Dynamic View-dependent Multiresolution Terrain Visualization

Din-Chang Tseng * and Chung-Chieh Huang

Institute of Computer Science and Information Engineering
National Central University, Chung-li, Taiwan

* Email: tsengdc@ip.csie.ncu.edu.tw

Abstract

Large-area terrain visualization is important to multimedia and military applications. However, such a huge amount of terrain data is not easily able to process on a general personal computer. Thus, how to efficiently process and display a large-area terrain data is the main challenge. In this study, we combine the proposed techniques of multiresolution terrain modeling, view-dependent rendering, and dynamic loading with the digital terrain models and satellite images to dynamically browse the large terrain models and scene environment on a general personal computer.

Keywords: multiresolution terrain modeling, view-dependent browsing, dynamic loading, scene visualization, flight simulation

1. Introduction

Terrain models were commonly used in many applications, including flight simulation, geographical analysis, and computer games [1, 2, 12, 13]. In these applications, an efficient representation and browsing for the terrain models is necessary. Terrain models are generally represented by triangle meshes. A terrain model is usually a complicated geometric object; to browse such a model is not realized in real-time without reducing the number of triangles even if a high-performance graphics accelerator is used. Multiresolution modeling is one way to achieve the real-time visualization for complicated models. The idea of using multiresolution modeling is to show a coarser (lower-resolution) model while the viewpoint is far away from the terrain and show a finer (high-resolution) model while the viewpoint is near the terrain; then the rendering performance is improved without reducing the visual quality [3-7, 10, 15].

A view-dependent browsing is a level-of-detail (*LOD*) control framework to selectively refine or coarsen the mesh according to the view parameters. This framework efficiently determines the coarsest model to satisfy quality requirements to further improve the rendering performance [9, 11, 14].

It is impractical to load the entire terrain model into memory if only a small portion of the terrain is visible; thus a large terrain model is partitioned into blocks and then the necessary terrain blocks are dynamically loaded into the memory for rendering as viewpoint changing. Dynamic loading does not only increase rendering speed, but also reduce the memory requirement.

In this study, the techniques of multiresolution modeling, view-dependent visualization, and dynamic loading are combined for large terrain visualization to improve the visualization performance without reducing the visual quality; moreover, (*i*) cracks between two adjacent blocks with different resolutions are eliminated by the proposed re-triangulated boundary method; (*ii*) the active block number is automatically adapted according to the height of the viewpoint; (*iii*) the flight path is predicted to pre-load the necessary blocks in advance to reduce the suspending phenomenon during loading necessary terrain blocks in a one-processor computer; (*iv*) a runtime geomorph method is used to smoothly swap different-resolution terrain blocks.

The proposed visualization system is briefly described as follows. (*i*) A large terrain model is partitioned into blocks. (*ii*) All blocks are constructed as view-dependent multiresolution structures. (*iii*) Based on the current view parameters, the necessary terrain blocks are dynamically loaded. (*iv*) According to the defined screen-space error, a refining/simplifying procedure is applied on all active blocks. (*v*) The cracks are eliminated by the proposed re-triangulation boundary method. (*vi*) Based on the last flying directions, the terrain blocks on the predicted path are pre-loaded. (*vii*) The runtime geomorph method is used to smoothly swap different-resolution terrain blocks.

2. Multiresolution Modeling

The progressive mesh (*PM*) proposed by Hoppe [8] is utilized to generate the multiresolution terrain models. A *PM* representation for an arbitrary triangle mesh $\hat{M}$ consists of a coarse base mesh M^0 and a sequence of n refinement transformations called *vertex split* as shown in Fig. 1. A *PM* representation for $\hat{M}$ is obtained by carefully simplifying it using n successive *edge collapse* transformations as shown in Fig.1. In the process of an edge collapse, two vertices on the ends of an edge are merged and the two triangles that share the edge are vanished. A vertex split is the inverse transformation of an edge collapse; which

replaces a vertex with two new vertices and two new triangles are generated.

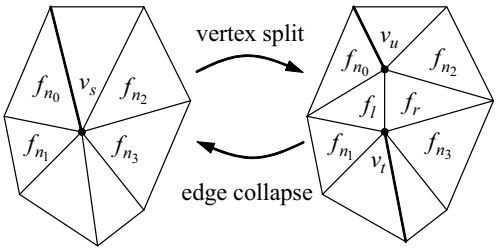


Fig. 1. An example of vertex split and edge collapse transformations.

In the construction of a *PM*, the sequence of the edge collapse transformations must be chosen carefully since it determines the qualities of the intermediate meshes. We construct a *PM* structure for a terrain mesh $\hat{M}$ by iteratively simplifying the mesh using edge collapse (*ecol*). In each iteration, the algorithm performs the *ecol* with the minimal approximation error. This procedure continues until the mesh can't be simplified further:

$$(\hat{M} = M^n) \xrightarrow{ecol_{n-1}} \cdots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 .$$

Then the simplest mesh together with the *vsplits* obtained by reversing the *ecol* sequence form a *PM* structure:

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \cdots \xrightarrow{vsplit_{n-1}} (M^n = \hat{M}) .$$

We use L_∞ norm to estimate the approximation error E introduced by an *ecol*. That is, for an *ecol* transformation that unifies two vertices to generate a new vertex v, we calculate the maximum elevation difference between the simplified area in the transformed mesh and the original, fully detailed mesh $\hat{M}$, as follows. Formally, the error is calculated by the formula

$$E^N(v) = \max_{\boldsymbol{p} \in D_v} \left\| \phi_{N_v}(\boldsymbol{p}) - \phi_{\hat{M}}(\boldsymbol{p}) \right\|, \quad (1)$$

where $\phi_H : \boldsymbol{R}^2 \to \boldsymbol{R}$ is the scalar field of height field H; N_v is the height field formed by the union of triangles incident to v in the transformed mesh; D_v is the domain of ϕ_{N_v}.

3. View-dependent Visualization

A vertex hierarchy for a *PM* is constructed for managing data structure and achieving view-dependent browsing. At first, the vertices of the base mesh are inserted into the hierarchy as roots. Then for each *vsplit*(v_t, v_u, v_s), vertices v_t and v_u are inserted as the children of v_s. The parent-child relation on the vertices establishes a vertex hierarchy as shown in Fig. 2. A selectively refined mesh corresponds to a "vertex front" through this hierarchy, such as M^0 and M in Fig. 2. At runtime, selective refinement based on the view frustum and screen-space error is achieved by moving a vertex front up and down through the hierarchy. This operation corresponds to a sequence of edge collapse or vertex split transformations.

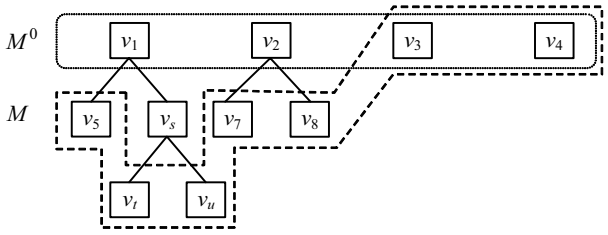


Fig. 2. A vertex hierarchy.

As shown in Fig. 3, if the projected length of distance between the approximated mesh M^A and the original $\hat{M}$ is greater than the pre-defined threshold τ, then vertex v_s will be split into v_t and v_u. In this study, the screen-space error is defined as the ratio of screen-projected deviation of the distance between the approximated mesh M^A and the original $\hat{M}$ to the height of the screen. We compute the screen-space error for each active vertex v.

As shown in Fig. 4, for each vertex v in the hierarchy, we compute the radius r_v of a bounding sphere S_v that is centered at v and bounds all its descendants. In particular, if v is a leaf, S_v is the smallest sphere that is centered at v and bounds N_v. For each vertex v at leaf, we compute r_v as the radius of the smallest sphere that is centered at v and bounds all its adjacent vertices in $\hat{M}$. Then make a postorder traversal of the hierarchy to assign each parent vertex v_s the radius of the smallest sphere S_{v_s} centered at v_s that bounds the spheres S_{v_u} and S_{v_t} as one example shown in Fig. 4.

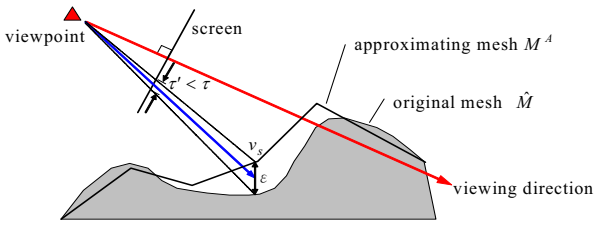


Fig. 3. Screen space error.

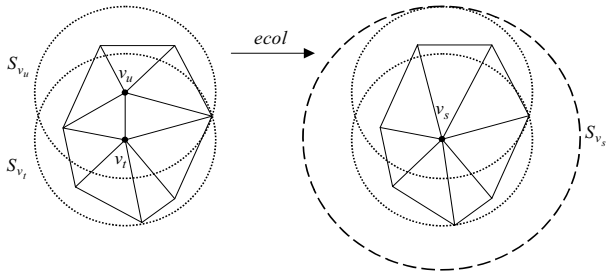


Fig. 4. Computation of bounding spheres.

A geomorph [9] consisting of meshes built by a sequence of smooth changes of vertices is used to reduce the popping effects during mesh transformations. For any two selectively refined meshes M^A and M^B, we construct a mesh $M^G(\alpha)$ whose vertices vary as a function of a parameter $0 \le \alpha \le 1$, such that $M^G(0)$ looks identical to M^A and $M^G(1)$ looks identical to M^B.

4. Dynamic Loading

The dynamic loading module is an on-line manager for terrain blocks. It decides (*i*) which terrain blocks should be loaded into memory while the viewpoint is changed and (*ii*) to eliminate cracks between two blocks with different boundary structures.

4.1 *Dynamic loading management*

We partition a large terrain model into several blocks, and then simplify all blocks with four corners fixed to product a multiresolution terrain model with view-dependent *PM* structure. These blocks will be saved as a terrain database with a 2D index, and queried by dynamic loading management.

The steps of the dynamic loading management are described as follows. At first, finding a terrain block called *center block* where the viewpoint is located. Then, according to the height of the viewpoint to load $(2(n+p)+1)\times(2(n+p)+1)$ blocks surrounding the center block as active blocks. The inner $(2n+1)\times(2n+1)$ blocks are modeled as the view-dependent *PM* structure; the outer *p*-layer blocks are modeled as the two-triangle mesh (*TM*) which is the coarsest mesh only composed of two triangles.

4.2 *Adaptive terrain blocks*

The proposed terrain visualization system can automatically change the number of active terrain blocks according to the height of the viewpoint. In general, the less an included angle θ between the view direction and the ground is, the more blur the watched terrain. We assume the height of the viewpoint is a, the minimum visible angle θ, and the farthest visible location to the location of the viewpoint on the ground is b as shown in Fig. 5; then

$$b = a \cot\theta . \qquad (2)$$

The higher the viewpoint is, the farther the visible location is. The default value of θ is 20 degrees and can be adjusted by the user. From Eq.(2), several threshold values for a are defined for changing the number of active terrain blocks.

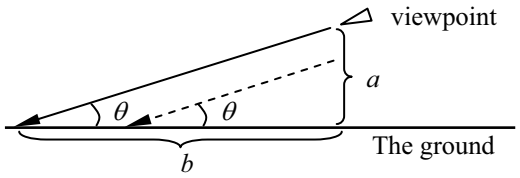


Fig. 5. The farthest visible distance is defined based on the included angle between the view direction and the ground.

4.3 *Re-triangulation*

Cracks between two adjacent blocks with different resolutions are always generated. We here solve the boundary matching problem by directly triangulating unmatched triangles on the boundary as one example shown in Fig. 6.

Some sliver (*i.e.*, long and narrow) triangles may appear if two unmatched vertices in two adjacent terrain blocks are too close. The solution is that if the distance of two adjacent mismatched vertices is less than a tolerance τ, the two unmatched vertices are reset to their midpoint.

A triangle located at a block corner is generally adjacent to two or more triangles in two neighboring blocks. The triangles may need to be split twice and the two splits are interdependent. We must care the split order; otherwise, the wrong structure would be generated.

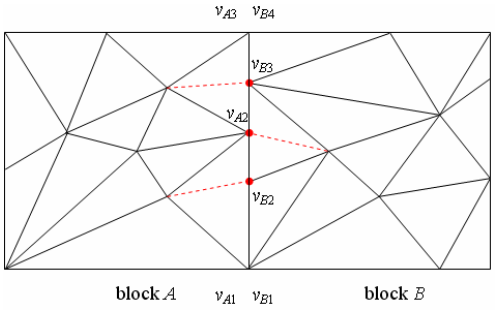


Fig. 6. Triangulation for unmatched triangles.

5. Experiments

We here give experimental results of progressive meshes, view-dependent multiresolution modeling, and dynamic loading.

The area shown in Fig. 7(a) is a fully detailed mesh of 129×129 vertices at ground resolution 40*m*×40*m* that covers the bank and the dam body of the Shih-Men dam, Taiwan. We constructed the *PM* structure for this area and several intermediate meshes are shown in Fig. 7.

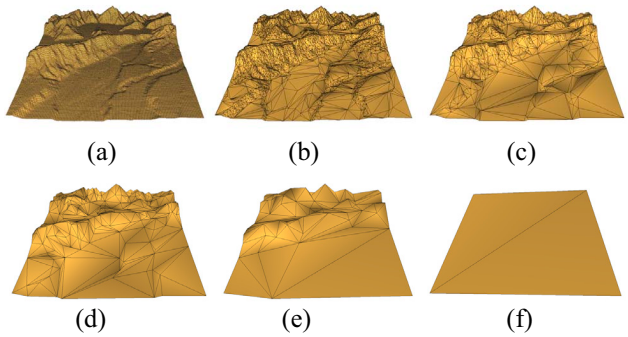
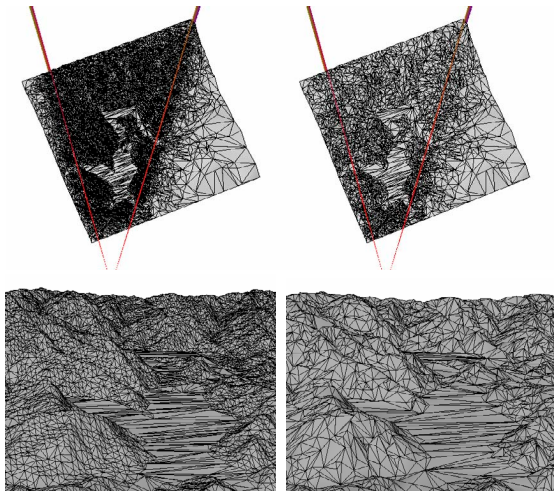


Fig. 7. Progressive meshes. (a) A fully detailed terrain mesh with 32768 triangles. (b) 8109 triangles. (c) 1978 triangles. (d) 467 triangles. (e) 109 triangles. (f) 2 triangles.

We choose a 6.4 *km*×6.4 *km* terrain mesh to evaluate the performance of view-dependent progressive meshes. First we build a *PM* structure for the mesh; then we adjust the screen space error tolerance τ and perform the selective refinement according to the view-dependent criteria. Two examples with different error tolerances are shown in Fig. 8. Comparing with the view-independent progressive meshes, we can find that the view-dependent progressive meshes use fewer triangles while produce better rendering results.



(a) (b)

Fig. 8. View-dependent progressive meshes with (a) $\tau = 0.2\%$ (19717 triangles) and (b) $\tau = 0.8$ (6791 triangles).

We used a 64 *km*×64 *km* terrain located in northern Taiwan for experiments of dynamic loading. The fully detailed terrain mesh consists of 5120000 triangles. We first partition the mesh into 10×10 blocks and build a *PM* structure for each block independently. Note that there is a 40*m* gap between two adjacent blocks. The generated *PM* structure is about 2.7*M* bytes for each block. The images for texture mapping are also partitioned into tiles. A 1024×1024 24-bit bitmap tile is associated with a block. In the *PM* construction, the four corner vertices of a block are kept immovable. During terrain browsing, we always maintain a 3×3 array of loaded blocks in main memory. An example of textured mapped block array is shown in Fig. 9.

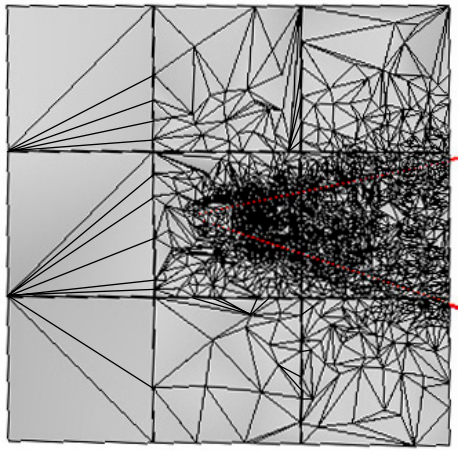


Fig. 9. An example of combining techniques of multiresolution modeling, view-dependent visualization, and dynamic loading, where a texture mapped 3×3 terrain block array with 453152 triangles.

References

[1] Cignoni, P., E. Puppo, and R. Scopigno, "Representation and visualization of terrain surfaces at variable resolution," *The Visual Computer*, Vol.13, No.5, pp.199-217, 1997.

[2] Cohen-Or, D. and Y. Levanoni, "Temporal continuity of levels of detail in Delaunay triangulated terrain," in *Proc. Visualization'96*, San Francisco, California, Oct.27–Nov.1, 1996, pp.37-42.

[3] Cohen, J., A.Varshney, D.Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *Proc. SIGGRAPH'96*, New Orleans, LA, Aug.4-9, 1996, pp.119-128.

[4] De Floriani, L., P. Marzano, and E. Puppo, "Multiresolution models for topographic surface description," *The Visual Computer*, Vol.12, pp.317-345, 1996.

[5] Erikson, C., *Polygonal Simplification: An Overview*, Tech. Report of Dept. Com. Sci., Univ. North Carolina at Chapel Hill, TR96-016, 1996.

[6] Garland, M. and P. S. Heckbert, "Surface simplification using quadric error metrics", in *Proc SIGGRAPH'97*, Los Angeles, CA, Aug.3-8, 1997, pp.209-216.

[7] Heckbert, P. S. and M. Garland, "Multiresolution modeling for fast rendering," in *Proc. Graphics Interface'94*, Banff, Alberta, Canada, May 1994, pp.43-50.

[8] Hoppe, H., "Progressive meshes," in *Proc. SIGGRAPH'96*, New Orleans, LA, Aug.4-9, 1996, pp.99-108.

[9] Hoppe, H., "View-dependent refinement of progressive meshes," in *Proc. SIGGRAPH'97*, Los Angeles, CA, Aug.3-8, 1997, pp.189-198.

[10] Hoppe, H., *Efficient Implementation of Progressive Meshes*, Tech. Report of Microsoft Research, Microsoft Corporation, MSR-TR-98-02, Jan. 1998.

[11] Hoppe, H., "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *IEEE Visualization'98*, Research Triangle Park, NC, Oct.18-23, 1998, pp.35-42.

[12] Leclerc, Y. G. and S. Q. Lau Jr., *Terra Vision: A Terrain Visualization System*, Tech. Report No.540, SRI International, California, 1994.

[13] Li, P. P., W. H. Duquette, and D. W. Curkendall, "RIVA: A versatile parallel rendering system for interactive scientific visualization," *IEEE Trans. on Visualization and Computer Graphics*, Vol.2, No.3, pp.186-201, 1996.

[14] Luebke, D. and C. Erikson, "View-dependent simplification of arbitrary polygonal environments," in *Proc. SIGGRAPH'97*, Los Angeles, Aug.3-8, 1997, pp.199-208.

[15] Xia, J. C., J. EI-Sana, and A. Varshney, "Adaptive real-time level-of-detail-based rendering for polygonal models," *IEEE Trans. on Visualization and Computer Graphics*, Vol. 3, No.2, pp.171-183, 1997.