International Journal of CAD/CAM Vol. 3, No. 2, pp. 111-117 (2003)

INTERNATIONAL JOURNAL OF CAD/CAM
www.ijcc.org

An Efficient Algorithm for Real-Time 3D Terrain Walkthrough

Michael Hesse and Marina L. Gavrilova*
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada

Abstract – The paper presents an efficient algorithm based on ROAM for visualization of large scale terrain models in real-time. The quality and smoothness of the terrain data visualization within a 3D interactive environment is preserved, while the complexity of the algorithm is kept on a reasonable level. The main contribution of the paper is an introduction of a number of efficient techniques such as *implicit coordinates method* within the patch array representing ROAM and *the viewpoint dependent triangle rendering method* for dynamic level of detail (LOD) updates. In addition, the paper presents experimental comparison of a variety of culling techniques, including a newly introduced method: *relational position culling*. These techniques are incorporated in the visualization software, which allows to achieve more realistic terrain representation and the real-time level of detail reduction.

Keywords: **Terrain rendering, ROAM, Implicit coordinates method, Culling techniques, GIS**

1. Introduction

High quality rendering and meshing techniques for displaying complex geographical data, such as terrain models, play an important role in the fast growing domain of CAD oriented towards Geographic Information Systems (GIS). According to a recent review, conducted as a part of Virtual Terrain Project on convergence of the fields of CAD, GIS and visual simulation [13], even today the vast majority of CAD is only 2D blueprints, not 3D models. The conversion from 2D to 3D is generally a difficult process requiring highly efficient algorithms, and there are practically no freely available tools for 3D CAD that could be useful in GIS. In addition, for exploring different kinds of geographic-based data sets on screen it is necessary to display data at interactive frame rates. Because of the inherent geometric complexity, this goal is often hard to achieve, unless the original data is approximated in order to reduce the number of geometric primitives that need to be rendered. This problem is particularly prevalent in applications dealing with large polygonal surface models, such as digital terrain modeling and visual simulation.

The paper addresses the above problems by introducing an efficient and easy to implement algorithm for visualization of large scale terrain models in real-time. The quality and smoothness of terrain data visualization within a 3D interactive environment is preserved, while the complexity of the algorithm is kept on the reasonable level. The Real-Time Optimally Adapting Mesh (ROAM) approach is used as an underlying model. We introduce a number of efficient techniques such as *implicit coordinates method* within the patch array representing ROAM, *the viewpoint dependent triangle rendering* method for dynamic Level of Detail (LOD) updates and some culling techniques. The efficiency is confirmed by experiments conducted on greyscale Digital Elevation Maps.

*Corresponding author:
Tel: +1-403-220-5105
Fax: +1-403-284-4707
E-mail: marina@cpsc.ucalgary.ca

2. The Motivation

The two traditionally used techniques for surface representation and visualization of terrain models are the TIN (triangulated irregular network) [14], and the uniform grid [5]. They are typically used to address problem of mesh simplification, although having some significant drawbacks preventing on-the-fly generation of multiple levels of detail. TIN models, for example, require a highly extensive computational effort for their generation. Because TINs are non-uniform in nature, surface following (e.g. for animation of objects on the surface) and intersection (e.g. for collision detection, selection, and queries) are hard to handle efficiently. This factor is especially important in many applications, such as games and CAD, where dynamic deformations of the mesh may occur. The most common drawback of regular grid representations is that the polygonalization is never optimal [7]. Large, flat surfaces may require the same polygon density as small, rough areas do. This problem may be alleviated by reducing the overall complexity and applying temporal blending, or morphing, between different levels of detail [12]. Some visual simulation systems handle transitions between multiple levels of detail by "alpha blending"

two models during the transition period. Ferguson [4] claims that such blending techniques between levels of detail may be visually distracting, and discusses a method of Delaunay triangulation which smoothly matches edges across areas of different resolution. However, this method is inherently difficult to implement.

Another approach based on maintaining of Real-time Optimally Adapting Meshes (ROAM) was introduced by Lindstrom [7], and utilized by DeBerg and Duchaineauy [2, 3]. The approach suggested in [7] is based on a hierarchical quadtree technique. In order to reduce the projected pixel error, the terrain is dynamically triangulated in a bottom up fashion according to the distance to the point of view. Since resolution is allowed to change smoothly, the result is a much better image quality. However, this algorithm still has a room for improvement. When the viewpoint is changing, the triangulation is continuously updating, resulting in a so-called "popping". As the observer approaches an area with detail information, this detail will suddenly appear at a certain distance. Another problem, crucial for ROAM algorithms, is controlling a smoothness and quality of the rendered image. While many simplification methods are mathematically viable, the level of detail generation and selection are often not directly coupled with the screen-space error resulting from the simplification.

To eradicate these problems, a number of approaches were suggested. A rapid geomorphing algorithm, performing top-down manipulations on the quadtree data structure, was presented in [11]. A recently proposed method of view-dependent refinement allows to build a mesh with a small number of triangles that for a given view is a good approximation of the original, dense mesh [8]. Novel methods for ROAM optimization, utilizing a variety of culling techniques, continue to evolve [1, 6, 8, 14]. As can be seen from the above discussion, efficient and easy algorithms for displaying complex geographical data are still in high demand, and this paper introduces one of them and discusses its performance on an example of real-time visualization of 3D Digital Elevation Models.

3. The Proposed Approach

Real-time Optimally Adapting Mesh (ROAM) method is selected as an extendable, efficient tool for internal data representation and dynamical updates of the terrain model. The method is extended with an original *implicit coordinates method* within the patch array *and the viewpoint dependent triangle rendering* method for dynamic level of detail (LOD) changes. The method is characterized by the following set of unique features:

- Smooth, continuous changes between different surface levels of detail
- Dynamic generation of levels of detail in real-time
- Introduction of implicit coordinates method within the patch array for more efficient ROAM representation
- Introduction of the viewpoint dependent triangle rendering method for dynamic level of detail (LOD) updates
- Implementation of culling techniques, including the original Relational Position culling for more efficient terrain rendering
- Flexibility in choosing/selecting various culling techniques
- Reduction in the amount of time required to achieve a given triangle count
- Application of error metrics for increased smoothness and continuity

The terrain data sets studied are the simple gradual contour changes and the complex steep contour changes, represented by greyscale Digital Elevation Maps of 1024 pixels by 1024 pixels. Each experimental set is internally represented by three quadrant detail levels, corresponding to 16, 64 and 128 nodes per side within the binary tree structure. The load time, the total number of triangles per path, the total number of culled triangles per path, the number of frames per second and the number of triangles per frame are examined for this structure. The occlusions culling techniques are individually and collectively combined with ROAM technique and examined with the different representations of detail levels to verify the algorithm feasibility and efficiency.

3.1. Digital Elevation Model (DEM)

Digital Elevation Model (DEM) can refer either to a specific elevation file format or to sources of elevation data in general. DEM data is usually stored as an array of regularly spaced elevation values, referenced horizontally either to a Universal Transverse Mercator (UTM) projection or to a geographic coordinate system. The grid cells are spaced at regular intervals along south to north profiles that are ordered from west to east. A standard grid posting is interpolated directly from the contour files to create DEMs with 10 - 90 meter (<1 - 3 arc second) resolution (depending on the source paper map scale or contour interval) [10]. Two greyscale Digital Elevation Model (DEM), representing progressive contours with gradual elevation changes and sudden steeper elevation changes (Fig. 1) were studied.

3.2. Real-time terrain rendering algorithm

Our approach to terrain data rendering is based on the Real-time Optimally Adapting Mesh (ROAM) method (see [7]). The method presents a highly flexible and adaptable method for representing continuous mesh with controlled Level of Detail. It allows controlling the details of a mesh and to maximize the quality and minimize the number of triangle primitives used in the process. The ROAM-based system constructs a consistent

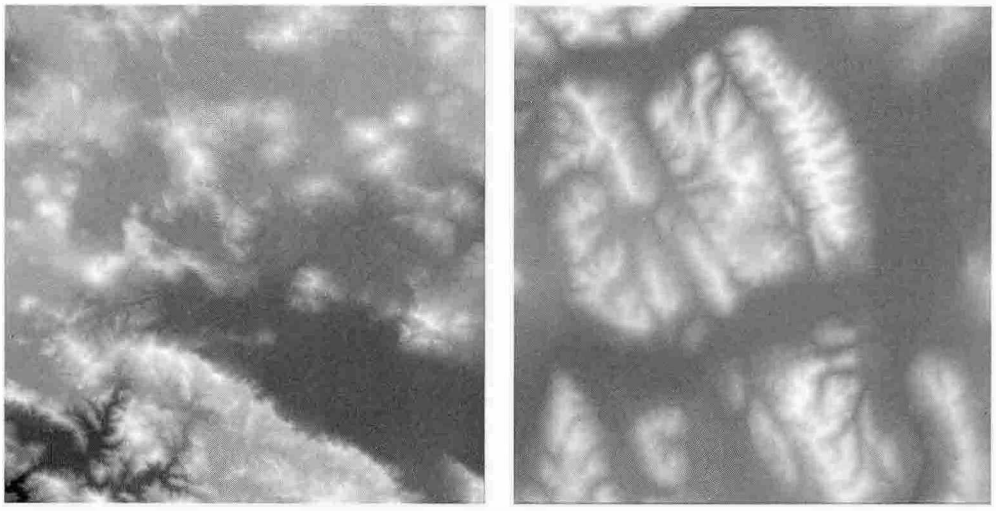

Fig. 1. Gradual elevation changes (left) and steep elevation changes (right).

and dynamic detail representations of terrain data by utilization of two main priority queues. They are driven by *split and merge* operations that adjust the terrain detail level dynamically. The split and merge functions are both built and changed from the data information held within a preprocessed Binary Triangle Tree data structure. We suggest to use the implicit coordinates method within an array of patch objects for memory conservation. In addition, we use a specific viewpoint dependent triangle procedure to reduce the total amount of computations needed to render the terrain data. The more detailed description of the algorithm is provided below.

The Real-time Terrain Rendering Algorithm

1. *Preprocessing step.*
   1.1 Build a Binary Triangle Tree data structure to represent the geometric properties of the rendered terrain.
2. *Dynamic rendering step.*
   2.1 Create two queues, the split queue and the merge queue, to keep the priorities for each individual triangle in the mesh triangulation;
   2.2 Implement the split and merge operations for updating the triangulated mesh.
   2.3 Perform recursive, incremental update to real-time optimally adapting mesh, using the implicit coordinates method within an array of patch objects;
   2.4 Optimize processing using the viewpoint dependent triangle rendering method;
   2.5 Perform updates for triangle strips affected by the culling changes;
   2.6 Implement error metric control.

3.3. Split and merge function

As it was mentioned above, the ROAM algorithm is built around a Binary Tree structure that supplies triangle information for the split and merge operations [7]. Obtaining any level of triangulation from a sequence of splits or merges can be done from any other triangulation level. Two triangles that share the same base and are on the same detail level are referred to as a *diamond*. The split operation adds a new vertex at the diamond center resulting in the creation of four new right-isosceles triangles, which will increase the number of triangles representing a terrain area. As the number of triangles increase, the detail level that can be represented will also increase. The merge operation works inversely to the split operation.

In our implementation, the split and merge operations provide a flexible framework for making detailed updates to the triangulated mesh (Fig. 2). The basic idea of each queue is to keep the priorities for each individual triangle in the mesh triangulation. The split operations would then start with the base triangulation level in the queue and then repeatedly split the triangle until the highest priority triangle is reached. The only requirement for the split priority queue is that the child's priority level must not be more than its parent's. The merge priority queue allows the merge operation to start from the previously rendered mesh triangulation. This allows a more consistent and quicker frame-to-frame coherence.

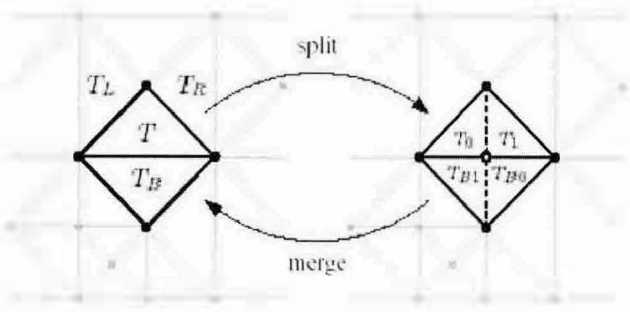

Fig. 2. Split and merge operations.

3.4. Implicit coordinates method

In our implementation, patches of triangles are used to create and manage the mesh approximation within the terrains landscape. We introduce the *implicit coordinates method* within an array of patch objects to ensure more efficient memory usage. The method is based on the following idea. Instead of storing explicitly X, Y and Z coordinates for each vertex of a triangle, implicit coordinates, within the landscape, are stored for the isosceles right triangles that will be rendered onscreen. The advantage of this approach is that implicitly defining coordinates saves 36 bytes of RAM per triangle. An index within the patch array references an individual Binary Triangle Tree that in turn stores the references to each triangle level of detail for that patch. The size of the patch determines the relative size of each patch within the landscape. The patch objects are held within the Landscape object. The landscape object is built by combining each patch section until the entire terrain is rendered.

3.5. Viewpoint dependent triangle rendering

A conventional method to reduce the amount of computations needed to render a complex scene is to apply Level of Detail (LOD) techniques. LOD method determine which sections of rendered mesh require less detail based on any number of criteria. In this paper, we extend the LOD technique with the *viewpoint dependent triangle rendering* method. The method allows more flexible information storage for dynamic and interactive first person view rendering. We allow portions of the terrain that are currently too far away to be rendered with few triangles, and the same sections of terrain to be rendered with more triangles if the viewpoint moves closer. This is done by examining the field of view with the view frustum to determine which patch sections need more detail due to their proximity to the user.

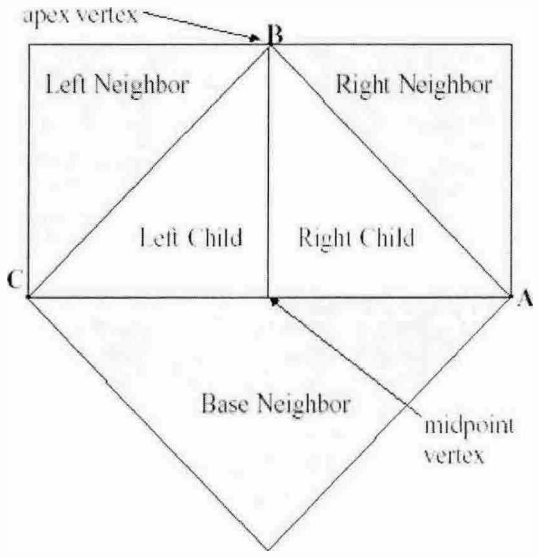

Fig. 3. Binary triangle tree data structure.

3.6. Recursive bintree data structure

To satisfy the LOD requirement a binary triangle tree structure will be used to hold the various levels of detail that is needed by the graphics-rendering engine. In the case of ROAM, a binary triangle tree structure, or a *bintree*, is a recursive structure where, at its lowest level, represents a right-isosceles triangle (see Fig. 3). In our implementation, each patch of terrain will have an individual bintree to define the triangle detail levels. The triangle bintree structure starts with the base terrain, either the least detail representation or the detail level from the previously rendered image, in the leaf components of the structure. The need for a change in detail level is determined by examining the corresponding error metrics.

4. Improving Rendering Quality

Data culling is a process of selecting, from the whole scene, particular information that needs to be rendered. Culling at this level is often achieved by using geometry-based methods to determine which scene information needs to be rendered. We implement three types of geometric culling algorithms to improve smoothness and reduce "popping": View Frustum Culling, Backface Culling and an original Relational Culling technique (see Fig. 4 and Fig. 5). As far as we know, this is the first study that performs detailed comparison analysis of a variety of culling methods in application to different GIS data.

The *view frustum* is the volume of space that includes everything that is currently visible from a given viewpoint. Six planes arranged in the shape of a pyramid with the top removed, define the view frustum area. If a point or object is inside this volume then it is within the frustum area and is potentially visible. If a point is outside of the frustum then it is not visible to the user, it needs not be rendered. To determine the position of the points and object, their bounding volumes are computed. If the bounding volume lies on

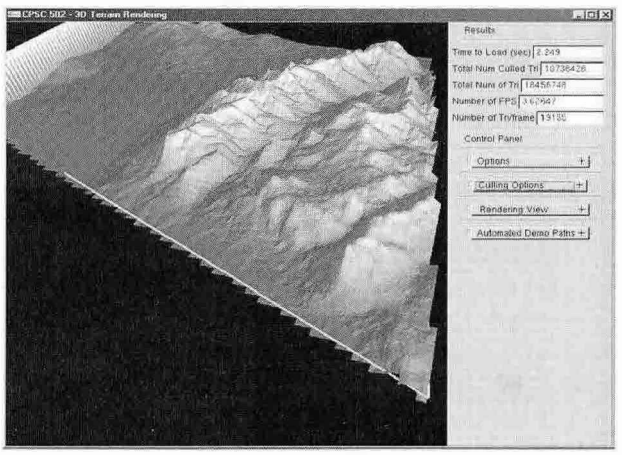

Fig. 4. Sample culling of rendered terrain.

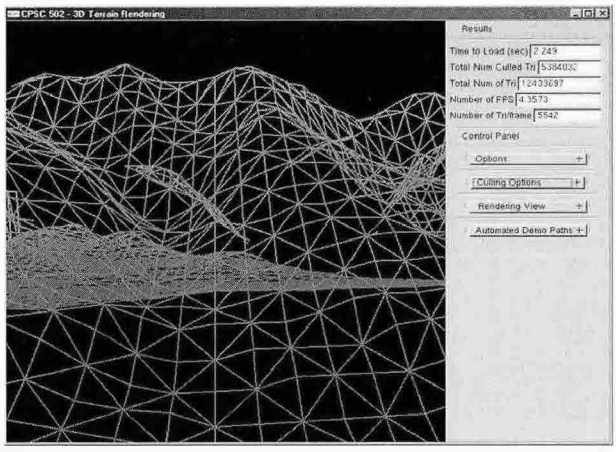

Fig. 5. Sample meshing of rendered terrain.

one of the frustum edges then that bounding volume is further subdivided into smaller bounding volumes until each object is either determined to be inside or outside the frustum area. If at the lowest detail level an object still lies on a frustum edge, the portion of the object inside the frustum area is rendered while the rest is culled.

Historically, the geometric shapes used as bounding volumes are boxes, or spheres, that are quicker to test and require less memory to store. They also require only four floating-point numbers for representation, while a bounding box requires eight. The disadvantage of a bounding sphere is sometimes it needs to be very large to surround an object that could be easy encompassed by a bounding box. To rectify this disadvantage, we link multiple bounding spheres in a chain to better simulate the object shapes.

We store the essential information described by the bounding spheres in the hierarchies of bounding volumes as a Direct Acyclic Graph (DAG). This structure will allow for quick and easy access of object information based on their relative positions. Modeling is done using OpenGL environment to extract the six planes of the current view frustum by retrieving the current PROJECTION and MODELVIEW matrices, combining the two, and then extracting the frustum values from the resulting matrix. Our algorithm results in four numbers that can represent the six planes. A point is within the view frustum if it is in front of all six planes simultaneously. To determine if a patch is within the view frustum a bounding box approach is utilized. The eight corners of the patch box are used to determine if the patch should be rendered or not rendered based on the same procedure as identifying if a point is with the view frustum.

The second method that we implement in this project is the *backface culling*. Based on a user's eye-space, back-facing polygons are located on the far side of an opaque object. These polygons, although part of the viewer's scene, are not visible to the viewer and do not need to be rendered. Once the polygons are determined to be back-facing, they can be culled before the scene is rendered. We calculate the normal of the projected polygon to determine if it is back-facing. This test involves calculating the polygon's normal and the vector formed from the viewing point to any point on the polygon.

The third technique that we introduce is the original *relational position culling* technique, based on pre-processing the terrain landscape into patches. Each patch would contain the Binary Triangle Tree structure of its terrain data and store each triangle's detail information within its node. Additionally, a visibility flag is stored to determine which patch is seen within the view frustum. This approach is developed to quickly cut the generalized unnecessary terrain data from the terrain data set. Initially, the algorithm determines the frustum triangle corners from a two-dimensional (2D) view frustum, which gives the algorithm the user and user's viewpoint's positions. These three points are used to determine the minimal rectangle that encompasses the 2D view frustum. Any points not within this rectangle are immediately culled. Advantages of this method are its simplicity and performance, that are discussed in the previous work [6] by the authors.

5. Algorithm Performance Analysis

The main contribution of this work is in the development of an efficient and adaptive real-time rendering algorithm based on ROAM technique, combined with a number of methods for increased rendering speed, smoothness and realism. The algorithms were implemented in Open GL.

When examining the results from the experiments, several relationships were observed. One of the most notable observations was the change of the number of frames per second (frame rate) during each of the paths corresponding step. Fig. 6 demonstrates the change in frame rate with all three culling techniques (View Frustum, Position based and Backface culling) enabled with three distinct patch sizes represented by their corresponding array size. The size-16 frame rate performs as expected with the graph trend line remaining rather flat and consistent throughout the entire experimental path, except for its initial load up stage, which is completed by the 25th frame. The size-64 frame rate demonstrates some interesting qualities. The first 100 frames emulates a similar pattern as the size-16 trend line with the exception of a consistently lower frame rate due to the increase of the number of patches that need to be rendered. The size-64 trend illustrates a significant increase in frame rate from frame 100 to 170. The increase corresponds to the first set of constant right and left hand turns in the experimental path that lasts until frame 161. Note that there are no significant differences in the number of frames rendered per second for different terrain models.

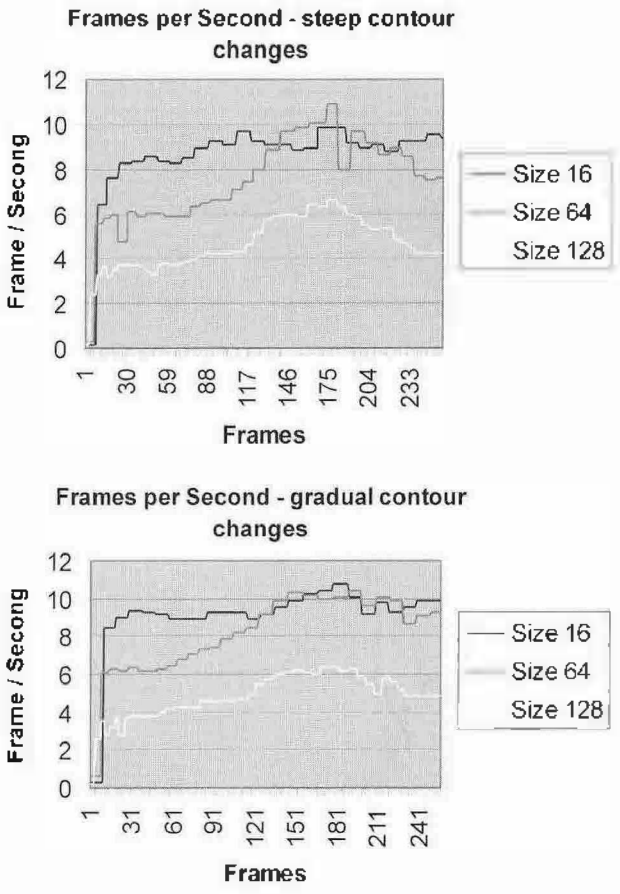

Fig. 6. Number of frames per seconds for two models.

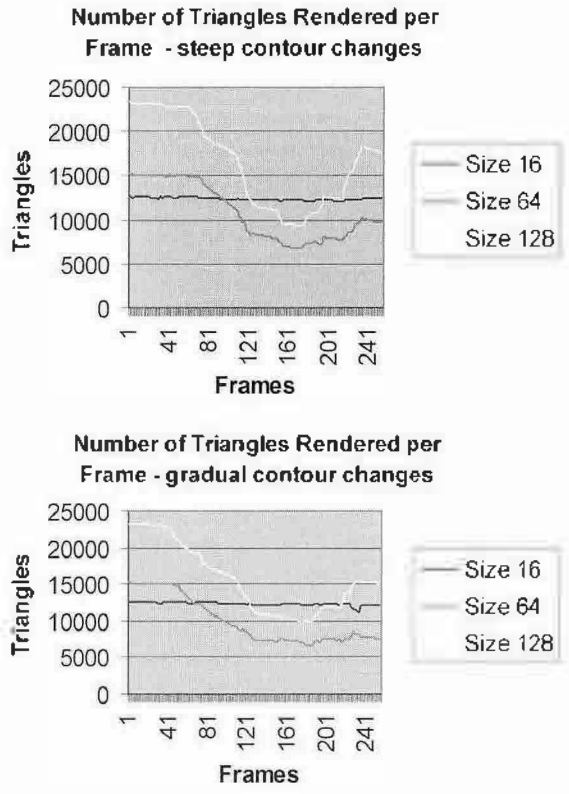

Fig. 7. Number of triangles rendered per frame for two models.

This set of turns can also be seen when examining Fig. 7 for both terrain models. The decrease in triangles per frame corresponds to the first left hand turn during a specific path over the terrain, most likely due to the quick pace of the turn and the difficultly of the rendering engine to propagate the necessary triangle detail levels before the next turn begins. As each turn is performed, the viewed landscapes true detail level is reduced. The number of triangles per frame continues to reduce as the turn progresses. This is the result of the view frustum largely shifting out of the frame of view. The trend line flattens as the forward movement allows the rendering engine time to increase the detail level of each frame, due to the limited changes in the view frustum.

The rise of the size-64 trend line frame 180 is due to quick left and right turns which would leave the middle section of the view frustum untouched with only the frustum edges needing to be recomputed. The largest increase in the size – 64's trend line is due to the experimental path moving directly backwards (back stepping), which significantly decreases the amount of the view frustum that must be recomputed.

Now, consider the time complexity of the algorithm. Analysis of time required for split and merge functions is quite interesting. It was anticipated that there would be an equal sharing of time between both the split and merge functions. That is, as one set of triangles needs to be split it would cause another set of triangles to congruently be merged adjusting to meet the overall set triangle detail level. When the application is examined in further detail, this is not always the case. Initially, as the triangles propagate through the landscape to their set detail level there is a relatively equal amount of time being spent in both algorithms. Interestingly, this is not the case when the culling techniques are active. In this case, a higher proportion of application time is spent in the split function and is directly related to the number of triangles being rendered per frame, Fig. 7, and the movement of the view frustum. When a patch is culled from a the frustum, it has some detail level of rendered triangles. These triangles are eliminated without the use of the merge algorithm. This elimination of triangles allows the application to split new triangles on the new patches entering the frustum without increasing the overall triangle detail level. As the movement of the frustum increases with consecutive turns, this relationship becomes even more pronounced.

This observation helps to clarify the relationship between the number of triangles per frames and the movement of the frustum seen is Fig. 7. As the movement of the frustum becomes more continuous there is a reduction in the number of triangles being rendered per frame. This would be directly related to the disappearance of the rendered patches, with some triangle detail level, being eliminated the frustum view and the appearance of new patches, with lowest detail level, being brought

into view. As the turn becomes more continuous, the greater the number of patches that is eliminated and the higher the demand on the split algorithm.

The time efficiency of the algorithm is also related to the size of the patch set. It also explains the algorithm behaviour for Size-16 and Size-64 patch sizes in Fig. 7. Since the Size-16 patches are larger, there is a greater area outside the view frustum being rendered. This wasted rendering area would act as a frustum buffer. Even though the patches would be eliminated at the same pace as a smaller patch size frustum, Size-64, these larger patches would have the smallest amount of detail on the portions of the patch that are outside the frustum site lines. Therefore the elimination of these patches will have less effect on the number of triangles being rendered per frame. This relationship also adjusts the time usage relationship between the split and merge algorithms. Due to the frustum buffer the merge function has time to adjust the triangle detail level of the patch that will be eliminated. This triangle merge reduction allows the split algorithm more time to readjust triangle detail level before the jolt of having the entire patch eliminated. As the patch size becomes smaller the adjustment time for the merge algorithm becomes less and the split algorithm more dominant.

6. Conclusions

The main contribution of this work is in the development of an efficient and adaptive real-time rendering algorithm based on ROAM technique, combined with a number of methods for increased rendering speed, smoothness and realism. Examining both the number of frames per second and the number of triangles per frame suggests a number of conclusions to be drawn. When investigating the experimental path's frame rates with all of the culling techniques active, we show that the patch sizes within the landscapes are significantly related to the change of frustum position. This correlation is confirmed by examining the number of triangle rendered per frame.

This paper also explored the time efficiency of the split and merge algorithms. The experimentation demonstrated that patch size has a correlation between the times being spent in each algorithm. Larger patch sizes effectively provide frustum buffer. As the patch size becomes smaller, the merge algorithm becomes less effective but the split function become more in demand due the rapid removal of patch with large detail levels. These results provide a unique inside view on the correlation between the different mechanisms, incorporated together in the presented algorithm for realistic and efficient terrain representation and the real-time level of detail reduction. Further investigation of culling techniques and error metrics for improved visualization results for different types of rendered terrain models is planned.

Acknowledgements

Authors would like to express their gratitude to GEOIDE and NSERC granting agencies for their partial support of this project.

References

[1] Blow, J. (2000), "Terrain Rendering at High Levels of Detail," *Game Developers' Conference*, San Jose, California, USA.

[2] De Berg, M. and Dobrint, K. (1995), "On levels of detail in terrains," *Proc. 11th ACM Symposium on Computational Geometry*, ACM Press, C26-C27.

[3] Duchaineauy, M. *et al.* (1997), "ROAMing Terrain: Real-Time Optimally Adapting Meshes," *IEEE Visualization '97 Proceeding*, 81-88.

[4] Ferguson, R., Economy, R., Kelly, W. and Ramos P. (1990), "Continuous Terrain Level of Detail for Visual Simulation," *IMAGE V Conference*, 144-151.

[5] Gross, M. H., Gatti, R. and Staadt, O. (1995), "Fast Multiresolution Surface, Meshing," *Proceedings of Visualization '95*, 135-142.

[6] Hesse, M. and Gavrilova, M. (2003), "Quantitative Analysis of Culling Techniques for Real-time Rendering of Digital Elevation Models," *WSCG 2003*, Science Press, 130-137.

[7] Lindstrom, P. and Koller, D. (1996), "Real-time continuous level of detail rendering of height fields," *Computer Graphics, SIGGRAPH 1996*, 109-118.

[8] Lindstrom, P. and Pascucci, V. (2002), "Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization," *IEEE Visualiz. and Comp. Graphics*, **8**(3), 239-254.

[9] Lloyd, B. and Egbert, P. (2002), "Horizon Occlusion Culling for Real-time Rendering of Hierarchical Terrains," *IEEE Visualization 2002*, Boston, Massachusetts.

[10] National Mapping Division, *U.S.Geological Survey*, US GeoData Dig. Elev. Models.

[11] Rottger, S., Heidrich, W., Slusallek, P. and Seidel, H. (1998), "Real-Time Generation of Continuous Levels of Detail for Height Fields," WSCG'08, 315-322.

[12] Taylor, D. C. and Barret, W. A. (1994), "An Algorithm for Continuous Resolution, Polygonalizations of a Discrete Surface," *Graphics Interface*, **94**, 33-42.

[13] VTP Virtual Terrain Project, *http://www.vterrain.org/index.html*.

[14] Zhao, Y., Zhou, Y., Shi, J. and Pan, Z. (2001), "A Fast Algorithm for Large Scale Terrain Walkthrough," *CAD/Graphics '2001*, International Academic Publishers.