View-dependent Real-time Terrain Rendering Using Static LOD‡

Ye Yuting, Wang Guoping

Department of Computer Science and Technology, Peking University, Beijing, 100871, P. R. China
{yyt, wgp}@graphics.pku.edu.cn

Abstract Though dynamic and continuous LOD has been widely accepted so far, the more and more powerful 3d graphics hardware brings about new concepts in fast terrain rendering. The time spent on deliberately calculating the perfect set of triangles may overwhelm the cost of leaving them rendering. De Boer presented the GeoMipMap algorithm as a static LOD that wins over ROAM, a typical and prevailing dynamic LOD algorithm, in performance. This article aims at further polishing the static algorithm to overcome its rigid nature, and generates as good image quality as, if not better than the dynamic algorithm. Firstly, intensive comparisons between GeoMipMap and ROAM are made to reveal their underlying differences. Then refinements of detail level decision, tile transition and vertex morphing are presented to provide flexible and smooth results that make such static approach satisfiable enough to use.

Keywords GeoMipMap; ROAM; real-time performance; 3d graphics hardware

1. INTRODUCTION

Among the current various algorithms to render massive terrain in real time, Dynamic (and continuous) Level of Detail (LOD) approaches, especially the Real-time Optimally Adapting Meshes (ROAM)[2], may be the most prevailing. The underlying concept is to minimize the number of triangles while introducing no perceptible visual error. As de Boer[1] mentioned, it works well by reducing the workload of graphics hardware but transferring the burden to CPU. Since today's graphics hardware is capable to process and render a large amount of triangles per frame, a more up-to-date approach may be to exploit the 3D graphics hardware and free the CPU. Unlike the dynamic LOD, GeoMipMap presented by de Boer is a static algorithm relying on a series of fixed models. Instead of adjusting the existing meshes in real time, GeoMipMap picks up certain models in no time. Though it cannot deliver a "prefect" set of triangles to the rendering pipeline, GeoMipMap gains better performance over ROAM by the least amount of CPU overhead. Drawbacks are that it cannot guarantee either frame-to-frame coherence, or smooth transition between adjacent tiles of different levels because of the somewhat rigid models. This article presents refinements to de Boer's algorithm. We would use a more flexible equation to pick up detail levels, bringing fewer triangles without perceptible errors and also a more constant frame rate. An alternative mesh to connect different detail levels is presented for nicer quality. And the use of vertex morphing makes this discrete approach seem continuous.

2. GEOMIPMAP VS ROAM

This section will first briefly review the two algorithms. Then comparisons will be made on the algorithms, memory requirements, visual fidelity and real-time performance.

2.1 Review

GeoMipMap can be understood as texture mipmap technique in geometry, which uses tiles of

‡This work is supported by Natural Science Fund of China (60173062), and by Natural Science Fund of Beijing (4012008), by National 863 project (2001AA115126)

different resolutions to match the projection terrain. First, it defines levels by skipping every other line in a previous level of the height field. Then it pre-computes the actual errors caused by these rough models. They will be used later as the error metric. Finally it chooses certain level to render each tile, according to the real-time viewpoint and the screen pixel error. The essence is to utilize graphics hardware by delivering as many triangles as possible, and to use triangle strip or triangle fans to speed up rendering. Calculations are minimized to a few multiples and additions each frame.

ROAM seems a bit different. Its meshes are organized in a binary tree hierarchy. A triangle is split by the longest edge to reach a rougher level, while two triangles sharing the same longest edge (called a “diamond”[2]) merge into a finer level. Actual space errors of each triangle in each level are pre-calculated. When viewpoint changes in runtime, these errors are converted to screen pixel error as priorities. Two priority queues dynamically “split” or “merge” triangles that are not “optimal”. An effective way to maintain constant frame rate is to leave the small priority -- the relatively more acute triangles -- unprocessed if time slice is expired. Triangles transit smoothly by “force splitting” the rougher one. Considering the frame-to-frame coherence, only a small amount of triangles should be adjusted each frame. It yields so satisfiable results both in visual fidelity and real-time performance that it has been widely used so far.

2.2 Algorithm Evaluation

[3] presented 5 criteria to evaluate a real-time LOD algorithm for height fields as follows:

(i) At any instant, the mesh geometry and the components that describe it should be directly and efficiently queryable, allowing for surface following and fast spatial indexing of both polygons and vertices.

(ii) Dynamic changes to the geometry of the mesh, leading to recomputation of surface parameters or geometry, should not significantly impact the performance of the system.

(iii) High frequency data such as localized convexities and concavities, and/or local changes to the geometry, should not have a widespread global effect on the complexity of the model.

(iv) Small changes to the view parameters (e.g. viewpoint, view direction, field of view) should lead only to small changes in complexity in order to minimize uncertainties in prediction and allow maintenance of (near) constant frame rates.

(v) The algorithm should provide a means of bounding the loss in image quality incurred by the approximated geometry of the mesh. That is, there should exist a consistent and direct relationship between the input parameters to the LOD algorithm and the resulting image quality.

We can see that in general both algorithms fit these criteria quite well despite their own pros and cons. The mesh is static in GeoMipMap and is managed by triangle lists (two priority queues) in ROAM. Vertices in GeoMipMap can be directly queried as indices of the height field, but those of ROAM can only be reached by going through the binary tree. Considering the number of vertices, cost is larger in ROAM though trees of the tiles can be shorter. (ii) and (iii) can be achieved by tiling. Changes in geometry or localized high frequency data will affect only the tiles containing them, so recomputation and model complexity can be kept relevantly small. Likewise, (iv) and (v) both rely on the error metric. Both algorithms use screen pixel error as simplification criterion to guarantee visual fidelity. But ROAM can maintain a more constant frame rate than GeoMipMap. It can leave the “near optimal” triangles (those of small priorities) as they are when time slice is expired, at the cost of a slightly larger error. On the other hand, GeoMipMap maintains the error boundary strictly by rendering more details. More triangles than necessary is produced to describe the rest flat area if small bumps occur. So frame rate depends partly on the mesh geometry. Anyway, since viewpoint seldom changes dramatically, geometry will usually have

frame-to-frame coherence.

2.3 Memory

Though the capacity of RAM is fleetly increasing, it can never satisfy human desires. Data is swelling even more dramatically. GeoMipMap shows great advantage by requiring few extra memories. Only the maximum space errors of each level for each tile need to be stored in runtime. Given total M levels and N tiles of the terrain, M*N numbers is needed.

ROAM maintains a list of current triangles and a list of "mergeable" diamonds, as well as the space errors. Notice that these space errors are of every triangle in every level. Given total M levels, the increase in one level will double the number of triangles. N such tiles yield $(\Sigma 2^{m-1})*N$ numbers, where m begins from 2 to M. The maximum length of two queues is hard to estimate thought, it is sure that tens of thousands triangles will be rendered thus stored each frame.

Nonetheless, if raw data are in the form of integers or even bytes, the total requirement would not be large in common cases. Tiling technique may allow dynamic loading of data to further reduce runtime memory. Still, GeoMipMap has a more promising future in extraordinary massive data sets for its few memory requirements.

2.4 Quality

Although visual fidelity is bounded by screen pixel error threshold, the popping effects and T-junctions (gaps between different levels) can also affect visual impression.

ROAM is a polygon-based algorithm which can provide more flexible meshes. Its meshes have the merit of isoceles right-angle triangle in two dimensions. To guarantee smooth transition between levels, it forces the rougher triangles to split recursively, until no T-junctions exist[Fig1].

Such process generates smooth meshes but also many unnecessary triangles, which may possibly have great impact on real-time performance. Also, the adaptive process may cause problem in some cases. As mentioned in 2.2, smooth change in view parameter usually leads only to small changes in geometry complexity. ROAM

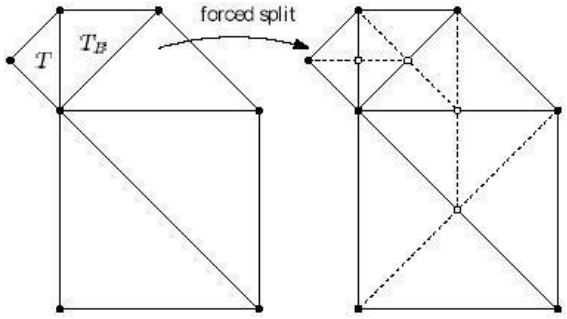

Fig1 Force split of triangle T brings 115% more triangles. Excerpt from [2]

can gracefully adjust the meshes by a few steps of split and merge. But at odd times, say when a change of view direction by turning around leads to great change of the scene, a large amount of slip and merge are required, resulting in long processing time, or large error if constant frame rate is wanted[Fig2].

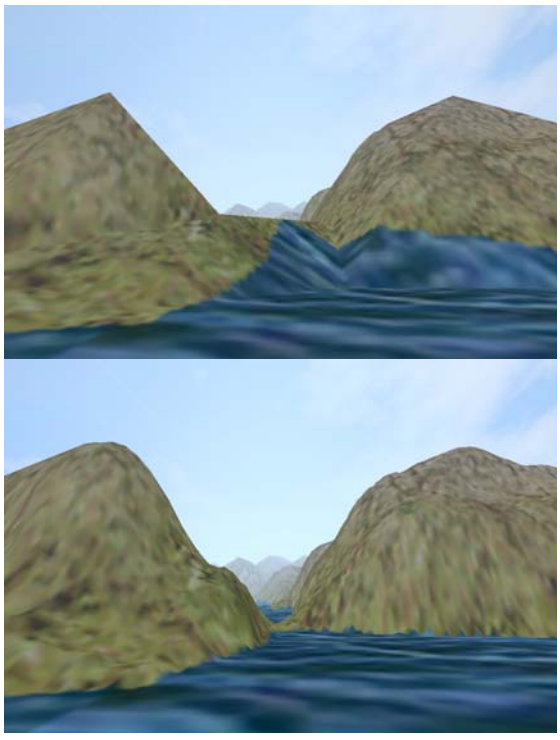

Fig2 The above was the first frame after turning around. Below has been adaptive after several frames. (ROAM implementation of Tuner[5])

Basically, GeoMipMap also divides meshes into isoceles right-angle triangles. Since it is a tile-based algorithm, gaps will appear if adjacent tiles are of different levels. De Boer[1] solved it by "changing connectivity (or indexing) of vertices for the higher

detail GeoMipMap."[Fig3] The solution is tricky in that it "does not alter the GeoMipMap level's vertex layout" and brings no extra cost. But when the differences between tiles are great, the connective triangles would degenerate into long thin lines, which ruin the mesh quality. Another problem is that it doesn't work when all four neighbors need to be connected. We have to render in a certain order to prevent this odds. A better solution will be presented in 3.2. Nonetheless, GeoMipMap do have merits in controlling screen pixel error. It is tolerant to great changes between frames. Calculations of each frame are fixed and unaffected by geometry. It can maintain high quality images anytime. No "adaptive" process is perceived.

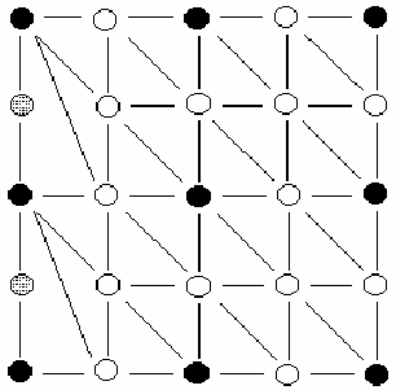

Fig3 Solution presented by De Boer. The grey points are skipped. Expert from [1]

Popping would be another matter. It is thought to be caused by sudden switching of detail levels. In fact it is affected by many factors. ROAM is a continuous LOD that continuously switches between detail levels. But it also pops when a triangle is split or merged. The choice of error metric, mesh geometry and mesh quality together make up popping. As a matter of fact, continuous LOD cannot win over discrete LOD in reducing popping, but vertices morphing can solve the problem in both cases. Vertex morphing of GeoMipMap would be presented in 3.3.

2.5 performance

The reason why de Boer presented GeoMipMap is that he saw great promise of it to speed up rendering. He is not the only person to do so. Marechal[4] did an experiment to illustrate the power of hardware rendering with low CPU overhead. He rendered a huge map using both ROAM (implementation of Bryan Turner[5]) and so called brute force approach, say simply draws a quad between any 4 adjacent points on the height map. Result shows in Fig4 that brute force approach can outweigh ROAM if less detail is acceptable. It's no doubt that GeoMipMap would be far better than the brute force approach.

Real-time calculations of GeoMipMap only include distances between viewpoint and the tiles, and a few comparisons to pick up the appropriate levels. Most time is spent in rendering triangles. Moreover, the static models can utilize triangle strip and triangle fans that further speed up rendering. As ROAM is elaborately calculating the perfect set of triangles, it has to call APIs as split and merge many times, as well as sorting to maintain its priority queues. The space errors of each Triangle also need to be computed in real-time. Optimal approach as triangle strip or display list cannot be utilized in dynamic algorithm.

The important fact is that the extra CPU overhead used to reduce triangles may be larger than leave them to the graphics hardware, especially when precompile technique is used. A perfect balance between CPU workload and that of graphics hardware may yield the best performance.

3. IMPROVEMENTS

As stated before, the static approach will suffer from its rigid nature in quality and in delivering too much triangles. In this section, we would like to present some refinements to GeoMipMap in order to improve both quality and performance.

3.1 Level Decision

De Boer's approach to speed up level decision is to pre-calculate the maximum space error of each level, then convert them to distance using projection parameters and the screen pixel error. Distances from viewpoint to each tile are computed in runtime, comparing with the precomputed distances to choose the appropriate levels. [Equ1] is used to precompute the distances.

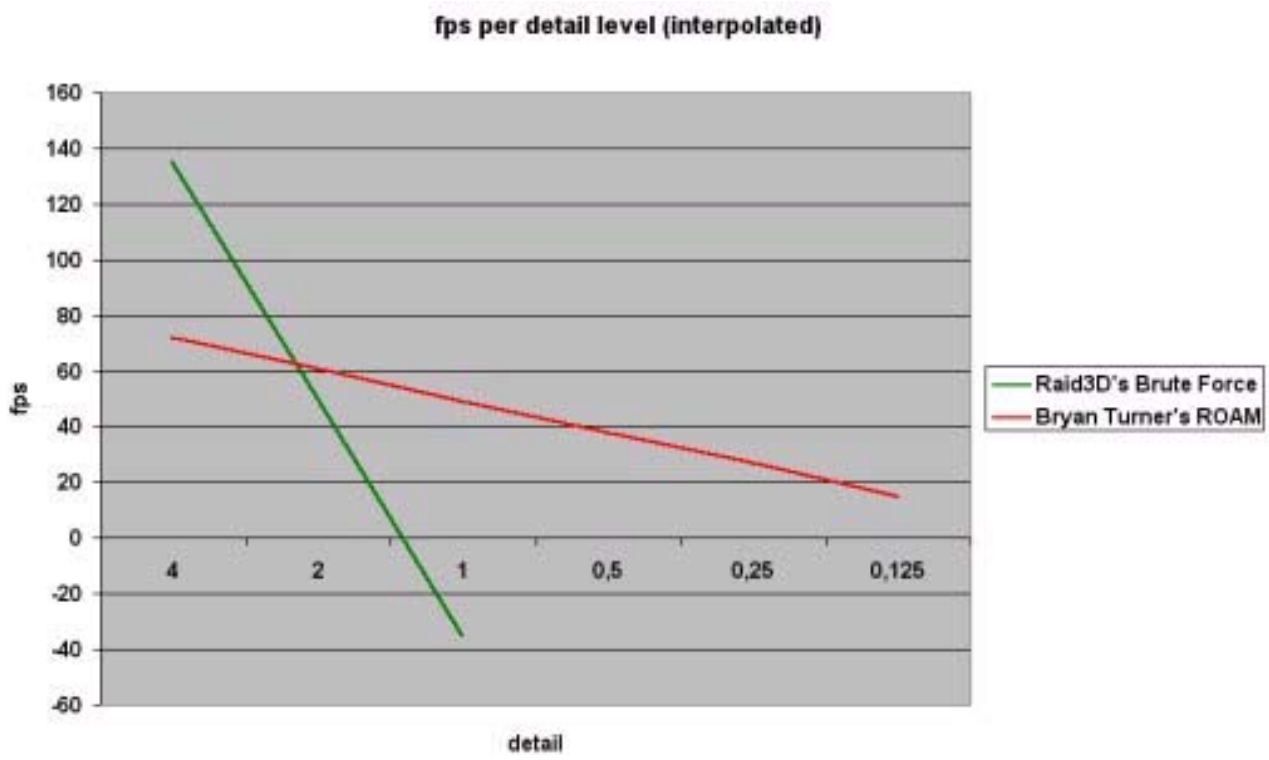

Fig4 Tests are run on Pentium III 600 Mhz, 128 Mb RAM, nVidia TNT2 32 Mb Video, 800x600x32 resolution

$$D_m = |\delta_m| \cdot \frac{n \cdot v}{|t| \cdot 2\tau} \qquad \text{[Equa1]}$$

Where n is the near clipping plan, t is its top coordinate as in (l, r, b, t, n, f), v is the vertical screen resolution in pixels and τ is the screen pixel error threshold. D_m and δ_m represent the maximum distance to use level m and the space error cased by this level, respectively. The equation is based on the simplest projection model[Fig5]. [Equa1b] can be deduced from [Equa1a], and [Equa1b] and [Equa1c] together deduce [Equa1].

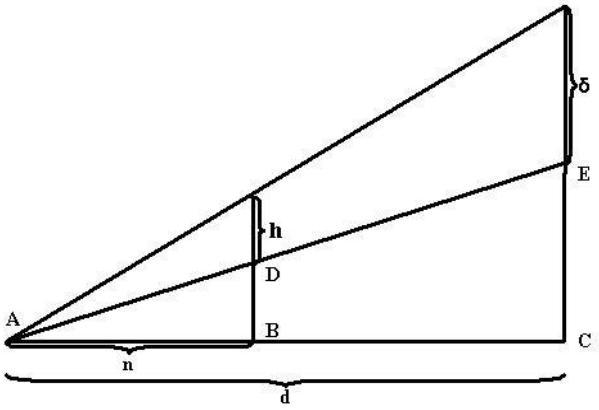

Fig5 A simplified projection model in 2 dimensions

$$\frac{\delta}{h} = \frac{AE}{AD} = \frac{d}{n} \qquad \text{[Equa1a]}$$

$$d = \frac{\delta}{h} \cdot n \qquad \text{[Equa1b]}$$

$$\tau = \frac{v}{2|t|} \cdot h \qquad \text{[Equa1c]}$$

In the equation, camera direction is assumed to be horizontal (parallel to x/z plane) because it brings the highest projection error. Also yaw angle is ignored (distance from viewpoint to the near clip plane is assumed to be n constantly) because eyes are more sensitive to central than peripheral. A greater peripheral error can be tolerated to speed up the calculation. To avoid the square root instruction, [Equa2] is actually used, where C is considered a constant in run time.

$$D_m^2 = \delta_m^2 \cdot C^2 \qquad \text{[Equa2]}$$

$$C = \frac{n \cdot v}{|t| \cdot 2\tau}$$

There are drawbacks in it. (i) Parameters n and t cannot change. It forbids zooming of the camera.

Detail level cannot change in the same way since τ is a constant. (ii) When projection is simplified in 2-dimension, distances should not be computed in 3-dimension but just of the x/z plane. (iii) Though [Equa2] is correct as an equation, it exaggerates the impact of δ to D_m because square is used when comparison is done linearly.

To avoid these downsides, we would like to revise it a bit as

$$\delta = \left(\frac{2|t| \cdot \tau}{n \cdot v} \right)^2 \cdot d^2 \qquad \text{[Equa3]}$$

Parameters have the similar meaning as in Equa1. d is the distance between current viewpoint to the tile center in x/z plane. δ is the square of space error caused by d for a given τ. But it still compares to the precalculated δ_m to pick up a right level. In this way, decision of level depends more on distance than on geometry feature. Thus a more constant frame rate can be maintained by diminishing the impact of changes in geometry. Notice that the actual screen pixel error is no longer bounded to τ, but various dynamically.

$$\delta = \left(\frac{2|t| \cdot \tau}{n \cdot v} \right)^2 \cdot d^2 = \frac{2|t| \cdot \tau'}{n \cdot v} \cdot d$$

$$\tau' = \frac{2|t| \cdot \tau}{n \cdot v} \cdot d \cdot \tau = \sqrt{\delta} \cdot \tau$$

That's to say, the actual screen pixel error is scaled by the square root of space error. A more precise level will be used for a smaller d, bringing smaller δ. The farther the distance, the larger δ, thus the larger the actual screen pixel error at that point. It's intuitive that we pay less attention to distant object than to the near. Therefore larger error in distance can be tolerated, when the near scene can be approximately bounded by τ. It reduces even more triangles and becomes faster than the previous equation. Moreover, n, t and τ can change in real time, allowing customized detail level and zooming of the camera. Nonetheless, constant C in Equa2 can still be used by being calculated once every frame. We don't bring in more calculations but gain all the merits mentioned.

3.2 Transition Between Tiles

The approach in GeoMipMap[Fig3] is satisfiable in most cases when the adjacent tiles differ not much. It is valuable in that no change of vertices or extra cost is introduced. The body of a tile is triangulated in triangle strips, so the connective edges should not ruin this structure, and they would be better to have the similar structure. But in some cases when two adjacent tiles differ much — it's not impossible for both distance and geometry determine the level, thin triangles produce. Though de Boer claimed that "This process must be performed for each of the four edges which connect the GeoMipMap to a lower detail"[1], his solution fails to do so.

[6] presented another solution that produce better meshes[Fig6]. But it doesn't utilize triangle strip, thereby sacrificing efficiency. [7] shows whether it makes a difference.

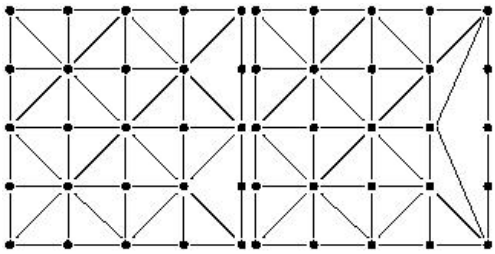

Fig6 An alternative approach generates better triangles but fails to utilize triangle strip. Excerpt from [6]

Our solution refers to the above approach and integrates them together[Fig7]. Only tiles of lower resolution need to "connect" to a higher resolution neighbor. The main body stays intact while four border strips triangulate into triangle fans. Compared with that of de Boer's, it produce much nicer triangles and at the same cost. Also it can connect to four neighbors at the same time.

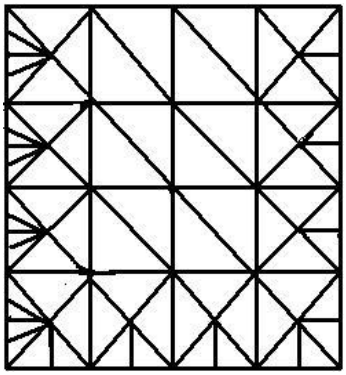

Fig7 Change the triangulation of 4 corners to connect to the neighbors. Few extra cost but better quality.

3.3 Vertex Morph

Vertex morphing may be the most effective way to reduce popping, in both continuous and discrete LOD. The basic approach is to find a factor f between 0 and 1, to interpolate vertices between two successive detail levels. Though most articles would like to use time as f, de Boer considered the distance a more proper choice.

$$f = \frac{d - D_n}{D_{n+1} - D_n}$$

For GeoMipMap is view-dependent, it's nice to be continuous when viewpoint changes. As for our revised equation, D_n is no longer available, and then f accordingly becomes a function of δ.

$$f = \frac{\delta - \delta_n}{\delta_{n+1} - \delta_n}$$

They are the same because δ should be proportional to d (In fact, δ is proportional to d^2, but it dose not matter much in application).

Detail steps are more or less the same as those presented in [6]. What worth mention are the edges shared with two tiles. The shared points of the same level may not be the same because of the different f. The simplest way was not to morph the edges. If you do want to do so, be peculiarly careful in choosing f and the morph level, especially to those vertices that shared with many line segments.

The amazing effect of morphing is to make the discrete algorithm seems continuous, switching smoothly between detail levels.

4. RESULTS

We test our engine on a PC with x86 Family Model 8 AT/AT Compatible，261,616KB RAM，NVIDIA TNT2 Video，1024*768*16 resolution. The height map and texture map are 1025*1025, tiling in 33*33. GeoMipMap, our improvement and ROAM (Turner's version[5]) are tested and compared.

Though floating point is used for morphing in GeoMipMap and our improved version, their performance really outweigh that of ROAM, in terms of both image quality and frame rate. Under the same screen pixel threshold, GeoMipMap and our version produce 30000-odd and 20000-odd triangles respectively (it's not the number of triangles actually rendered considering culling), reaching frame rates of approximately 25fps and 30fps. We can sense obvious fluctuation in frame rates using GeoMipMap, while our version goes more smoothly. Our refinement in level decision really works well in reducing triangles and level off the frame rate.

Our meshes look smoother in some strange geometry[Fig8]. Another effect is to reduce popping because the connective strips change more nicely from one level to another.

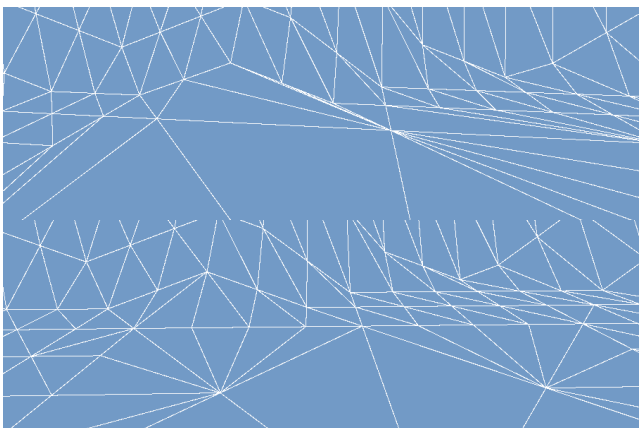

Fig8 Improvement of the mesh.
Above is original GeoMipMap and below is our improved version

ROAM produces constantly 10000 triangles and all the calculation are done with integer, without morphing. It works better in release mode at near 27fps, but only 20fps in debug mode. It proves that

high CPU overhead can slow down the performance. Since no morphing is done, popping is quite obvious especially when turning around. It produces much less triangles than the other program without better quality, but is still much slower.

5. CONCLUSIONS

The static LOD exploits 3D graphics hardware to speed up rendering, and achieves better performance than the dynamic LOD. We regard it as a promising approach to use widely in modeling huge objects (terrain in this case). Our refined equation for picking up detail level provides runtime control of the error threshold, which is though to be only in the dynamic approach, and zooming of the camera, as well as a flexible error bound that further reduces the number of triangles without affecting much the image quality. Moreover, a smooth transition between different levels can polish the oddness but introduce no extra cost. We see from this attempt that the static LOD can surely outweigh continuous LOD in performance, while producing the same pleasing quality.

REFERENCES

[1] W. de Boer，Fast Terrain Rendering Using Geometrical MipMapping, October 31, 2000 http://www.flipcode.com/tutorials/geomipmaps.pdf

[2] Mark Duchaineau, Murray Wolinksy, David E. Sigeti, Mark C. Miller, Charles Aldrich, Mark B. Mineev-Weinstein, ROAMing Terrain: Real-time Optimally Adapting Meshes, Los Alamos National Laboratory, Lawrence Livermore National Laboratory, October 19, 1997 http://www.llnl.gov/graphics/ROAM/roam.pdf

[3] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, Gregory A. Turner, Real-Time, Continuous Level of Detail Rendering of Height Fields, ACM SIGGRAPH 96，August 1996，pp 109-118 http://www.gvu.gatech.edu/people/peter.lindstrom/papers/siggraph96/siggraph96.pdf

[4] Sander Marechal， The Second Life of Brute Force Terrain Mapping， 6/20/2002 ， http://www.gamedev.net/reference/programming/features/bruteforce/

[5] Bryan Turner，Real-Time Dynamic Level of Detail Terrain Rendering with ROAM，April 03, 2000， http://www.gamasutra.com/features/20000403/turner_01.htm

[6] Dalgaard Larsen，Niels Jørgen Christensen， Real-time Terrain Rendering using Smooth Hardware Optimized Level of Detail，Technical University of Denmark Bent http://wscg.zcu.cz/wscg2003/Papers_2003/C05.pdf

[7] Ernest Szoka ，Supervisor: Dr. Wilf R. LaLonde ， Triangle Strip preserving LOD (T-Strip LOD)V 1.1 (last updated 5/15/2002)，Computer Science 95.495B Honours Project，Carleton University, Ottawa, Ontario, Canada，April，2002 http://chat.carleton.ca/~eszoka/tstriplod/tstrip.htm