The View-Dependent Real-Time Rendering of Large-Scale Terrain in Continuous Levels of Detail

Chin-Shyurng Fahn and Shih-Tung Wu

Department of Electrical Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan 106, Republic of China
E-mail: csfahn@mouse.ee.ntust.edu.tw

ABSTRACT

This paper presents a straightforward and efficient algorithm for the view-dependent real-time rendering of large-scale terrain in continuous levels of detail (LOD). It can render the terrain in continuous LOD without the overhead of tree structures or off-line computations of supplemental data. Such a terrain simplification algorithm consists of two main processes that we call block-based concatenation and vertex-based refinement for each visible block.

Before the block-based concatenation process, a piece of terrain had been partitioned into small units referred to as blocks. In the course of matching adjacent blocks, the LOD value for each block is first determined according to the importance of local geometry and the distance from an observer to the foreground formed by the terrain. The difference between the LOD of adjacent blocks is set to be at most one. To achieve this, a look-up table is employed to record all the cases of boundary matching. In this manner, T-junctions or cracks occurring in the boundaries of blocks can be avoided with no additional effort.

In the vertex-based refinement process, a screen-space error metric is used as a vertex refinement criterion. And the estimation of the error is on a pixel level. Prior to vertex refinement, each block is further divided into a set of sub-blocks; after that, they are manipulated from the outside to the center within a block in clockwise spiral order. During the vertex refinement, all the non-boundary vertices of the currently processed block are measured by an error function. If the error associated with a vertex is smaller than a specified pixel error, the vertex is considered to remove. Conversely, the vertex is reserved.

The experimental terrain data are originally generated by a fractal algorithm, and followed by manual editing. A primary terrain rendering system that we have developed is demonstrated on general PC platforms. In this system, the terrain simplification algorithm has been implemented for approximating and rendering digital terrain models as well as the other kinds of height fields, which consistently performs at interactive frame rates of high image quality.

Keywords: *Levels of detail, terrain rendering, geometry reduction, view-frustum culling, geomorphing.*

1. INTRODUCTION

The efficient visualization of a large-scale surface like terrain models is a challenging problem. The most common way to render such surfaces is to exploit triangulated surface models for simulation. A triangle is an essential geometric primitive in 3-D computer graphics. It is easily to be transmitted, computed, and rendered. In addition, a set of triangles can approximately compose any geometric object with arbitrary accuracy, and it can be implemented efficiently by the aid of graphics hardware. However, a large-scale triangulated surface is still hard to be rendered in real time if no accelerating techniques support, even on a powerful workstation. The major problem of most applications is that the requirements for rendering far exceed the capacities of typical graphics hardware. It motivates us to effectively reduce the complexity of the surface and display it at interactive frame rates.

In the past, several schemes have been proposed, one of which is LOD control [1-11]. The primary idea of LOD is to moderate the number of geometric primitives that need to be rendered without loss of compromising visual quality. Because LOD produces multi-resolution models, the graphics complexity can be controlled by adaptive surface triangulation together with taking advantage of different levels of detail. So far, many LOD algorithms have been presented in literatures. Recently, a new approach called *Continuous Levels of Detail* introduces a hierarchical quadtree technique [1]. In order to minify a projected pixel error, the height field is dynamically triangulated in a bottom-up fashion according to the distance from an observer to the foreground. In this way, nevertheless, the terrain is partitioned into a set of blocks and represented as regular grids, so it need create extra triangles to avoid T-junctions or cracks. Besides this, the most common drawback of such a regular grid representation is that the polygonalization is seldom optimal, or just near optimal. Also, large and flat surface mats require the same polygon density as small and rough areas do. Another famous method is based on the structure of *Progressive Meshes (PM)* [2]. This method shows that a multi-resolution hierarchy for arbitrary meshes can be defined by using a general refinement transformation, namely *vertex split*. For the same screen-space error, the PM scheme uses fewer active triangles than a quadtree scheme does. But, the former adopts a complex data structure to represent meshes.

There are many other examples of applying multi-resolution models. One of the applications is flight simulation, where complex terrain models are used. Virtual reality applications also require many complex models, depending on the degrees of importance for the

applications. CAD/CAM applications need the interactive visualization of complex models, and medical applications often create complex models from magnetic resonance imaging (MRI) scanning. In recent years, the visualization of complex molecular models has become a new research direction in chemical applications. For the aforementioned applications, LOD is not only applied to express the geometric surfaces of different complexities, but also to textures with a similar concept. Texture mapping is an important issue for a terrain rendering system, which can simulate realistic imaging without complex geometric surfaces to approximately model objects. In human vision, an object is perceived to be blurry if the distance from an observer to the object is out of the clear-sighted range. Accordingly, the texture resolution is determined by means of the distance and the angle of view. When an observer is close to objects, they need high-resolution textures. By contrast, when an observer moves further and further away from objects, they just adopt low-resolution textures.

In this paper, we will propose a view-dependent algorithm for the real-time visualization of multi-resolution terrain models. A straightforward and efficient approach is employed to accomplish continuous LOD control. This terrain simplification algorithm has been realized on general PC platforms. To achieve good performance, some criteria of the algorithm can be further modified in accordance with the requirements of practical applications.

2. SYSTEM ARCHITECTURE

Most real-time rendering algorithms for multi-resolution height fields may be logically composed of two processes. The first is vertex selection for a given frame based on some specified criteria, such as a weighted function of the importance of local geometry and the distance from an observer to the foreground, for example, the screen-space error metric. Moreover, the selection process is constrained to guarantee that the triangles should be continued, excluding from T-junctions or cracks, after which the second process collects the selected vertices that are fed to a rendering pipeline.

The terrain simplification algorithm presented here will integrate the two processes mentioned above into a single process over the visible portion of a height field. This algorithm needs not any off-line computations and special efforts to eliminate T-junctions or cracks. Although the algorithm can not generate optimal triangles, it will not influence the achievement of the rendering at interactive frame rates.

The terrain database comprises small units referred to as *blocks (or pages)*. All these blocks are in the form of square height fields of the same size, and the number of vertices in each dimension of the height field is a power of two plus one. Also, some attributes are attached to the block, such as a set of texture maps (e.g., different resolution texture maps), texture coordinates, and bounding box vectors.

There is a clear tradeoff on determining the size of a block. Larger blocks allow a greater reduction in the number of triangles to be drawn; conversely, smaller blocks permit more efficient rendering of those constituted triangles. In our implementation, the length or width of each block possesses 33 vertices separated equally. This amount is suitable for a general PC platform, even without the aid of a 3-D graphics accelerator. The disadvantage of dividing the terrain into blocks occurs in the geometry reduction that no quad may be larger than a block. The system architecture of our terrain simplification algorithm for view-dependent rendering in real time is shown in Fig. 1.

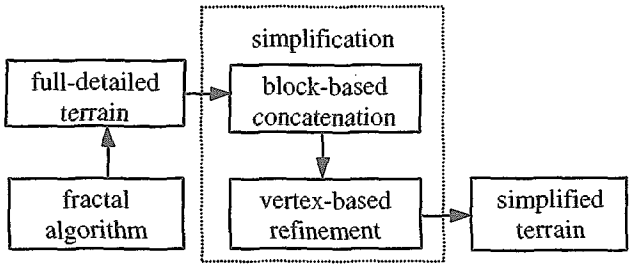


Fig.1. The system architecture of our terrain simplification algorithm.

3. OUR TERRAIN SIMPLIFICATION ALGORITHM

In this section, we describe our terrain simplification algorithm, including a block-based concatenation process and a vertex-based refinement process for each visible block.

3.1 The Block-Based Concatenation

The levels of detail selected at a given vertex are a function of the relative position of the vertex and an observer's eyes. In our experiments, each block consists of 33x33 vertices, whose LOD is classified into 5 levels. How many levels of detail required are dependent on the block size. The only one thing needed to check is that the difference between the levels of two adjacent blocks is at most one. The block-based concatenation process is executed in row-major order, i.e., in the horizontal (major) direction, followed by the vertical (minor) direction, as illustrated in Fig. 2.

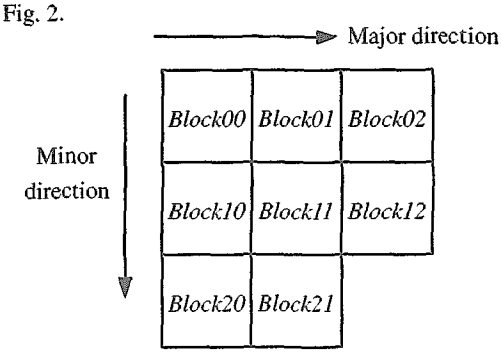


Fig.2. The processing directions of the block-based concatenation.

When the block-based concatenation process proceeds in the major or minor direction, it is prohibited from revisiting the previously processed blocks. According to this regulation, we can unify the two steps: selecting vertices and feeding them to the rendering pipeline into one step. Therefore, it is merely required to scan all visible blocks one time.

During the block-based concatenation process, we simply want to check the boundaries of the adjacent blocks in both the major and minor directions. In Fig. 2, for instance, *Block00* is adjacent to *Block01* and *Block10*, so *Block00* just requires for matching its LOD with those of *Block01* and *Block10*. In other words, each block only needs to check its right and bottom boundaries. Figure 3 shows one case of boundary matching, where the LOD of *Block00* is higher than that of *Block01* by one but equal to that of *Block10*.

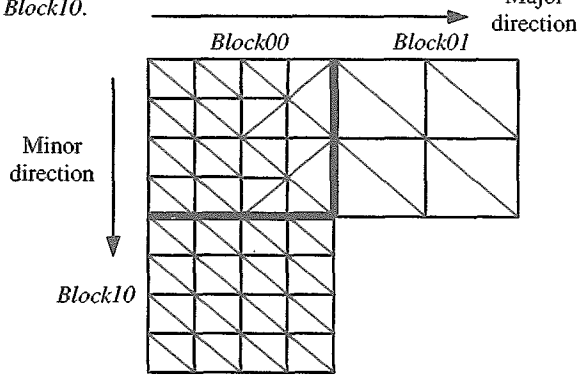


Fig.3. An example of the boundary matching between a currently processed block and its adjacent blocks.

We totally categorize nine cases of boundary matching between different LOD of a currently processed block and its eastern and southern adjacent blocks. The following lists the nine cases:

1) The eastern block has the same LOD and the southern block has the same LOD.
2) The eastern block has a higher LOD and the southern block has the same LOD.
3) The eastern block has a lower LOD and the southern block has the same LOD.
4) The eastern block has the same LOD and the southern block has a higher LOD.
5) The eastern block has the same LOD and the southern block has a lower LOD.
6) The eastern block has a higher LOD and the southern block has a higher LOD.
7) The eastern block has a higher LOD and the southern block has a lower LOD.
8) The eastern block has a lower LOD and the southern block has a higher LOD.
9) The eastern block has a lower LOD and the southern block has a lower LOD.

Figures 4 (a)~(i) illustrate how to match the boundaries of adjacent blocks as depicted in the above nine cases. In case 7 and case 8, however, we can not directly handle them, as shown in Figs. 4 (g)&(h), because their boundaries occur in cracks. In these situations, we further adjust the LOD of

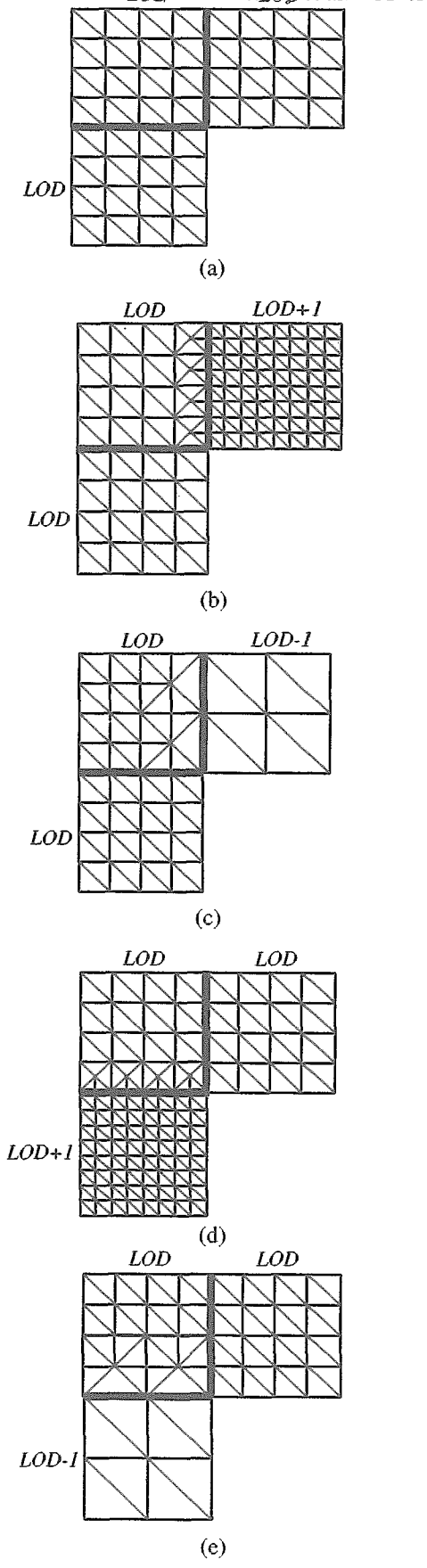
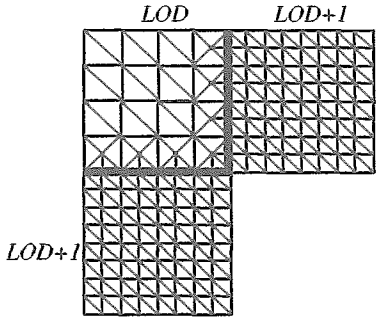
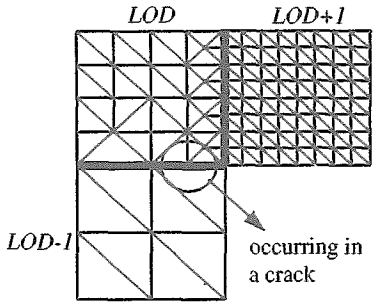


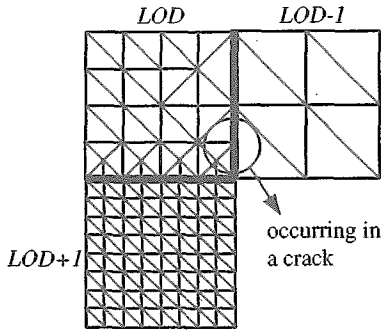
Fig.4. The nine cases of boundary matching between



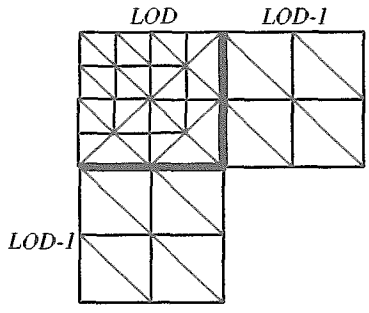
(f)



(g)



(h)



(i)

Fig.4. (Continued)

the eastern adjacent block to coincide with that of the currently processed block. Note that the difference of two adjacent block's LOD is at most one for each of the nine cases once the boundary matching is completed. Figure 5 shows the sequential block-based processing for the visible blocks obtained from a view frustum culling operation.

3.2 The Vertex-Based Refinement

After the block-based concatenation process, each block is still associated with a representative LOD except parts of its boundaries. Here, we will consider the vertex-based refinement for individual blocks. In the current implementation, each block has 33x33 vertices. Because the block is not too large, the vertex-based refinement is unnoticeable to improve the rendering performance. But, if we adopt larger blocks, the vertex-based refinement becomes important. Our algorithm processes a block only one time, which can not go back to refine the block repeatedly. In addition, we do not perform the refinement of the vertices on boundaries.

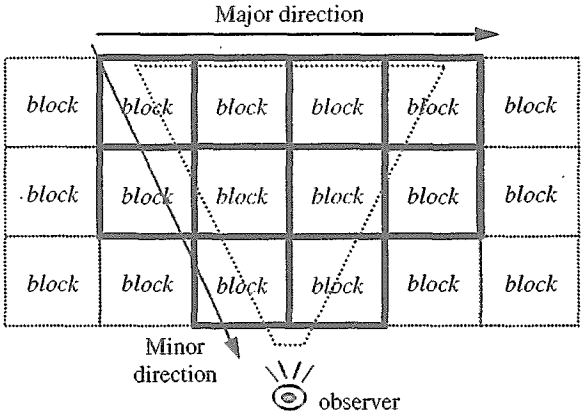


Fig.5. The sequential block-based processing of visible blocks.

There are three types of an attribute to record the status of a vertex: *lock*, *enable*, and *disable*. When the vertex-based refinement is carried out, the attributes of the vertices on the boundary are set as lock. This inhibits such vertices from refinement to preserve the continuity of the LOD of adjacent blocks. If a vertex is considered for removal, then the attribute of the vertex is *disable*. On the contrary, the attribute of the vertex is *enable*.

In the course of block-based concatenation, we only feed the boundary vertices into the rendering engine. In the following process, we will consider non-boundary vertices for refinement. A sub-block is defined as the fundamental unit used for the vertex-based refinement. In our implementation, each block is further divided into 4x4 sub-blocks. That is, each sub-block comprises 9x9 vertices. Figure 6 shows the sub-block consisting of eight triangle fans.

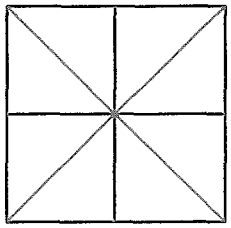


Fig.6. The geometry contents of a sub-block.

To guarantee the boundaries of adjacent sub-blocks matched well with each other, we process the vertex-based refinement for the sub-blocks within a block in clockwise spiral order from the outside to the center as shown in Fig. 7.

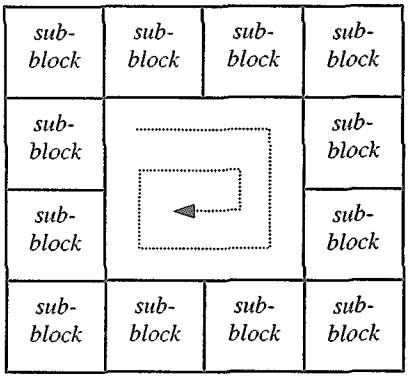


Fig.7. The refinement sequence of the sub-blocks within a block.

In principle, if a sub-block can be reduced to the coarsest block, i.e., composed of only two triangle fans, then we will replace this sub-block and its eastern, southern, and southeastern adjacent sub-blocks with a larger sub-block that contains eight triangle fans. Figure 8 illustrates the general case of such a geometry reduction process. This replacement strategy for the sub-block located in the rightmost column or the lowermost row should be modified according to its existing adjacent sub-blocks within the same block. The screen-space error metric described in [1] is adopted, which is derived from projecting the difference of the height of a vertex and the mean height of its adjacent vertices. If the resulting pixel error is smaller than a user-defined value, then the vertex will be considered for removal. Otherwise, the vertex is hold. Consequently, the LOD of a block turns to be non-uniform.

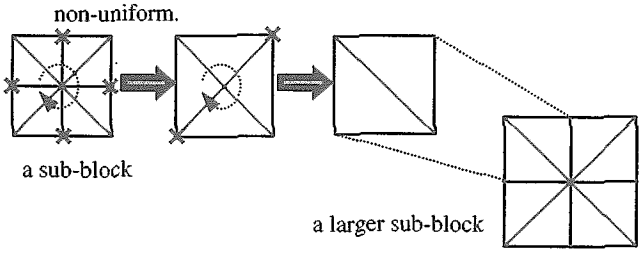


Fig.8. The geometry reduction of sub-blocks in a general case (the cross indicates the removed vertex).

4. OTHER ACCELERATING TECHNIQUES

In this section, we give some techniques for facilitating our terrain rendering system, including texture mapping, texture LOD, view frustum culling, terrain modeling tools, and geomorphing.

4.1 Texture mapping

Texture mapping is a technique of adding detail to a rendering image without detailed modeling. Texture mapping can be thought of pasting a picture to the surface of an object. The use of texture mapping needs two pieces of information: a texture map and texture coordinates. The texture map is the picture to be pasted, and the texture coordinates specify the location where the picture is pasted. Most texture maps and coordinates are 2-D data, but 3-D texture maps and coordinates are becoming more popular.

In our experiments, we employ 96 texture maps that demand the memory of about 5 megabytes. Each texture map is possessed of three different resolutions used for texture LOD. We will discuss this in the next subsection. In the current implementation, tile-based texture maps are adopted to approximate real terrain images.

4.2 Texture LOD

In our terrain rendering system, we partition a piece of terrain into a set of blocks, and we perform the same operation on the textures used. Each block is pasted by a sequence of texture maps with various resolutions. In our implementation, the highest resolution of a texture map is 128×128, and the lowest one is 32×32. All of them are applied to the block that comprises 33×33 vertices. And all the texture coordinates are generated in real time by a linear mapping technique. Like the height field data, the image data on the boundaries of adjacent texture maps should be identical. That is, the last texel in a given row of the current page has to equal the first one in the same row of the next page. The texture resolution will be determined on the fly by the distance from an observer to the associated block in the foreground. When the distance increases, the resolution must decrease. In the experiments, we merely exploit three different resolutions for each texture map.

4.3 View Frustum Culling

The basic idea of view frustum culling is that before drawing the entire scene for the current frame, we can decide whether an object is inside the field-of-view (FOV) of an observer. If the objects completely or partially lie outside the view frustum, we can remove or cull them to speed up the rendering. Nevertheless, it is quite inefficient if we accomplish this for all the triangles constituting the foreground formed by the terrain individually. Therefore, we had better divide a piece of terrain into smaller blocks for the facilitation of culling.

After the terrain is represented by a set of blocks, the bounding box can be computed for each block. One advantage of using blocks is to achieve rapid triangle culling. To avoid excessive time spent in rendering the triangles outside the FOV, we intersect the terrain with a view frustum and render only those blocks that belong to this intersection. The view frustum culling is recursively operated if the bounding boxes are built into a hierarchical organization. This will be more efficient than a non-hierarchical organization. For the hierarchical organization, the culling is performed in a recursive pre-order manner. If a block is not within or across the FOV (this can be determined by intersecting the bounding box of the block with a view frustum), none of its children is visible. If the block is visible, its children are recursively checked against the view frustum. Thus, for the invisible blocks on the top levels of the hierarchical organization,

we can cut out the corresponding large terrain that need not be rendered.

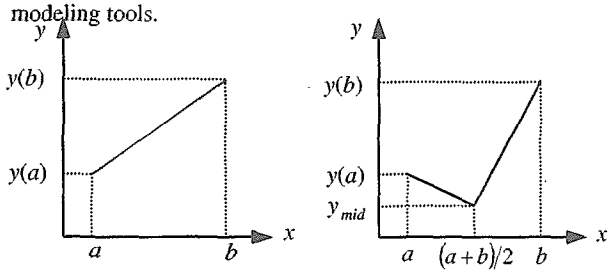
The view frustum culling on blocks is a very economical way to reduce the number of triangles considered for rendering. For an FOV of 90 degrees, the culling operation moderates the number of triangles to be rendered by more than half. In our implementation, each block has 33x33 vertices and about 600~700 blocks are visible in each frame. For most of invisible blocks, they can be discarded with little effort when the terrain simplification algorithm is in progress.

4.4 Terrain Modeling Tools

Terrain modeling is an essential task for flight simulation. We apply our terrain modeling tools to create height field data. Through the tools, we can specify elevation data by fractals or by hand, and specify texture maps manually. What follows introduces how to generate fractal terrain. The key concept behind any fractals is self-similarity. An object is said to be self-similar when magnified subsets of the object look like the whole and resemble each other. Terrain falls into the category of such self-similarity. A diamond-square algorithm is employed to generate fractal terrain, which is formerly in a two-dimensional version. Here, a one-dimensional midpoint-displacement algorithm is used to explain how the diamond-square algorithm generates the terrain. The midpoint-displacement algorithm starts with a straight-line segment, and calculates a displaced y value for the mid-position of the line segment by taking an average of the y values of endpoints a and b, followed by adding a random offset r:

$$y_{mid} = [y(a) + y(b)]/2 + r.$$

To approximate fractional Brownian motion, we generate the value r from a Gaussian random variable with a mean of zero and a variance proportional to $|b\text{-}a|^{2H}$, where $H=2\text{-}D$ and $D>1$ is the fractal dimension. Figure 9 shows a fractal curve obtained with this method. And then we can extend this midpoint-displacement algorithm to create two-dimensional elevation data. Figure 10 exemplifies the fractal terrain shown in a window opened by our terrain modeling tools.



(a) (b)

Fig. 9. The generation of a fractal curve: (a) an original straight-line segment; (b) a random midpoint-displacement of (a).

4.5 Geomorphing

There are two principal factors for a good visual flight simulation: a high frame rate and non-poping artifacts. The LOD techniques can reduce the number of triangles for rendering and then increase the performance to achieve

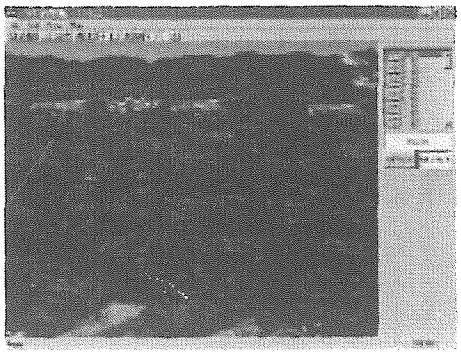


Fig.10. An example of the fractal terrain shown by our terrain modeling tools.

high frame rates; however, they also result in poping artifacts. Such a drawback can be overcome if the tolerance of the screen-space error is kept near a value of 1 pixel. In our current implementation, the tolerance of the screen-space error is required to exceed 2.5 pixels more or less for reaching interactive frame rates. To eliminate poping artifacts, we incorporate morphing techniques into our terrain rendering system. In [2], the term *geomorph* was previously to denote the morphing between different LOD of the same terrain segments. The LOD morphing can be stated as two steps. First, geometric primitives are grouped into new interpolated vertices, edges, and faces for remodeling an object with no change of approximation accuracy. Not only a vertex itself but also all its associated attributes should be interpolated, such as normals and texture coordinates. Second, each vertex is

$$f(v', v, d) = (1.0 - d)v' + dv,$$

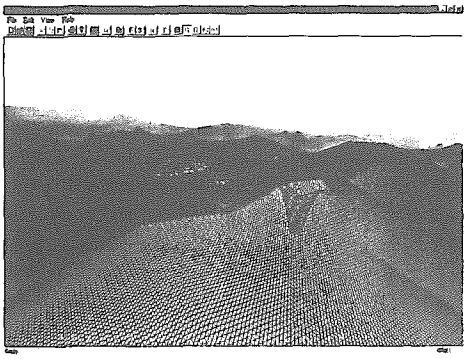
continuously moved from its interpolated position v' to the true location v using a morphing function as follows: where $d \in [0,1]$ is the normalized distance from an observer to the vertex. Additionally, some different approaches that adopt time functions are exploited to accomplish the geomorphing [11]. Nevertheless, they are not appropriate for our system architecture.

5. EXPERIMENTAL RESULTS

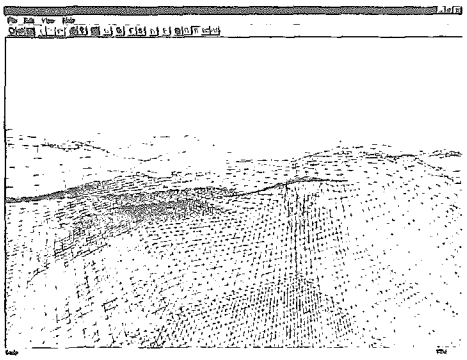
In this section, we demonstrate the experimental results from our proposed view-dependent terrain simplification algorithm. All the testing programs have been implemented in C++ language by using the Microsoft Visual C++ 5.0 compiler under the Microsoft Windows 98 operating system. The platform is a personal computer equipped with an Intel Pentium II 350 CPU and an ASUS AGP-V3800 graphics board with an OpenGL accelerator.

Two of the experimental results are illustrated in Figs.11&12, respectively. Figure 11(a) shows the original terrain I with texture mapping, while Fig. 11(b) with wireframes. And Figs. 11(c) & 11(d) individually show the simplified terrain I through the block-based concatenation and vertex-based refinement processes. For terrain II, the similar results are shown in Figs. 12(a)~(d). Table 1 records the numbers of triangles of each original terrain

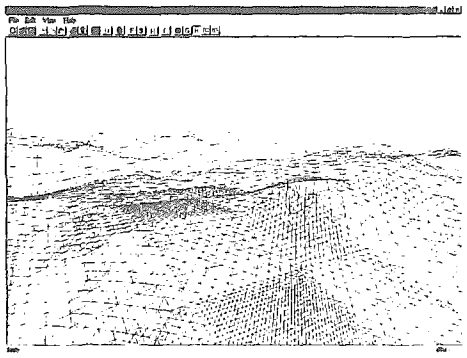
(a)



(b)

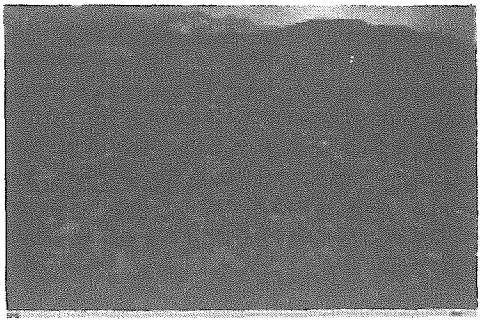


(c)

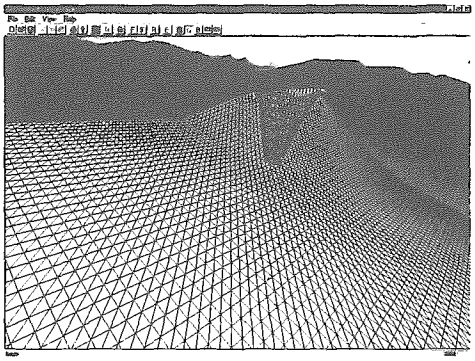


(d)

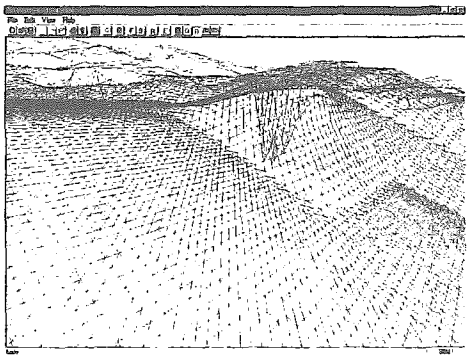
Fig.11. The illustration of terrain I rendered in: (a) full detail with texture mapping; (b) full detail with wireframes; (c) block-based simplification with wireframes; (d) vertex-based simplification with wireframes.



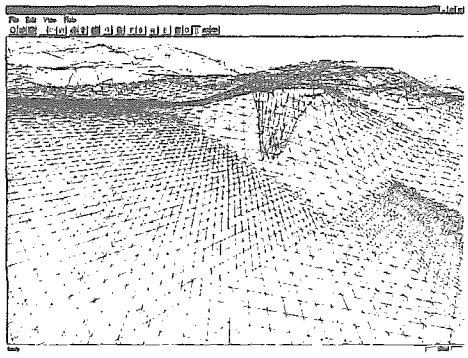
(a)



(b)



(c)



(d)

Fig.12. The illustration of terrain II rendered in: (a) full detail with texture mapping; (b) full detail with wireframes; (c) block-based simplification with wireframes; (d) vertex-based simplification with

wireframes.

terrain simplification algorithm with the angle of view 60°. On an average, the data simplification ratios attain 91 and 223 for the first and second processes, respectively. Table 2 reveals the number of visible blocks and the average frame rate obtained from rendering the terrain in continuous LOD with texture mapping. This performance fulfills the requirements for the view-dependent real-time rendering of large-scale terrain.

TABLE 1. The numbers of triangles of the original terrain and its simplified ones resulting from the block-based concatenation and vertex-based refinement.

Terrain no. \ No. of triangles	Full detail	Block-based simplification	Vertex-based simplification
I	1,177,600	11,899	4,957
II	1,224,704	14,975	5,918

TABLE2. The number of visible blocks and the average frame rates of the terrain rendered in continuous LOD with texture mapping.

Terrain no.	No. of visible blocks	Average frame rate (sec^{-1})
I	542	25.0
II	519	23.2

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a straightforward approach to achieving the view-dependent real-time rendering of large-scale terrain in continuous LOD. The advantage of our terrain simplification algorithm needs not any off-line computations. But, the disadvantages of this algorithm are that the terrain simplification is not optimal and the poping artifacts can not be completely eliminated yet. At present, we merely consider the distance from an observer to the foreground and where the viewpoint is. Other refinement criteria may be taken into account, as mentioned in the view-dependent algorithm [11].

In our current implementation, the poping effect still exists. On the other hand, our geomorphing function is not the best one, because it just has several levels, depending on the distance. We can further adopt a time function to exclude the poping artifacts, not only referring to the distance. Also, our refinement method has a room for improvement due to the inherence of the regular grid representation. In this representation, extra triangles are required to prevent from the occurrence of T-junctions or cracks. In the near future, we will try to find an optimal refinement algorithm.

So far, we solely render a piece of terrain loaded into the memory. It is very waste and may decrease the rendering performance if the terrain is extremely large. Dynamic loading techniques can provide the efficient use of memory, and browse a larger piece of terrain with the aid of registration. This is adequate for the machine that only has a finite size of memory. As a result, the terrain data set can be extended unlimitedly, if necessary.

REFERENCES

[1] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner, "Real-time, continuous level of detail rendering of height fields," in *Proc. of the ACM SIGGRPAH Conf. on Computer Graphics '96*, New Orleans, LA, 1996, pp.109-118.

[2] H. Hoppe, "Progressive meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics '96*, New Orleans, LA, 1996, pp.99-108.

[3] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," *Computer Graphics*, Vol.26, No.2, 1992, pp.65-70.

[4] G. Turk, "Re-tiling polygonal surfaces," *Computer Graphics*, Vol.26, No.2, 1992, pp.55-64.

[5] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright, "Simplification envelopes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics '96*, New Orleans, LA, 1996, pp.119-128.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Porc. of the ACM SIGGRAPH Conf. on Computer Graphics '93*, Anaheim, CA, 1993, pp.19-26.

[7] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics '97*, Los Angeles, CA, 1997, pp.209-216.

[8] M. Duchaineau, M. C. Miller, M. Wolinsky, C. Aldrich, D. E. Sigeti, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes," in *Proc. of the IEEE Visualization Conf. '97*, Phoenix, AZ, 1997, pp.81-88.

[9] R. J. Fowler and J. J. Little, "Automatic extraction of irregular network digital terrain models," *Computer Graphics*, Vol.13, No.2, 1979, pp.199-207.

[10] H. Hoppe, "View-dependent refinement of progressive meshes," in *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics '97*, Los Angeles, CA, 1997, pp.189-198.

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