Implementation and Optimization Issues of the Triangular Patch-Based Terrain System

Choong-Gyoo Lim**, Seungjo Bae, Kyoung Park, and YoungJik Lee

Electronics and Telecommunications Research Institute(ETRI)
161, Gajeong-dong, Yuseong-gu, Daejeon, 305-350, South Korea
{cglim,sbae,kyoung,ylee}@etri.re.kr
http://www.etri.re.kr

Abstract. A new dynamic LOD system is recently proposed to represent fairly large artificial terrains using hierarchical triangular patches[5]. Its mesh structure is unique from other conventional methods because it is designed to reduce the total number of draw primitive calls in a frame rather than the number of primitives. Another distinctive feature is its no-triangulation during the mesh adaptation by using its hierarchical layers and pre-constructed matching blocks. The purpose of the paper is to explain its implementation issues to make the new approach more understandable. Some of optimization issues are also discussed for better implementations.

Key words: artificial terrain, fairly large terrains, level of details, triangular patches, hierarchical layers, terrain cell windowing, dynamic cell loading

1 Introduction

Even with the high-end PCs, it is not easy to represent fairly large terrains in interactive applications because of the quite large number of mesh polygons involved. Dynamic LOD systems where a less number of polygons are used to represent remote areas and more polygons for close areas have been proposed in the literature such as ROAM(Real-time Optimally Adaptive Meshes)[2] and RQT(Restricted Quad-tree Triangulation)[6, 8, 9].

The mesh structure of ROAM is based on RTIN(Right Triangular Irregular Networks) which has a nice property that refining a triangle affects at most 2 triangles of each size in $[4.8^2]$ Laves nets[3]. ROAM optimizes re-triangulation by using the deferred lists of splitting and merging[2]. The mesh structure of RQT is based on quad-trees as its name implies. Each quad is refined according to the LOD value of the current area. This approach is well suited for texture allocation because of its rectangular shape while we need to enforce a block structure into a ROAM-based terrain system.

** This work was supported by the IT R&D program of MIC/IITA,[2006-S-044-02, Development of Multi-core CPU & MPU-Based Cross-Platform Game Technology]

The new approach is proposed to take better advantage of the pipelined architecture of modern GPUs. Modern GPUs are known to perform better with a less number of draw primitive(DP) calls[7, 1, ?]. While conventional methods trying to reduce the total number of primitives, the new one try to reduce the total number of draw primitives.

2 Hierarchical Triangular Patches

One of the major components in conventional terrain systems is how to represent terrains in polygonal meshes. ROAM represents them in right triangles as it is a derivation of RTIN while RQT does it in quadtree-based triangles. The new approach represents terrains in triangular patches. Another component is how to adapt to the newly computed LODs and how to fill up the possible cracks between neighboring patches, which is called re-triangulation. A moving window system is critical to terrain systems because of the limited amount of system resources.

2.1 Mesh Structure

Modern GPUs use polygons, mostly triangles, to represent 3D meshes. The performance of a graphics system, thus, heavily depends on the number of polygons it has to draw in a single frame. That's why most traditional methods try to reduce the number of polygons. Modern GPUs are, however, known to work better with a less number of draw primitive calls due to their pipelined architectures. They are designed to perform better on a group of consecutive primitive inputs than on separate primitive inputs. One can reduce the number of draw primitive calls significantly if a set of triangles can be represented by a triangle which is just a single triangle of another triangle set as in Fig. 1.

Instead of trying to reduce the number of polygons, the newly proposed method utilizes a unique mesh structure to represent terrains. It constructs hierarchical layers of triangular patches where a patch can be represented by a single triangle at the next layer as in Fig. 1. A triangular patch p_{0i} consists of $k \times k$ finest triangles from the given set of evenly spaced surface points, where k is the number of grid in each triangular patch. Non-overlapping p_{0i}s collectively constitute the lowest layer of the hierarchy $P_0 = \bigcup p_{0i}$ and represent the whole terrain in LOD 0. The resulting patches p_{ni} of layer n becomes a triangle of a triangular patch $p_{(n+1)i}$ of the next layer that collectively constitutes the next layer of the hierarchy $P_{n+1} = \bigcup p_{(n+1)i}$

2.2 Re-triangulation

Once the camera moves, terrain systems re-compute LODs of each terrain area. There are possibly some changes in LODs so that they have to re-triangulate terrain meshes in order to keep terrains continuous. Conventional methods firstly re-triangulate whenever splitting or merging of triangles is required. Splitting or

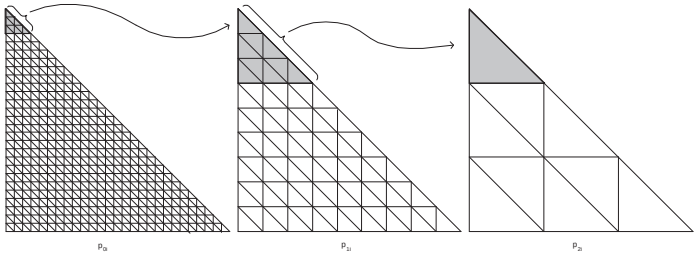

Fig. 1. A hierarchical layer of triangular patches when $k = 3$(from [5])

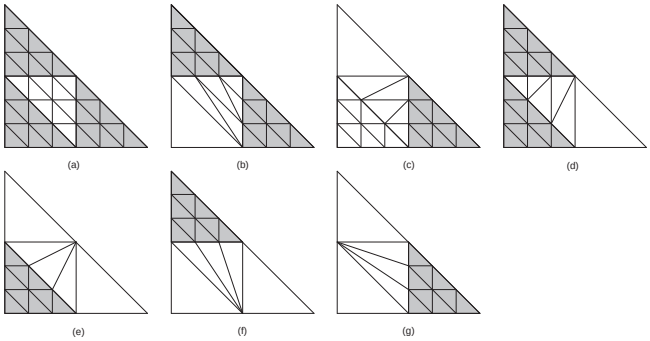

Fig. 2. 7 matching blocks(from [5])

merging itself cause the re-structuring of the terrain mesh, that is re-triangulation. This re-triangulation often leaves gaps between triangles. Another kind of re-triangulation is often carried out to fill up the gaps. As pointed out in Section 1, refinement of a triangle affects at most 2 triangles of each size in $[4.8^2]$ Laves nets.

The new system does not re-triangulate at all. Because it simply replaces the current patch with another patch from a higher or lower layer. If the LODs of 2 neighboring patches are different, then it replaces the current patch with one of 7 matching blocks. It identifies 7 matching blocks to stitch 2 neighboring patches of different LODs as shown Fig. 2. There are 7 different cases which are categorized by how the inner patch p of level i is surrounded by outer patches p_h, p_l, and p_r of level $i+1$, where the patch p_h borders on the hypotenuse of the inner patch and the patches p_l and p_r on the left and right sides, respectively, facing the hypotenuse. The LOD difference of neighboring patches is 1 at most, which is not a rare practice[4]

3 An Implementation

There may be many various implementations for the new method. One can implement it with a few phases as follows.

- **Phase 1: Initialization**
  The terrain system first creates a few threads for cell loading in background and multiple LOD computation. It also creates 1 vertex buffer and 7 index buffers to keep vertices and indices in memory. The vertex buffer can be used for every triangular patches because the numbers of vertices are all the same for the patches, which is 10. There are 7 different matching blocks such that the system needs 7 different index buffers of which one is also used for the original triangular patch.

- **Phase 2: Cell Loading**
  The whole terrain is divided into equal-sized cells. For each cell. there is a heightmap which is a collection of height values for the cell. The terrain system reads in the height map for the visible cells and the candidate cells. After reading the heightmaps, it constructs the hierarchical layers of triangular patches. For the simplicity, it uses 3 layers only. 3 layers are enough to validate the feasibility of the new method. Using more layers, however, means bigger terrain cells. In other words, it means less flexibility for cell management because each cell needs more memory.

- **Phase 3: Cell Binding**
  Now that the mesh structures have been constructed for the loaded cells, it needs to bind them together. Because it should propagate the LOD information on the boundary to the neighboring cells later during the phase of LOD computation

- **Phase 4: LOD Computation**
  The system selects a single patch from 3 layers based on how much detail it needs to represent the area, which is the LOD value for the area. The current implementation determines LODs in proportion to the distances to the camera. Another way is to compute the sizes of the edges of the bounding box of the current patch[5].

- **Phase 5: Vertex Buffer Merger**
  There is a severe penalty for the locking and transferring vertex buffers. The current implementation merges the whole vertex buffers of the current cell into a single one before transferring to the graphics pipeline. It saves the bandwidth of the AGP or PCI-express bus, thus improving the performance.

- **Phase 6: Rendering**
  The terrain system renders each patch according to the LOD of the patch. If there is no neighboring patch of different LOD from the current patch, it is rendered with the patch from the layer of the current LOD. If not, it replaces the current patch with one of the 7 pre-constructed matching blocks. In order to generate realistic terrain images,

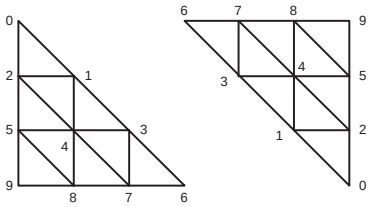

Fig. 3. Repeated uses of indices for the patches of different orientation

4 Optimization Issues

4.1 Re-use of Indexed Buffers

Indexed primitives are widely used in 3D applications as in [8] because of their smaller vertex buffers, which can maximize the use of the limited bandwidth of an AGP or PCI-express bus. Because of the same shape of triangular patches as shown in Fig. 3, the new system can transfers the indexed buffers at the beginning of the current cell and repeatedly use on all the patches, making better use of AGP or PCI-express buses. It transfers only the vertex buffers for the next patches.

A possible index list is $\{(0,1,2),(1,3,4),(1,4,2),\ (2,4,5),(3,6,7),(3,7,4),(4,7,8),(4,8,5),(5,8,9)\}$. It uses the fixed index lists for matching blocks as well with keeping the corresponding vertex buffers intact.

4.2 Dynamic Loading of Terrain Cells

The new terrain system uses a windowing system to avoid loading cells outside the viewing frustum. It loads the visible cells that move in the viewing frustum and off-loads the cells that move out.

There are, however, some IO delays when loading the cells that newly became inside of the viewing frustum. In order to avoid those delays which usually cause some popping artifacts, the new system load 2 rows and columns of neighboring cells using background threads. There is no popping artifacts, having the inside cells already loaded into the system.

It computes the LODs for inside patches before rendering. Those LODs of boundary patches are correct only if the LODs of candidate cells have been computed already. The system, thus, computes the LODs for candidate cells as well. The problematic situation occurs if the neighboring cells of a newly visible cell haven't been loaded yet and thus it can't compute the correct LODs for the patches of the new cell. The new system loads 4 more rows and columns of the candidate cells for correct LOD computation. 2 more rows and columns of candidate cells are enough if we allow some popping artifacts on the boundary cells, which can't be totally avoided because the LODs of the newly loaded candidate cells affects the LODs of the now visible cells.

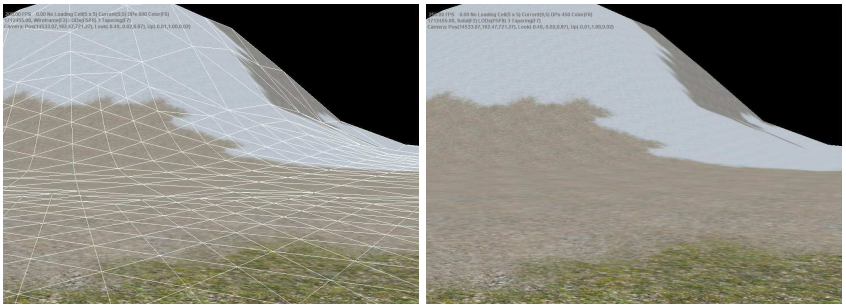

Fig. 4. A resulting surface in wireframe mode and in solid mode(from [5])

5 Conclusion

The implementation clearly shows that the newly proposed method can be used to represent fairly large terrains as some of screen shots are shown in Fig. 4.

References

1. K. Ashida. Optimising the graphics pipeline. In *China Joy 2004*. www.chinajoy.net, 2004.
2. M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Millder, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization '97 Proceedings*, 1997.
3. W. Evans, D. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. Technical Report 97-09, University of Arizona, Computer Science, 1997.
4. B. V. Herzen and A. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings SIGGRAPH 87*, pages 103–110. ACM SIGGRAPH, 1987.
5. C.-G. Lim. A dynamic construction and representation scheme of multi-level LOD terrains using triangular patches. *Submitted for Presentation to Pacific Graphics 2007*.
6. R. Pajarola. Overview of quadtree-based terrain triangulation and visualization. Technical Report UCI-ICS Technical Report No. 02-01, Department of Information & Computer Science, University of California, Irvine, 2002.
7. A. Rege. Optimization for DirectX9 graphics. In *Game Developers Conference 2004*. www.gdconf.com, 2004.
8. G. Snook. *Real-Time 3D Terrain Engines Using C++ and DirectX*. Charles River Media, 2003.
9. A. Szofran. Global terrain technology for flight simulation. In *Game Developers Conference 2006*, 2006.