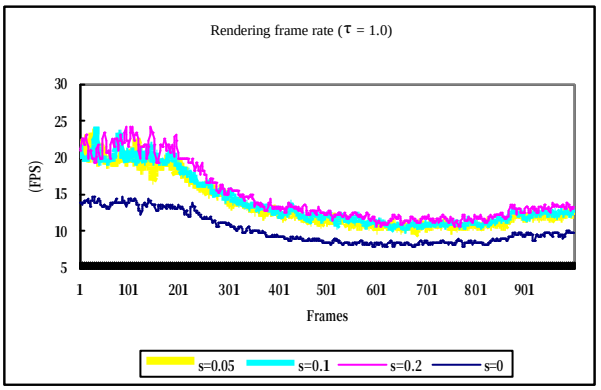
Fig. 7. Performance plot for $\tau = 0.5$.



(a)

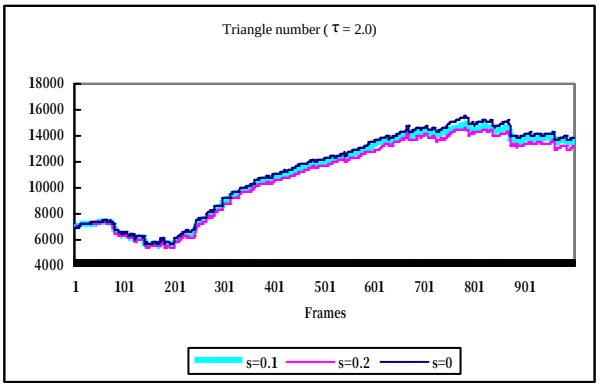


(b)

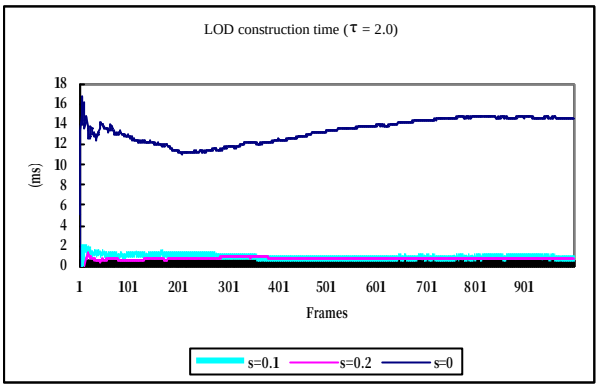


(c)

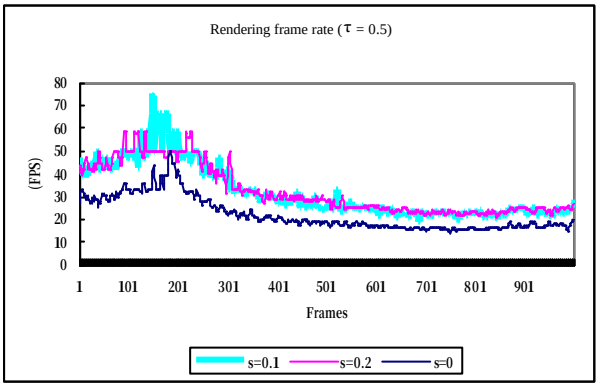
Fig. 8. Performance plot for $\tau = 1.0$.



(a)



(b)



(c)

Fig. 9. Performance plot for $\tau = 2.0$

To do a better examination on quality performance of the proposed LOD caching mechanism, we count the number of vertices that should be selected but are not selected due to LOD caching; that is, those vertices that have projected height difference exceeding τ; but are not selected. Table 3 depicts that, when $s = 0.1\ \tau$, the percentage of those vertices is bounded by 2%. Figures 10 and 11 are two images obtained in navigating the Dan-Shoei River.

Table 3. Quality performance of LOD caching.

τ (pixel)	S (pixel)	Average number of selected vertices	Vertices: should be selected; but not selected	
			Average number	%
0.5	0.05	23,731	374	1.5%
	0.10	22,971	782	3.4%
1.0	0.05	12,239	94	0.7%
	0.10	11,953	205	1.7%
	0.20	11,414	434	3.8%
2.0	0.10	5,425	35	0.6%
	0.20	5,295	74	1.3%

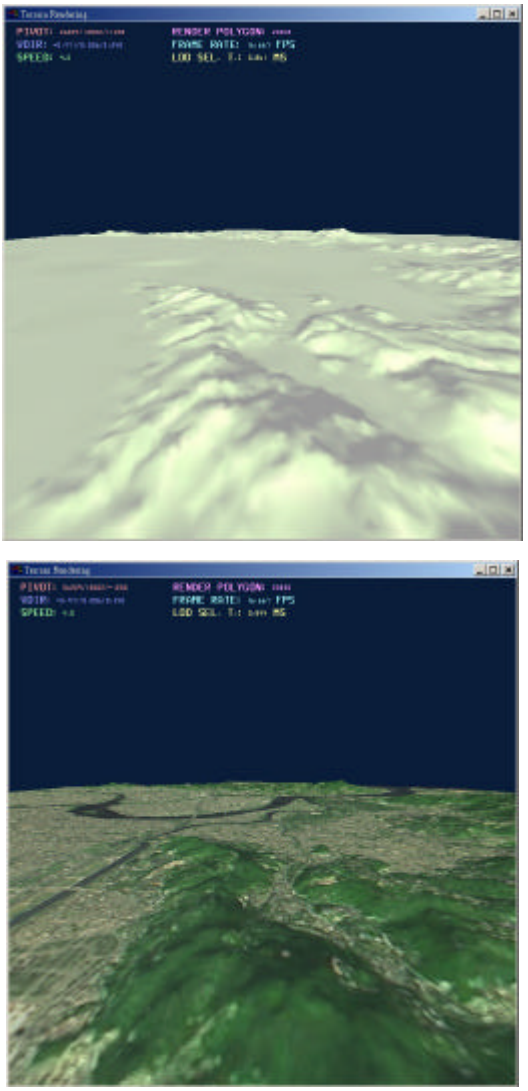
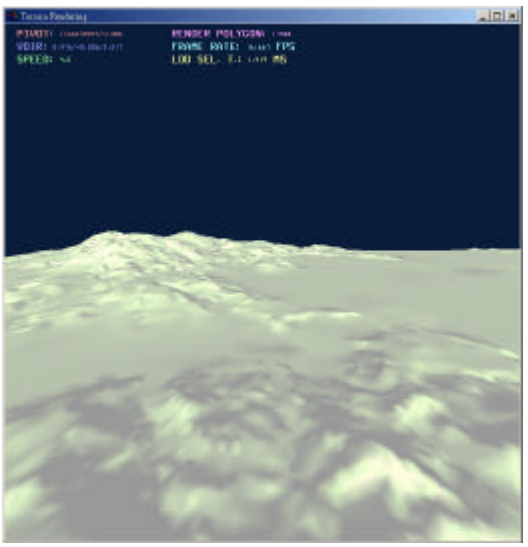


Fig. 10. Terrain image of Dan-Shoei River.



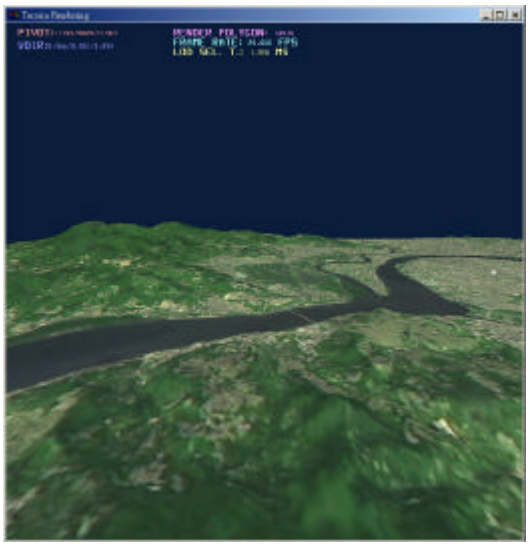


Fig. 11. Another terrain image of Dan-Shoei River.

4. Concluding Remarks

We have presented a terrain rendering system in which a dike structure and a LOD caching mechanism have been proposed to, respectively, remove cracks usually occurring in the boundary of adjacent blocks and speed up the LOD selection by reusing previously constructed LOD models. The experiments we have done revealed that dike structure successfully blends two LOD models of different resolution, and the LOD caching mechanism is able to speed up the LOD construction by 92-98%, and the frame time by 25-46%. Among future study plans, we will focus on frame-time control and the integration of hybrid rendering techniques into terrain rendering systems.

References

1. M. D. Berg, and K. T. G. Dobrindt, *"On Levels of Detail in Terrains"* Tech. Rep. UU-CS-1995-12, Department of Computer Science, Utrecht University, April 1995.

2. M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. Mineev. *"ROAMing Terrain (Real-Time Optimally Adapting Meshes)."* Proceeding of IEEE Visualization `97.

3. C. S. Fahn and S T. Wu. *"The View-Dependent Real-Time Rendering of Large-Scale Terrain in Continuous Level of Detail."* In Proceeding II of NCS99, Pages B374-B381.

4. R. J. Fowler and J. J. Little. *"Automatic Extraction of Irregular Network Digital Terrain Models."* Computer Graphics (Proceeding of SIGGRAPH `79), Vol. 13, No. 2, pages 199-207.

5. R. Klein and D. Cohen-Or. *"Incremental View-dependent Multiresolution Triangulation Terrain."* Proceeding of Pacific Graphics, 1997.

6. P. Lindstrom, D. Koller, W. Ribarsky, W. Hodges, L. Faust, and G. Turner. *"Real-Time, Continuous Level of Detail Rendering of Height Fields."* In Proceeding of ACM SIGGRAPH `96.

7. H. Hoppe. *"Smooth view-dependent level-of-detail control and its application to terrain rendering."* Proceeding of IEEE Visualization `98, October 1998, pages 35-42.

8. R. Pajarola, *"Large Scale Terrain Visualization Using The Restricted Quadtree triangulation."* Proceeding of IEEE Visualization `98.

9. M. F. Polis, and D. M. Mckeown. *"Iterative TIN Generation from Digital Elevation Models."* In Proceeding of IEEE Conference on Computer Vision and Pattern Recognition, June 16-18, 1992, pages 787-790.

10. J. Shade, D. Lischinski, D. H. Salesin, T. DeRose and J. Snyder. *"Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments."* Proceeding of ACM SIGGRAPH `96, pages 75-82.