Real-time Optimally Adapting Meshes - An Overview

Christopher Hoult

May 10, 2004

Abstract

This paper provides a brief overview of the ROAM (Real-time Optimally Adapting) Continuous Level of Detail (CLOD) terrain rendering algorithm, focussing on the triangle bintree, diamond splitting and merging and error metrics aspects of the algorithm.

1 Introduction

Real-time Optimally Adapting Meshes, or ROAM, is a continuous level-of-detail (CLOD) technique for rendering terrain in real-time (as the name suggests) [1]. A triangle mesh is built from a multi-resolution height-field, and heuristic methods are applied to determine which parts of the terrain need greater levels of detail in order to get the best appearance. In this way, large terrains can be rendered without impacting performance greatly - an analogy to this might be the practice of *mipmapping* 1. Further performance gains can be achieved through the application of view-frustrum culling and other reductive techniques, although the majority of these techniques (described by Duchaineau et al. [1]) are outside the scope of this paper.

CLOD involves the creation of a continuous mesh of polygons to render terrain efficiently, tessellating the underlying landscape either into quadrilaterals or triangles, and most often requires the use of a specialised technique to determine the level of detail required to display a particular tile. While ROAM is one such technique, there are others such as that of Lindstrom et al. [2] that similarly uses triangle bintrees, but uses a triangle fusion technique, rather than the splitting and merging technique described in this paper.

2 The Algorithm

The ROAM algorithm consists of a preprocessor and four runtime components. The preprocessor calculates error bounds for the entire landscape to be

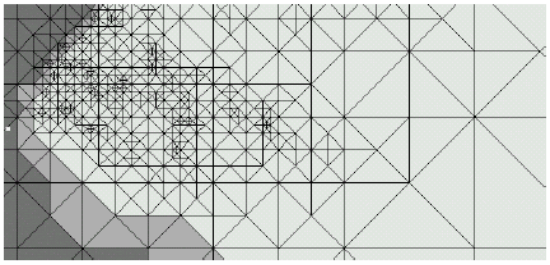

Figure 1: Triangulation example - dark region is outside of the view frustrum, light region inside, and grey region overlapping [1]

rendered; these bounds are used later on in the process to determine the acceptable margins of accuracy for individual portions of terrain. At runtime, each frame requires the following components [1]:

1. Recursive, incremental update to view-frustrum culling
2. Priority update only for output triangles that can potentially split / merged in phase 3
3. Triangulation update using greedy split and merge steps driven by two priority queues (for splits and merges, respectively)
4. As-needed updates for triangle strips affected by the culling changes from phase 1 and the splits/merges from phase 3

2.1 Error Bounds

In the preprocessing stage, the terrain data is used to calculate *wedgies*, pie-shaped volumes that determine the accuracy of each triangular section of the landscape - the larger the volume of the *wedgie*, the less accurately the section within will be rendered. These *wedgies* are then nested, so that the children of a high-level node collectively describe the same volume (as can be seen in Figure 2), but more accurately - this tree structure is built in a bottom-up fashion, from *wedgies* with zero volume. In this way, the tree contains multiple-resolution information on the terrain that can be used for error metrics, as well as to determine geometric screen-space distortions.

2.2 Triangle Bintrees

For the triangulation of the terrain, the ROAM algorithm makes use of 'triangle bintrees'. These are similar in concept to quadtrees, but consists of right-angled isosceles triangles which are successively split into equal halves to produce two more right-angles isosceles triangles, and so on, as can be seen in Figure 3. The triangle bintree is then traversed for the splitting and merging processes as

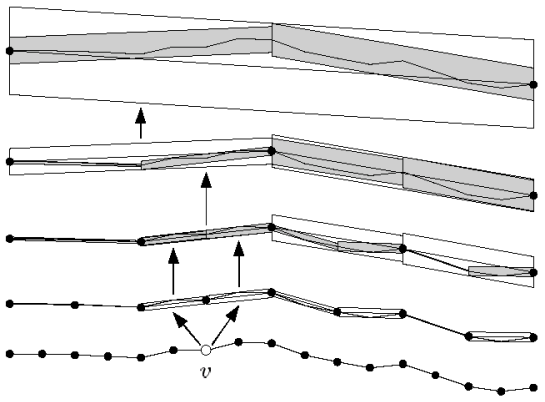

Figure 2: Nested *wedgies* for 1D domain with dependents of **v** [1]

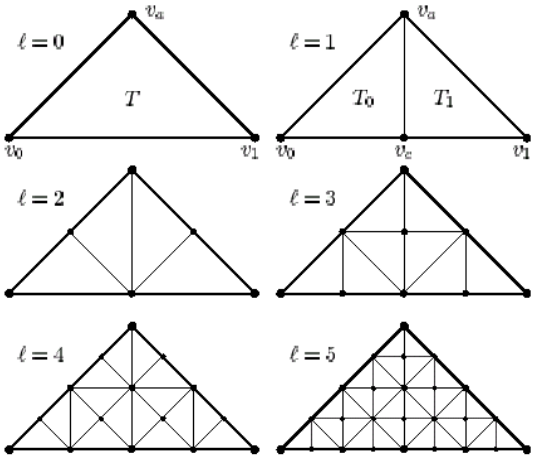

Figure 3: Levels 0-5 of a triangle bintree [1]

covered in the next subsection, and each vertex in the bintree is associated with a position in world space to form the terrain.

2.3 Splitting and Merging

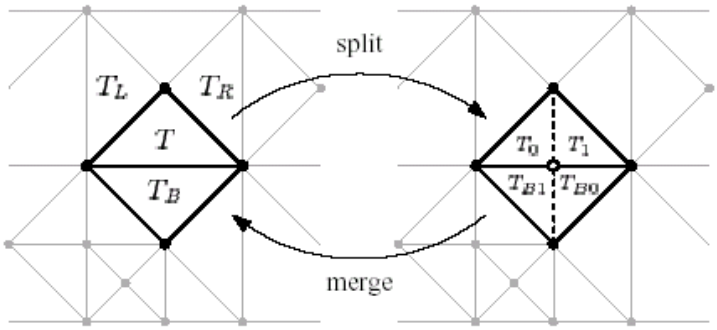

Figure 4: Split and merge operations on a bintree triangulation [1]

The splitting and merging of the triangle mesh is the process by which multiple levels of resolution are achieved in the terrain in order to reach the best combination of performance and detail. A triangle in the mesh, T, has three neighbours - T_B is the base neighbour, below the hypoteneuse; T_L and T_R are the left and right neighbours, respectively - as can be seen in Figure 4. Triangles T and T_B form what is known as a *diamond* when they are of the same level of the bintree, and the neighbours of any triangle T must be within one level either way (one level courser, equal or one level finer).

To split a *diamond*, a new vertex is created in the centre, and new edges created between this vertex and the apex vertex of each triangle. In this manner, T is replaced by triangles T_0 and T_1, and T_B is by triangles T_{B0} and T_{B1}; merging is the inverse of this operation (see Figure 4).

However, if neighbour T_B of triangle T is from a coarser level of the bintree than T, T_B must be split in order to preserve the property that neighbouring triangles must be within one level of the bintree of each other. This process must be performed recursively if the base neighbour of T_B is coarser than it is, and so on. As pointed out by Duchaineau et al., to reduce 'pops', in which such splits and merges are visible due to their instantaneous nature, animation of the new vertex from the centre of the *diamond* being split to its new location may be performed [1] - this is known as *Vertex Morphing* or *GeoMorphing* [3].

Duchaineau et al. describe a greedy algorithm that drives the split and merge process, based on separate priority queues for splits and merges [1]. All triangles in a triangulation are given *monotonic* priorities - that is, any triangle's priority is greater than or equal to its children's - and on this basis, the order of splitting and merging is determined. These priorities are based on the triangulation error compared with the associated *wedgie*.

2.4 View-Frustrum Culling

In order to decrease the work for the ROAM algorithm to engage in, every triangulation T is tested to see whether it is within the current view frustrum for each frame, and given an IN flag for each of the six halfspaces defining the frustrum, and given an overall flag depending on the state of its IN flags. If all IN flags are set, the triangle is given the flag ALL-IN, and is rendered; if the triangle is entirely outside one halfspace OUT is given; and in the remaining cases, DONT-KNOW is given. From this, it can be determined whether to process the children of T for rendering.

3 Performance

While the ROAM algorithm is relatively old in computer graphics terms, it is still in use today in smaller projects requiring terrain rendering. One of the original authors still maintains the ROAM algorithm, and is making some efforts towards a version 2. On the ROAM homepage, Duchaineau claims a triangle count of 56 million triangles per second at a frame rate of 36fps during slow movement using a modified ROAM LOD terrain engine displaying procedurally-generated terrain, produced on a 3.06GHz Pentium IV with dual-channel DDR266 memory and an ATI All-in-Wonder Radeon 9700Pro graphics card [4] - see Figure 5.

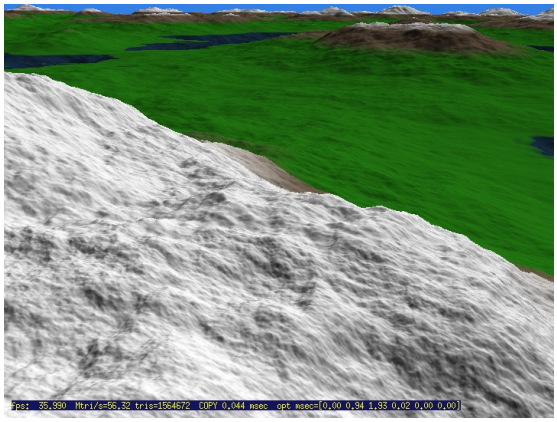

Figure 5: ROAM with AGP chunking [4]

In the original Duchaineau et al. paper [1], initial triangle counts of 6000 per frame (198000 triangles per second) were achieved at 33fps on a single R10000 processor Silicon Graphics Onyx with an Infinite Reality graphics board, although this pales in comparison with the triangle counts above. Turner [3], with a different implementation of the ROAM algorithm using a different error bounds system to the *wedgies* described in this paper (specifically, the Variance measure employed in the Tread Marks engine [5]), reports the frame rates in Table 1 for a 450Mhz AMD K6-2 with NVIDIA GeForce 256 DDR. Duchaineau et al. also point out that the ROAM algorithm's complexity is proportional to the number of triangles required to be rendered [1].

TriTree Nodes	Textured FPS	Solid-Fill FPD
5000	57	62
10000	30	36
15000	20	25
20000	16	19

Table 1: Frame rates and triangle counts [3]

4 Discussion

This paper has only scratched the surface of one part of the family of CLOD terrain engines and algorithms. While there are certainly a number of triangulation algorithms in use in a variety of situations, ROAM has shown itself to be one of particular note. One of the original developers is currently working on ROAM2, which promises even better, more accurate performance using a diamond-based triangulation, as well as taking advantage of the recent boom in 3d graphics hardware.

The ROAM CLOD algorithm is a versatile and scalable system for the determination of appropriate triangulation of meshes for the rendering of terrain. It has been shown to be appropriate to many levels of detail and speed, and will continue to be used as a optimal mesh-generating algorithm for long to come.

References

[1] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes", in *IEEE Visualization*, 1997, pp. 81–88.

[2] 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 *Proceedings of SIGGRAPH 1996*, August 1996.

[3] Bryan Turner, "Real-time dynamic level of detail terrain rendering with ROAM", April 2000.

[4] M. A. Duchaineau, "ROAM algorithm version 2.0", March 2003.

[5] "Tread Marks - battle tank combat and racing", February 2004.

[6] M. A. Duchaineau, M. Bertram, S. Porumbescu, and B. Hamann, "Interactive display of surfaces using subdivision surfaces and wavelets".

[7] A. Pfaffe, "Digital dawn graphics toolkit", 2001.

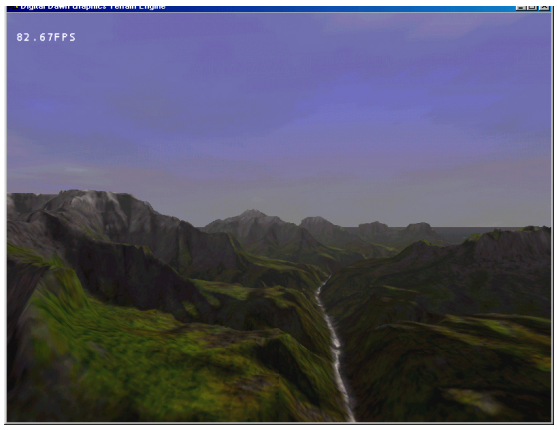

Figure 6: Digital Dawn Graphics Toolkit screenshot [7]

Typeset in LaTeX