

EVALUATION OF
TERRAIN RENDERING ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY

EMİN İNAM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

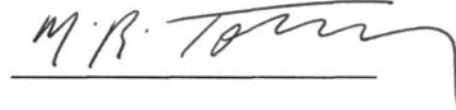
SEPTEMBER 2005

Approval of the Graduate School of Çankaya University



Prof. Dr. Yurdahan Güler
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of
Master of Science



Prof. Dr. Mehmet Tolun
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully
adequate, in scope and quality, as a thesis for the degree of Master of Science



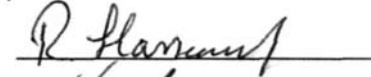
Asst. Prof. Dr. Reza Hassanpour
Supervisor

Examining Committee Members

Prof. Dr. Hayri Sever



Asst. Prof. Dr. Reza Hassanpour



Dr. Abdülkadir Görür



ABSTRACT

EVALUATION OF TERRAIN RENDERING ALGORITHMS

İnam, Emin

Master of Science Thesis

Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Reza Hassanpour

September 2005, 67 pages

Terrain rendering plays an important role in outdoor virtual reality applications, games, Geographic Information System (GIS), military mission planning's and flight simulations, etc. Many of these applications require real-time dynamic interaction from end users and thus are required to rapidly process terrain data to adapt to user input. Typical height fields consist of a large number of polygons, so that even most high performance graphics computers have great difficulties to display even moderately sized height fields at interactive frame rates. The common solution is to reduce the complexity of the scene while maintaining a high image quality. This thesis is an evaluation of three real-time continuous terrain levels of detail algorithms described in the papers ROAMing Terrain: Real-time Optimally Adapting Meshes by Duchaineau, Real-Time Generation of Continuous Levels of Detail for Height Fields by Röttger and Fast Terrain Rendering Using Geometrical MipMapping by Willem H. de Boer. The evaluation and comparison of the algorithms is based on the trade-off of polygon count to terrain accuracy over separate test data sets. The main aim of this thesis is research on terrain rendering algorithms that is generate high quality image in real-time with using height data.

Keywords: Terrain Rendering, Continuous Level of Detail (CLOD)

ÖZ

ARAZİ ÇİZME ALGORİTMALARININ DEĞERLENDİRİLMESİ

İnam , Emin

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Yöneticisi: Yrd. Doç. Dr. Reza Hassanpour

Eylül 2005, 67 sayfa

Arazi çizimleri, sanal gerçeklik uygulamalarında, üç boyutlu bilgisayar oyunlarında, Coğrafi Bilgi Sistemlerinde (CBS), askeri görev planı oluşturmada ve uçuş simülasyonlarında önemli bir rol oynamaktadır. Bu uygulamaların çoğu, son kullanıcı ile gerçek zamanlı dinamik etkileşim gerektirmektedir. Arazi verisi hızlı bir şekilde işlenerek kullanıcı girdisine göre uyarlanır. Standart yükseklik alanları çok sayıda poligon içerir. Bu yüzden, yüksek performansa sahip bilgisayarlarda bile belli bir kalitede yükseklik verilerini görüntülemek zor olabilir. Bu sorunun ortak çözümü, görüntü kalitesini koruyarak verinin karmaşıklığını azaltmaktır. Bu tezde Duchaineau, Röttger ve Willem H. de Boer tarafından yazılan makalelerde ortaya konulan algoritmalar karşılaştırılacaktır. Algoritmaların değerlendirilmesi sırasında farklı veri kümeleri uygulanacak, poligon sayısı ve arazi doğruluğu esas alınacaktır. Bu tezin amacı, yükseklik verilerini kullanarak, gerçek zamanda, yüksek kalitede arazi görüntüsü elde edilmesini sağlayan, arazi çizme algoritmalarını incelemektir.

Anahtar Kelimeler: Arazi Çizme, Sürekli Detay Düzeyi (CLOD)

*To my Mom, my Dad,
and SESE.*

ACKNOWLEDGMENTS

I express sincere appreciation to Dr. Reza Hassanpour for his guidance and insight throughout the research. To my parents, I offer sincere thanks for their unshakable faith in me and their willingness to endure with me the vicissitudes of my endeavors.

TABLE OF CONTENTS

ABSTRACT.....	iii
ÖZ	iv
TABLE OF CONTENTS.....	vii
LIST OF TABLES	ix
LIST OF FIGURES	xi
LIST OF FIGURES	xi
ABBREVIATIONS	xiii
CHAPTER 1	14
INTRODUCTION	14
1.1 3D Terrain Overview	15
1.2 Real-Time Continuous Level of Detail.....	15
1.3 Challenge of Terrain Rendering Algorithms.....	16
1.3.1 Tears, Cracks, and T-Junctions.....	17
1.4 Motivation.....	19
1.5 Outline of Thesis.....	20
CHAPTER 2	21
BACKGROUND ON TERRAIN RENDERING ALGORITHMS.....	21
2.1 Multiresolution Techniques for Terrain.....	21
2.1.1 Top Down and Bottom Up.....	21
2.1.2 Regular Grids and TINs	22
2.1.3 Quadtrees and Bintrees	24
2.2 Terrain Rendering Algorithms.....	26
2.2.1 The ROAM Algorithm.....	26
2.2.2 Real-Time Generation of Continuous LOD.....	30
2.2.3 Fast Terrain Rendering Using Geometrical MipMapping	34
CHAPTER 3	38
EVALUATING ALGORITHMS	38
3.1 Testing Environment.....	38
3.2 Test Data	39
3.2.1 TerraGen Test Dataset.....	39
3.2.2 Shuttle Radar Topography Mission (SRTM) Test Dataset.....	42
3.3 Evaluation Criteria	47

3.3.1	Terrain Accuracy.....	47
3.3.2	Polygon Count.....	49
3.4	Test Cases and Results.....	49
3.4.1	Terrain Accuracy Viewpoint Results	49
3.4.2	Desired Number of Polygons Viewpoint Results	58
3.5	Discussion of Results	59
CHAPTER 4	63
CONCLUSIONS	63
FUTURE WORKS	64
REFERENCES	65

LIST OF TABLES

Table 3.1 Test Machine Specifications	38
Table 3.2 TerraGen - Subdivide & Displace Method terrain data	40
Table 3.3 TerraGen - Perlin Noise Method terrain data.....	40
Table 3.4 TerraGen – Multi Perlin Method terrain data	41
Table 3.5 TerraGen – Ridged Perlin Method terrain data.....	41
Table 3.6 TerraGen – Ridged Multi Perlin Method terrain data.....	42
Table 3.7 N37E029 zone SRTM data	43
Table 3.8 N37E034 zone SRTM data	44
Table 3.9 N38E027 zone SRTM data	44
Table 3.10 N38E032 zone SRTM data	45
Table 3.11 N39E041 zone SRTM data	45
Table 3.12 N39E044 zone SRTM data	46
Table 3.13 N41E026 zone SRTM data	46
Table 3.14. Algorithm input parameters	50
Table 3.15. TerraGen - Subdivide & Displace Method results.....	50
Table 3.16. TerraGen – Perlin Noise Method results.....	50
Table 3.17. TerraGen – Multi Perlin Method results	51
Table 3.18. TerraGen – Ridged Perlin Method results	51
Table 3.19. TerraGen – Ridged Multi Perlin Method results	52
Table 3.20. SRTM – N37E029 zone results	53

Table 3.21. SRTM – N37E034 zone results	53
Table 3.22. SRTM – N38E027 zone results	53
Table 3.23. SRTM – N38E032 zone results	54
Table 3.24. SRTM – N39E041 zone results	55
Table 3.25. SRTM – N39E044 zone results	56
Table 3.26. SRTM – N41E026 zone results	56
Table 3.27. SRTM – N38E032 zone results	58
Table 3.28. SRTM – N39E041 zone results	58
Table 4.1. Average performance results of the algorithms	62

LIST OF FIGURES

Figure 1.1 a. Cracks and b. T-junctions	18
Figure 1.2 Eliminating cracks and T-junctions via recursive splitting.	18
Figure 2.1 Top down or bottom up approaches	22
Figure 2.2 (a) A regular grid terrain representation (b) TIN representation.	23
Figure 2.3 (a–d) Quadtree structure (e–h) binary triangle trees	24
Figure 2.4. Seumas McNally’s bintree tessellation code	26
Figure 2.5. The split and merge operations on a binary triangle tree.....	27
Figure 2.6 Illustrating nested ROAM wedgies for the 1D case	29
Figure 2.7 a. ROAM-simplified terrain (b) a bird’s-eye view of the terrain,.....	31
Figure 2.8 (a) The hierarchy of nested isosurfaces (b) Clustering of isosurfaces	32
Figure 2.9 (a) Textured and (b) wire frame images of Röttger CLOD	33
Figure 2.10. Triangle arrangement for a patch of terrain	34
Figure 2.11. Patch of terrain for Geomipmapping algorithm.....	35
Figure 2.12 Two patches, side by side, with different levels of detail.....	36
Figure 2.13 a screenshot from a geomipmapping implementation	37
Figure 2.14. Crack elimination by omitting rendering the vertex at points A and B.....	37
Figure 3.1 Turkey SRTM zone map	43

Figure 3.2 Calculating Terrain Accuracy.....	47
Figure 3.3. The normal to the plane is the vector (A, B, C).....	48
Figure 3.4 ROAM algorithm TerraGen – Perlin Noise data set texture map view.....	51
Figure 3.5 The Quadtree algorithm TerraGen – Ridged Perlin Method dataset view	52
Figure 3.6. The ROAM algorithm SRTM – N38E027 zone results wire-frame view	54
Figure 3.7 The Quadtree algorithm SRTM – N39E041 wire-frame view	55
Figure 3.8 The GeoMipmap algorithm SRTM – N39E044 data set wire frame view	56
Figure 3.9. Polygon count result graph.....	57
Figure 3.10. Terrain accuracy result graph	57

ABBREVIATIONS

BTT	Bintritree
CLOD	Continuous Level of Detail
CPU	Central Processing Unit
DEM	Digital Elevation Map
DTED	Digital Terrain Elevation Data
GeoTIFF	Geographic Tag(ged) Image File Format
GDC	Game Development Conference
GIS	Geographic Information Systems
GPU	Graphical Process Unit
LOD	Level of Detail
LOS	Line of Site
ROAM	Real-time Optimally Adapting Meshes
SRTM	Shuttle Radar Topography Mission
TIN	Triangulated Irregular Networks
USGS	United States Geological Survey

CHAPTER 1

INTRODUCTION

In the last decade, a fast growing domain in computer graphics is the so called Geographic Information Systems (GIS). GIS are a body of software, hardware and personal necessary for storage, modeling, analysis and reporting of positional data. The difference of GIS from other information systems is its positional analysis functions. These functions achieve the solution of real world problems by the usage of positional and non-positional data contained within the database. By this property GIS offers a powerful toolkit to decision makers for the analysis and interpretation of positional data. They allow exploring large geographic data sets interactively on screen, which involves displaying height fields in real-time.

Terrain rendering is specialized area of GIS. Terrain rendering is a statistical representation of the continuous surface of the ground by a large number of select points with known x, y and z coordinates in an arbitrary coordinate field. In the last several years, there has been a tremendous growth in the application of terrain rendering, not only in the traditional fields of geography, surveying and mapping, and earth and environmental sciences, but also in landscape design, environmental impact analysis, site selection for telecommunication facilities(radio, T.V ,and mobile telephone transmission towers), realistic flight simulators or terrain-based computer games, as well as to geographic information systems and military mission planning applications. This is partially due to the increasing availability of digital terrain data from government agencies, academic and research institutions, as same time, advancing computer technology has helped popularize the use of terrain rendering [1].

Like most subjects in computer science, the study of terrain rendering is subdivided into many problems and particular techniques. One such aspect that is very common in the above mentioned applications is the ability to efficiently render a 3D terrain in real-time. This thesis is an evaluation of three real-time continuous terrain levels of detail algorithms described in the papers ROAMing Terrain: Real-time Optimally Adapting Meshes by Duchaineau [2], Real-Time Generation of Continuous Levels of Detail for Height Fields by Röttger [3] and Fast Terrain Rendering Using Geometrical MipMapping by Willem H. de Boer [4]. The evaluation and comparison of the algorithms is based on the trade-off of polygon count to terrain accuracy over separate test data sets.

1.1 3D Terrain Overview

A 3D terrain is also called a height field because it is comprised of an $N \times N$ field of height values that make up the terrain. Height fields are usually stored as $N \times N$ grayscale Bitmap images where the color of each pixel represents the height value (0-255) for the corresponding location in the terrain. In most cases the length of one side of the height field, N , is $2^M + 1$ for some value of M . These height fields can be generated automatically or they can come in the form of Digital Elevation Model (DEM) maps that describe actual regions of the earth's surface. This thesis uses terrain height fields in the chapter 3. Rendering a 3D terrain may seem simple at first, since a height field can be triangulated into many little triangles that are easy to draw on the screen. The problem, however, lies in the fact that triangulating a terrain of size $2^M + 1 \times 2^M + 1$ results in 2^{2M+1} triangles, which become virtually impossible to render in real-time for increasingly large terrains. For this reason, most terrain rendering algorithms are real-time continuous level of detail algorithms [5].

1.2 Real-Time Continuous Level of Detail

Level of detail (LOD) rendering for any mesh (not just terrains) can be described as the process of generating a finite number of representations of the same mesh, each

at a different level of detail. Representations that have more detail would use more polygons, and the lowest detailed mesh would use the fewest polygons. At any given time an algorithm would determine which of the pre-computed mesh representations should be used based on the desired level of detail. Switching between representations usually produces a visual popping effect since more detail will suddenly appear or disappear all at once. Continuous level of detail (CLOD) is a LOD approach that can determine exactly how many polygons to use for any desired level of detail between the range of the maximum and minimum level of detail possible. CLOD algorithms eliminate the popping effect when changing the desired level of detail since the changes in polygon usage are gradual. Real-time continuous level of detail is any CLOD algorithm that renders a mesh while allowing a user to dynamically modify the desired level of detail each frame at run time. For example, a real-time continuous level of detail algorithm for height fields is one that allows a user to navigate around a terrain while continually rendering the area close to the user with a high level of detail. Regions of the terrain further and further from the user would be rendered with less and less detail, and this would be updated each frame as the user moves around dynamically[5][6].

1.3 Challenge of Terrain Rendering Algorithms

Terrain visualization is a difficult problem for applications requiring accurate images of large datasets at high frame rates, such as flight simulation and ground-based aircraft testing using synthetic sensor stimulation. Height fields play an important role in the GIS. For exploring different kinds of geographic-based data sets on screen it is necessary to display height fields at interactive frame rates. Because of the inherent geometric complexity, this goal is often unachievable even with new generations of powerful graphics computers, unless the original height field data is approximated in order to reduce the number of geometric primitives that need to be rendered without compromising visual quality.

Traditional triangle-reduction methods use a small set of discrete levels of detail that each represents the same object at different number of triangles. Simple visualization systems compute the distance from the observer to an object and choose a level of detail for the entire object based on that distance. Other visualization systems base the level of detail on the screen-space error of the object. If the distance or error changes beyond a certain limit, the whole object is rendered with another level of detail. There are two problems with triangle-reduction methods based on discrete levels of detail. First, large objects that may have some regions close to the observer and others more distant should be rendered at different levels of detail for the different regions. Terrain is an example of such an object. The horizon does not need to be rendered with as high detail as nearby parts. Second, changing from one level of detail to another leads to temporal aliasing artifacts known as popping [6] [7].

1.3.1 Tears, Cracks, and T-Junctions

A common problem when dealing with terrain LOD (particularly when dealing with quadtree- or block-based approaches) occurs when adjacent triangles exist at different levels of detail. In this situation, it is possible to introduce cracks along the edge, where the higher LOD introduces an extra vertex that does not lie on the lower LOD edge. When rendered, these cracks can cause holes in the terrain, allowing the background to peak through. Another undesirable artifact is the T-junction. This is caused when the vertex from a higher LOD triangle does not share a vertex in the adjacent lower LOD triangle. This can result in bleeding tears in the terrain due to small floating-point rounding differences, and visible lighting and interpolation differences across such edges. Figure 1.1 illustrates both of these cases.

There are a number of ways of dealing with cracks. Some of the more common solutions are described in the following. The triangles around the crack are recursively split to produce a continuous surface. This will often introduce

additional triangles into the mesh but produces a pleasing and continuous result. This approach is often used in bin tree-based systems. For example, the ROAM algorithm adopts this solution, along with other systems.

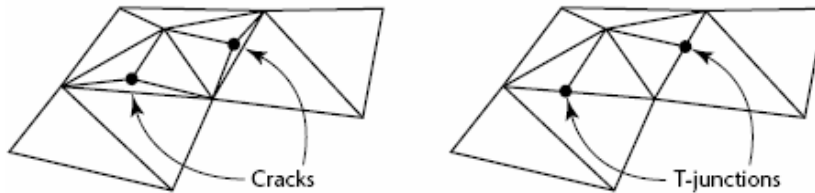


Figure 1.1 a. Cracks and b. T-junctions

Figure 1.2 gives an example of this technique. The extra vertex in the higher LOD mesh has its height modified so that it lies on the edge of the adjacent lower LOD mesh. This does not affect the number of triangles or vertices in the model, but it introduces a potentially visible error into the terrain surface unless care is taken that the vertex shift occurs below the accuracy threshold. This method essentially implies introducing a T-junction into the model, which is generally considered harmful to the continuity of the mesh, although some systems have used this approach due to its simplicity and compactness.

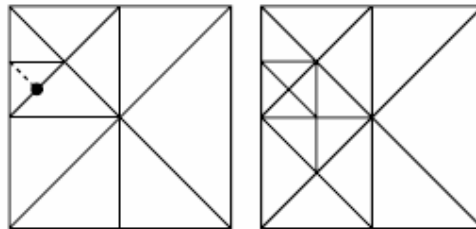


Figure 1.2 Eliminating cracks and T-junctions via recursive splitting.

A similar solution that avoids the T-junction is to simply skip the center vertex on the higher LOD mesh. An equivalent operation to the previous case is to add an

extra vertex to the edge of the lower LOD mesh and assign it the same location as the extra vertex in the higher LOD mesh. This produces a more faithful terrain surface, but at the cost of introducing an extra vertex. A further refinement of this approach is to shift both of the boundary vertices to the average of their two elevations. A common tool for managing extremely large terrain data sets is to segment them into a number of blocks, or tiles, so that these might be paged into main memory as needed. A simple solution to avoid tears between blocks is to prevent simplification of vertices that lie on the boundary of a block. A new triangle is inserted between the two meshes to plug up the gap. Although this results in a continuous mesh, the fill polygons lie in a plane perpendicular to the surface and hence can introduce unnatural-looking short cliffs. This solution also has the disadvantage of using extra polygons to represent the mesh [6].

1.4 Motivation

Terrain rendering is an exciting and challenging area of computer graphics. The prime problem of terrain rendering is simply size of data. Although, graphics hardware continues to evolve at an astounding rate and have the ability to draw a huge amount of polygons in real-time, clever algorithms for reducing the geometry to be rendered and for sending it to the hardware as quickly as possible will always be necessary. These terrain rendering algorithms are trying to decrease the amount of data. Performing CLOD on a mesh is generally done to reduce the number of polygons needed to render a scene. However, this reduction comes at a cost. When dealing with 3D terrains, refining the terrain to save on polygon usage results in a loss in terrain accuracy. The purpose of this thesis is to examine the trade-off between terrain accuracy and polygon count for the three algorithms being studied. Although the focus of this project is from a GIS point of view, other applications involving 3D terrain may consider this study a benefit. Finally, each algorithm will be discussed in terms of reach to terrain accuracy with using minimum polygon

count and usability for the different types of terrain data and we also try to find the best suitable algorithm for terrain rendering.

The main contribution of this thesis is evaluating algorithms with GIS point of view. We try to find the best algorithm for terrain rendering with using real world data. We already mention about the main problem of terrain rendering; very large data. The modern graphics cards will be able to display more and more polygons per second, but on the other hand the resolution of elevation data is also increasing day by day. For example, a 3 arc second (approx. 90m resolution) DEM data used in this thesis. But, a 1 arc second (less than 16m resolution) data product was also produced, but it is not available for public usage. In this case, the vertex count naturally increases for terrain rendering. Therefore, rendering terrain with minimum polygon is important criteria for GIS application. In according to our experimental results, the ROAM algorithm easily copes with this problem (See Chapter 3).

1.5 Outline of Thesis

In Chapter 2 discusses the requirements of terrain-rendering algorithms and surveys existing algorithms. We define test data sets and test cases at the Chapter 3. Also we discuss our test results in this chapter. Chapter 4 concludes this position thesis.

CHAPTER 2

BACKGROUND ON TERRAIN RENDERING ALGORITHMS

In this chapter, we described many of the fundamental concepts of terrain LOD. After that we explain the terrain rendering algorithms that are using in our thesis.

2.1 Multiresolution Techniques for Terrain

We begin this chapter by taking a look at some of the principal variables a developer faces when implementing a terrain LOD algorithm.

2.1.1 Top Down and Bottom Up

One of the major differentiators of terrain LOD algorithms, as with more general LOD techniques, is whether they are top-down or bottom-up in their approach to the simplification problem. In a top-down algorithm, we normally begin with two or four triangles for the entire region and then progressively add new triangles until the desired resolution is achieved. These techniques are also referred to as subdivision or refinement methods. In contrast, a bottom-up algorithm begins with the highest-resolution mesh and iteratively removes vertices from the triangulation until the desired level of simplification is gained. These techniques can also be referred to as decimation or simplification methods. Figure 2.1 illustrates these two approaches to terrain simplification. Bottom-up approaches tend to be able to find the minimal number of triangles required for a given accuracy. However, they necessitate the entire model being available at the first step and therefore have higher memory and computational demands.

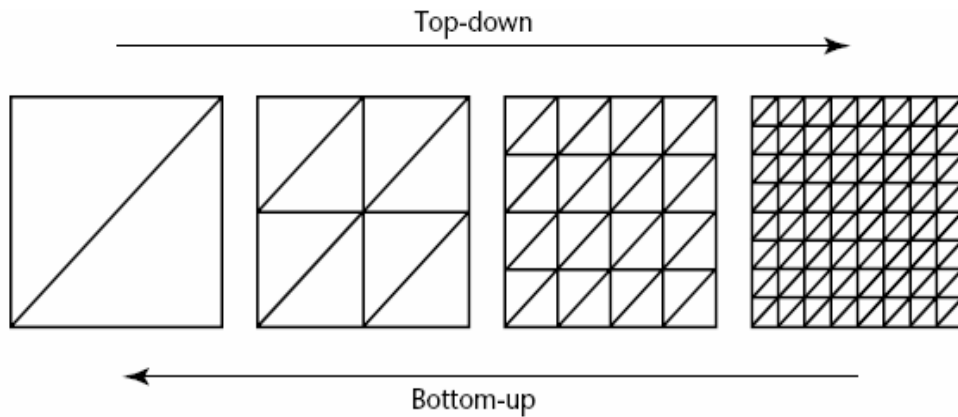


Figure 2.1 Top down or bottom up approaches

It is worth making an explicit distinction between the preparation-time and runtime sense of bottom-up versus top-down. Bottom-up approaches are almost always used during the initial offline hierarchy construction. However, at run-time, a top down approach might be favored because, for example, it offers support for view culling. Most interactive bottom-up solutions are usually hybrid in practice, often combined with a top-down quadtree block framework. Furthermore, a few systems perform incremental coarsening or refining of the terrain at each frame to take advantage of frame-to-frame coherency. As such, these systems cannot strictly be classified as exclusively top-down or bottom-up.

2.1.2 Regular Grids and TINs

Another important distinction between terrain LOD algorithms is the structure used to represent the terrain. Two major approaches in this regard are the use of regular gridded height fields and Triangulated Irregular Networks (TINs). Regular (or uniform) grids use an array of height values at regularly spaced x and y coordinates, whereas TINs allow variable spacing between vertices. Figure 2.2 illustrates these two approaches, showing a regular grid of 65×65 (equals 4,225) height values and a 512- vertex TIN representation with the same accuracy. TINs can generally

approximate a surface to a required accuracy with fewer polygons than other schemes. For example, they allow large flat regions to be represented with a coarse sampling, while reserving higher sampling for more bumpy regions. Regular grids, in comparison, tend to be far less optimal than TINs because the same resolution is used across the entire terrain, at flat places as well as high-curvature regions. TINs also offer great flexibility in the range and accuracy of features that can be modeled, such as maxima, minima, saddle points, ridges, valleys, coastlines, overhangs, and caves. However, regular grids offer the advantages that they are simple to store and manipulate. For example, finding the elevation at any point is a simple matter of bilinearly interpolating the four nearest neighbor points. They are easily integrated with raster databases and file formats, such as the DEM, DTED, and GeoTIFF file formats. In addition, they require less storage for the same number of points because only an array of z values needs to be stored rather than full (x, y, z) coordinates.

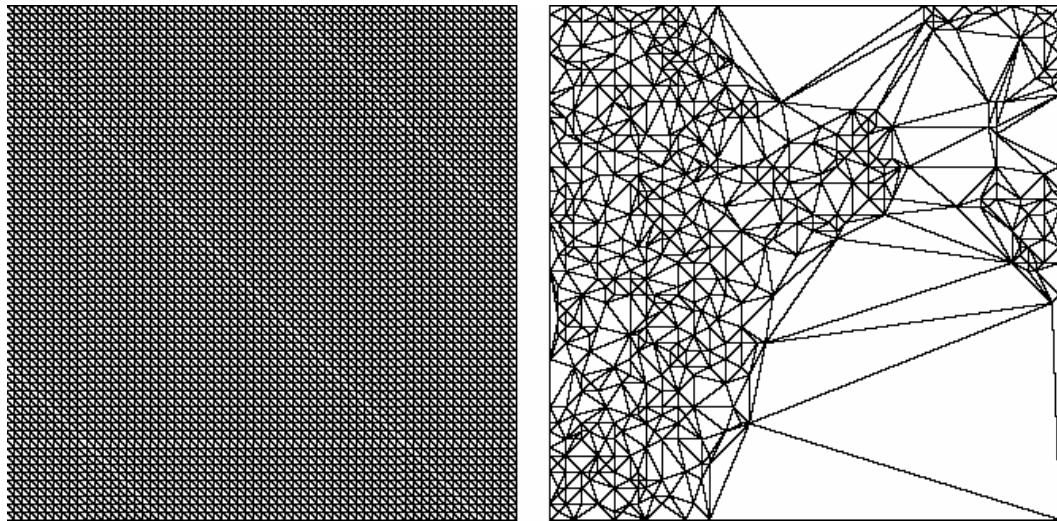


Figure 2.2 (a) A regular grid terrain representation (b) TIN representation.

Furthermore, TINs make implementing related functions (such as view culling, terrain following, collision detection, and dynamic deformations) more complex

because of the lack of a simple overarching spatial organization. Also, the applicability of TINs to run-time view-dependent LOD is less efficient than regular gridded systems. For these reasons, many contemporary terrain LOD systems favor regular grids over TINs. It is worth noting that a number of hybrid schemes have been proposed that try to gain the best of both worlds—most notably by using a hierarchical triangulation based on a regular grid.

2.1.3 Quadtrees and Bintrees

To implement view-dependent LOD for a regular grid structure, we must be able to represent different parts of the grid at different resolutions. This implies a hierarchical representation in which we can gradually refine further detail to different parts of the grid. There are a number of options available for achieving this multiresolution representation. The most common two are the quadtree and the binary triangle tree. A quadtree structure is where a rectangular region is divided uniformly into four quadrants. Each of these quadrants can then be successively divided into four smaller regions, and so on (See Figure 2.3 (a–d)). Quadtrees have been used for a number of terrain LOD systems.

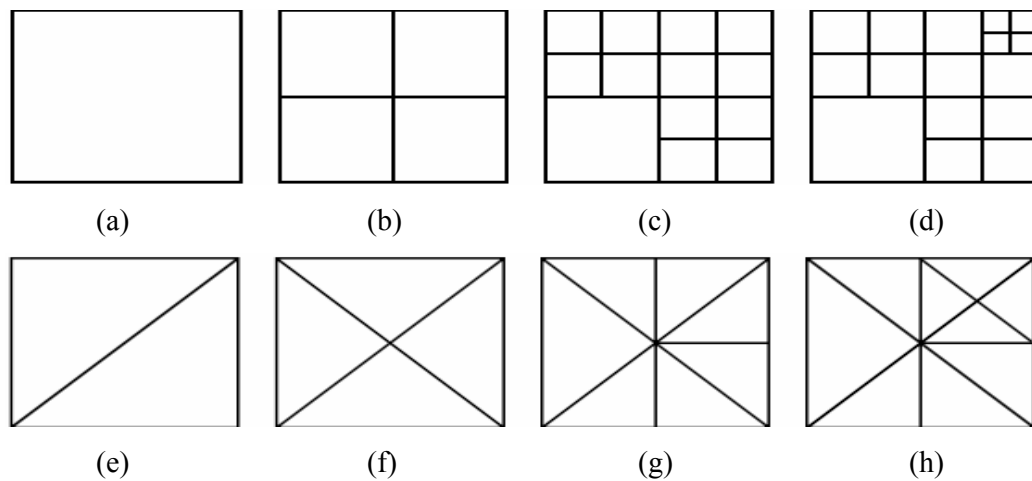


Figure 2.3 (a–d) Quadtree structure (e–h) binary triangle trees.

Note that you can still employ a quadtree structure and use triangles as your primitives. You would simply decompose each rectangle into two or more triangles. In fact, a number of different triangulation schemes could be implemented independent of the use of quadtrees or bintrees. A binary triangle tree structure (bintrintree, BTT, or simply bintrees) works the same way as a quadtree, but instead of segmenting a rectangle into four it segments a triangle into two halves. The root triangle is normally defined to be a right-isosceles triangle (i.e., two of the three sides are equal and they join at a 90-degree angle), and the subdivision is performed by splitting this along the edge formed between its apex vertex and the midpoint of its base edge (See Figure 2.3 (e–h)). Note that another, more general; term that can be used to describe a bintree is a kd-tree. A kd-tree is a binary tree that recursively subdivides a space such that a k-dimensional kd-tree divides a k-dimensional space with a $(k - 1)$ -dimensional plane. Systems that have implemented binary triangle tree techniques include Lindstrom [8] et al. and Duchaineau [2] et al. One of the big advantages of bintrees is that they make it easy to avoid cracks and T-junctions. Bintrees also exhibit the useful feature that triangles are never more than one resolution level away from their neighbors (this is not true for quadtrees, which often require extra care to preserve this condition). Seumas Mc-Nally wrote an excellent piece on bintrees for GameDev.net [9]. In that article he presents psuedo code for splitting a triangle in a binary triangle tree while avoiding cracks and T-junctions. The psuedo code follows, including some minor optimizations that have been reported recently. In this code, the left and right neighbors point to the triangles on the left and right with the hypotenuse down, and the bottom neighbor is the triangle that meets the hypotenuse. Figure 2.4 illustrates the split progression.

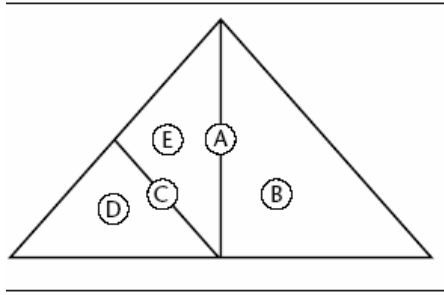


Figure 2.4. Seumas McNally's bintree tessellation code

2.2 Terrain Rendering Algorithms

We describe main features of the each algorithms briefly in this section.

2.2.1 The ROAM Algorithm

In 1997, Duchaineau published the ROAM algorithm [2]. This has proved to be an extremely popular algorithm, particularly among game developers; it has been implemented for the Tread Marks, Genesis3D, and Crystal Space engines, among others. ROAM (Real-time Optimally Adapting Meshes) uses an incremental priority-based approach with a binary triangle tree structure. A continuous mesh is produced using this structure by applying a series of split and merge operations on triangle pairs that share their hypotenuses, referred to as diamonds (See Figure 2.5).

The ROAM algorithm uses two priority queues to drive split and merge operations. One queue maintains a priority-ordered list of triangle splits so that refining the terrain simply means repeatedly splitting the highest-priority triangle on the queue. The second queue maintains a priority-ordered list of triangle merge operations to simplify the terrain. This allows ROAM to take advantage of frame coherence (i.e., to pick up from the previous frames triangulation and incrementally add or remove

triangles). Duchaineau et al. also note that splits and merges can be performed smoothly by geomorphing the vertex positions during the changes.

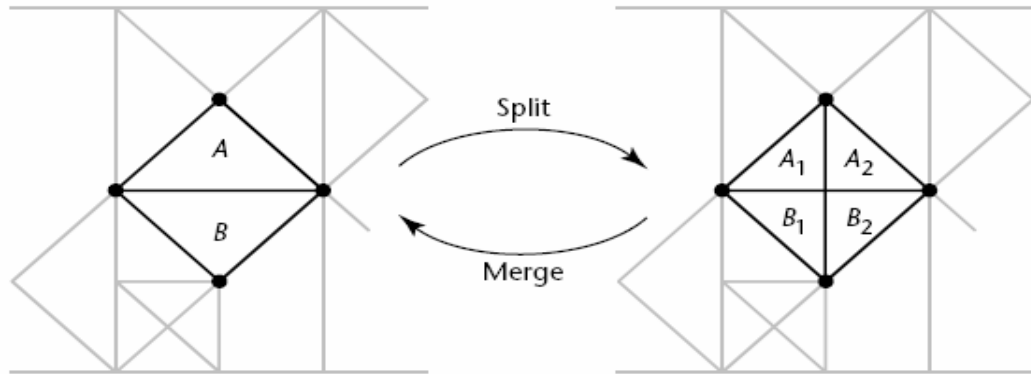


Figure 2.5. The split and merge operations on a binary triangle tree.

The priority of splits and merges in the two queues was determined using a number of error metrics. The principal metric was a screen-based geometric error that provides a guaranteed bound on the error. This was done using a hierarchy of bounding volumes, called wedgies, around each triangle (similar to the notion of simplification envelopes). A wedgie covers the (x, y) extent of a triangle and extends over a height range $z - e_T$ through $z + e_T$, where z is the height of the triangle at each point and e_T is the wedgie thickness, all in world-space coordinates. A preprocessing step is performed to calculate appropriate wedgies that are tightly nested throughout the triangle hierarchy, thus providing a guaranteed error bound (See Figure 2.6).

At run-time, each triangle's wedgie is projected into screen space and the bound is defined as the maximum length of the projected thickness segments for all points in the triangle (note that under the perspective projection, the maximum projected thickness may not necessarily occur at one of the triangle vertices). This bound is used to form queue priorities, and could potentially incorporate a number of other metrics, such as back face detail reduction, silhouette preservation, and specular highlight preservation.

The ROAM algorithm includes a number of other interesting features and optimizations, including an incremental mechanism to build triangle strips. Modern graphics processing units often provide significant performance gains when triangles are organized into strips. In the original ROAM algorithm, strip lengths of four to five triangles were favored. These strips were incrementally adjusted as triangles were split or merged. The authors report a significant frame time improvement of 72 ms per frame by using triangle strips. Another intriguing feature that was supported was line-of-site (LOS) based refinement. In this case the triangulation is made more accurate along a specified line of sight so that correct visibility and occlusion determinations can be made. This is particularly useful for military mission planners and ground-based aircraft testing using synthetic sensor stimulation. Another optimization defers the computation of triangle priorities until they potentially affect a split or merge decision. The authors report that this priority recomputation deferral saved them 38 ms per frame. Finally, the ROAM algorithm can also work toward an exact specified triangle count, as well as support fixed frame rate constraints.

Duchaineau [2] et al. tested their implementation with a United States Geological Survey (USGS) 1-degree DEM for Northern New Mexico (about $1,200 \times 1,200$ postings at 3-arc-second, or roughly 90-m, resolution). They report that on a R10000 Indigo2 workstation they achieved 3,000 triangles within a rate time of 30 ms (5 ms for view-frustum culling, 5 ms for priority queue calculation, 5 ms for split/merge operations, and 15 ms to output the triangle strips). In terms of frame coherence, the authors found that on average less than 3% of triangles changed between frames. Figure 2.7 shows an example of a ROAM-simplified mesh.

The original ROAM algorithm has been improved or modified by a number of researchers and game developers [10] [11] [12] [13] [14]. For example, one simplification sometimes used by game developers is to discard the frame coherence

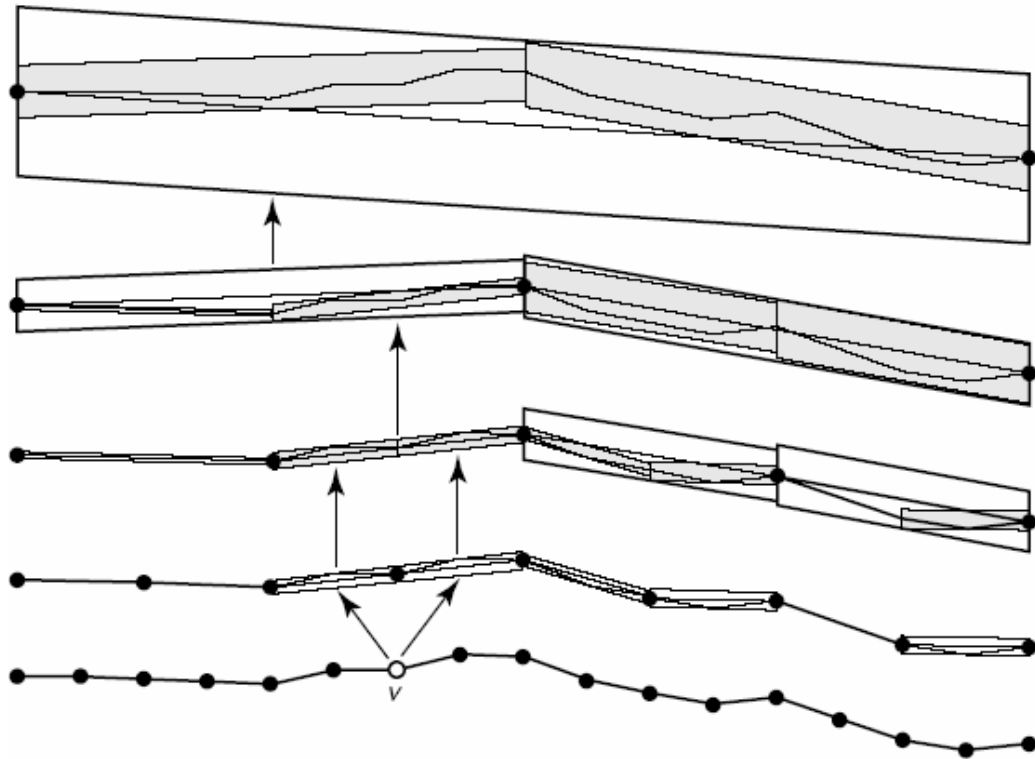


Figure 2.6 Illustrating nested ROAM wedgies for the 1D case

feature, resulting in a “split-only ROAM” implementation (such as that described by Bryan Turner in his Gamasutra. com article “Real-Time Dynamic Level of Detail Terrain Rendering with ROAM” [15]).

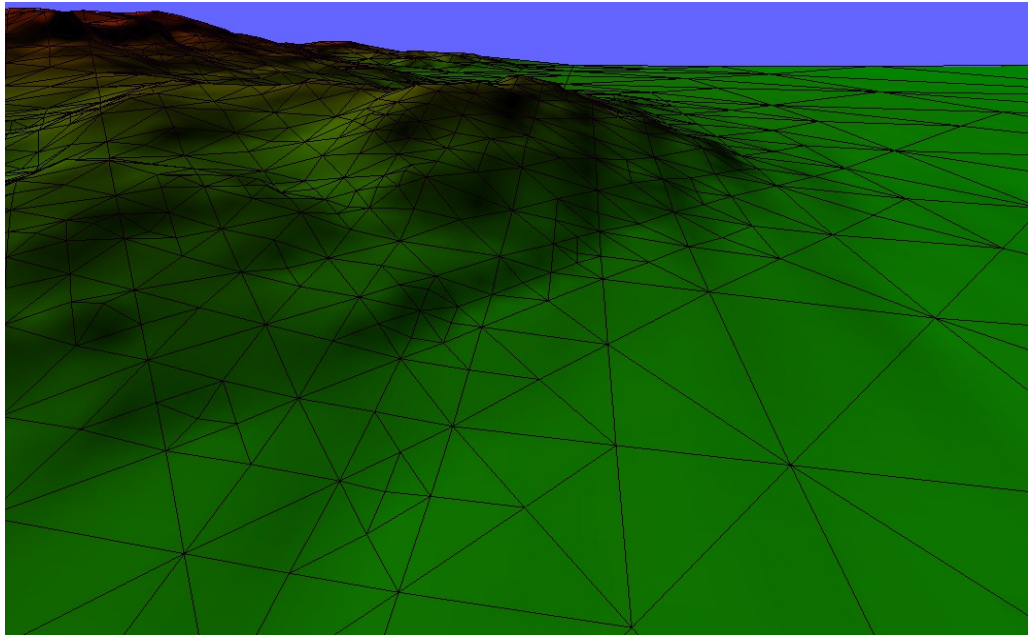
One noteworthy improvement of the original algorithm was provided by Jonathan Blow at the GDC 2000 conference [16]. Blow found that the original ROAM algorithm does not perform well for densely sampled data, and attributed this to the large number of premature recalculations of wedgie priorities that can occur in a well-tessellated high-detail terrain. Blow noted that both Lindstrom and Duchaineau used screen-space error metrics that compressed the 3D geometric error down to a

1D scalar value. Instead, Blow advocated using the full three dimensions of the source data to perform LOD computations and building a hierarchy of 3D isosurfaces to contain all vertices within a certain error bound. (It should be noted that this is simply another way to look at the error function and that Lindstrom et al. also illustrated their error function as a 3D isosurface.) For simplicity, Blow chose spheres as the isosurface primitive, such that each wedgie was represented by a sphere in 3D space. When the viewpoint intersects with the sphere, the wedgie is split, and when the viewpoint leaves a sphere, the wedge is merged. To optimize this process, a hierarchy of nested spheres was used and the algorithm only descends into nodes when the viewpoint intersects a sphere. In addition, spheres could be clustered at any level by introducing extra bounding volumes to provide further resilience to large terrain models (See Figure 2.8). Blow noted that this new error metric produced extremely efficient split and merge determinations for high-detail terrain in cases for which the original ROAM algorithm would stutter visibly. For example, at 640×480 resolution with a 3-pixel error threshold, Blow's approach produced a tessellation with 65% less triangles than their ROAM implementation.

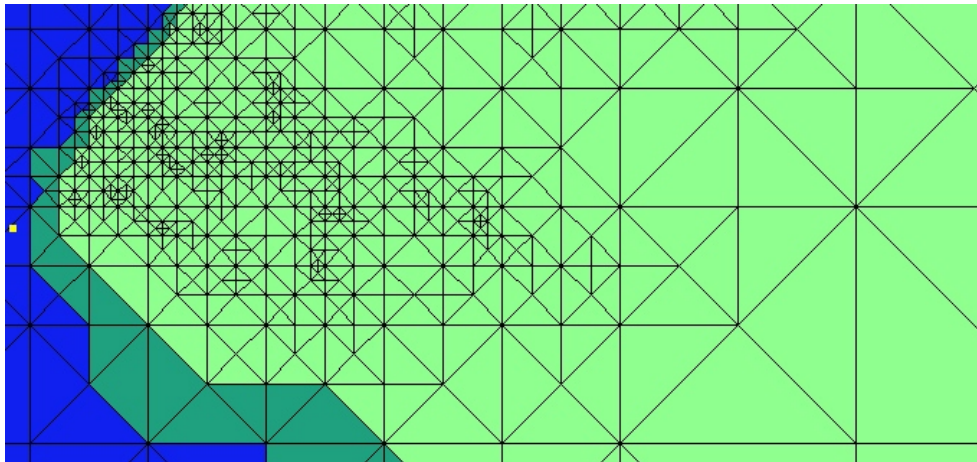
2.2.2 Real-Time Generation of Continuous LOD

In 1998, Röttger [2] et al. extended the earlier continuous LOD work of Lindstrom [8] et al. Instead of adopting a bottom-up approach, they chose a top-down strategy, noting that this meant their algorithm needed to visit only a fraction of the entire data set at each frame, but that this also made the addition of features such as silhouette testing problematic because these would require analysis of the entire data set.

They used a quadtree data structure rather than a binary triangle tree, and dealt with tears between adjacent levels of the quadtree by skipping the center vertex of the higher-resolution edge. To simplify this solution, Röttger et al. implemented a bottom-up process from the smallest existing block to guarantee that the level



(a)



(b)

Figure 2.7 a. ROAM-simplified terrain (b) a bird's-eye view of the terrain,

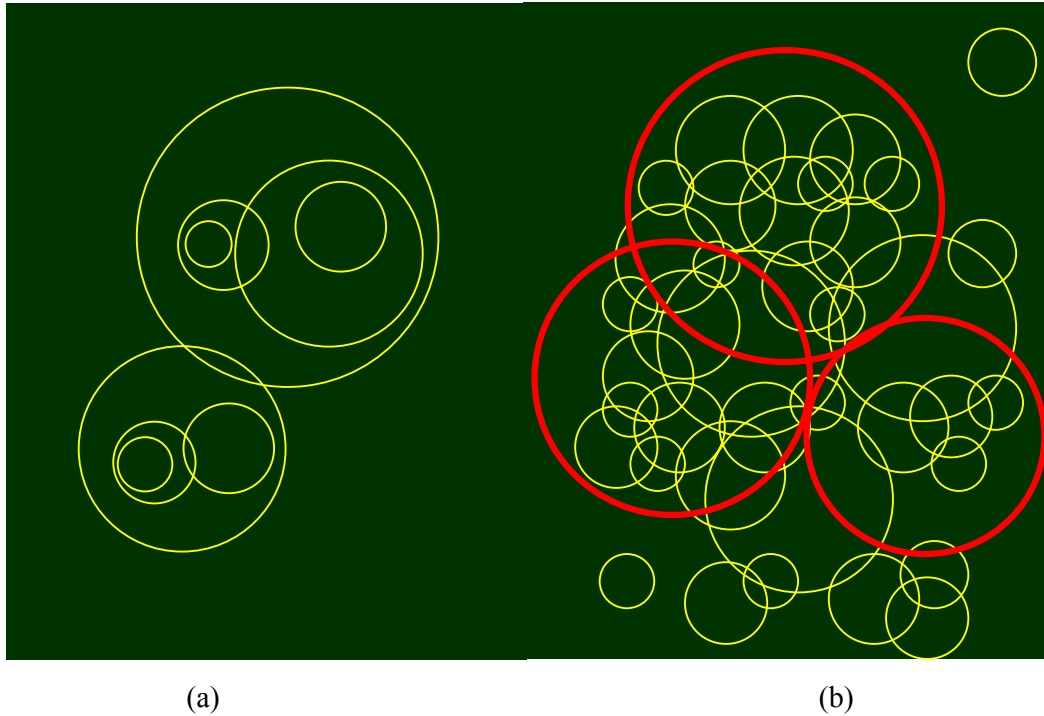


Figure 2.8 (a) The hierarchy of nested isosurfaces (b) Clustering of isosurfaces

difference between adjacent blocks did not exceed 1. They also introduced a new error metric that took into consideration the distance from the viewer and the roughness of the terrain in world space. Their metric can be written as follows (See Equation 2.1).

$$f = \frac{l}{d.C.\max(c.d2,1)} \quad (2.1)$$

Here, l is the distance to the viewpoint (Manhattan distance was used for efficiency), d is the edge length of a quadtree block, C is a configurable quality parameter that determines the minimum global resolution (a value of 8 was found to provide good visual results), and c specifies the desired global resolution that can be adjusted per

frame to maintain a fixed frame rate. The quantity d_2 incorporates the surface roughness criteria by representing the largest error delta value at six points in the quadtree: the four edge midpoints and the two diagonal midpoints. An upper bound on this component was computed by taking the maximum of these six absolute delta values. An important feature of Röttger et al.'s system is its direct support for geomorphing of vertices to smooth the transition between levels of detail. This was implemented by introducing a blending function, $b = 2(1 - f)$, clamped to the range $[0, 1]$ to morph vertices linearly between two levels of detail. Extra care was taken to avoid cracks that could occur during geomorphing due to adjacent blocks having different blending functions. This was done by using the minimum blending value for edges that were shared between quadtree blocks. The authors state that they were able to associate a single blending value and d_2 -value with each vertex using only one extra byte of storage. Their implementation was evaluated on an SGI Maximum Impact using a terrain model of a region in Yukon Territory, Canada. The c value was dynamically chosen to maintain a frame rate of 25 Hz, which produced roughly 1,600 triangle fans and 15,000 vertices per frame (See Figure 2.9) [17] [18].

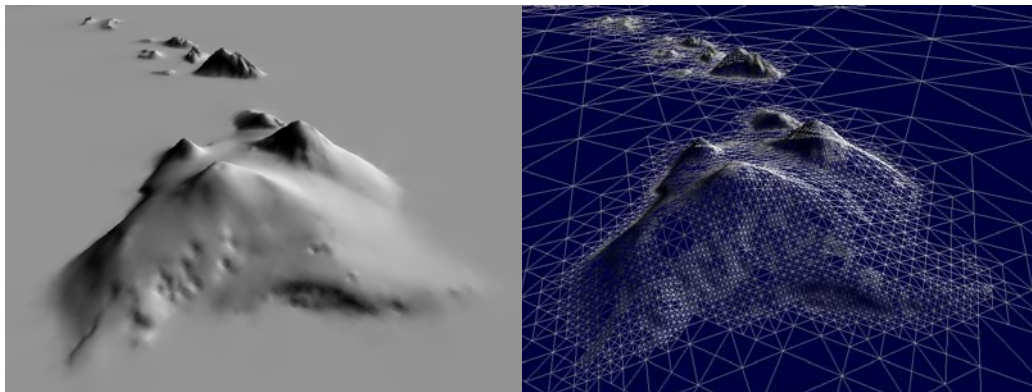


Figure 2.9 (a) Textured and (b) wire frame images of Röttger CLOD

2.2.3 Fast Terrain Rendering Using Geometrical MipMapping

The Geometrical MipMapping algorithm that is highly optimized for modern graphics cards was recently introduced by Willem H. de Boer [4]. This method divides the height-map into smaller tiles and creates a number of detail levels for each tile. Based on an approximated screen-space error, a switch between the different detail levels is made. When switching between detail levels a sudden change in the height-map (vertex popping) will occur, this will be noticeable to the viewer [19].

If you are familiar with the texturing concept of mipmapping, then geomipmapping should seem like familiar ground to you. The concepts are the same, except that instead of dealing with textures, we're dealing with vertices of a patch of terrain. The driving concept of geomipmapping is that you have a set patch of terrain. For this explanation, I'll say it's a patch with a size of 5 vertices (a 5×5 patch). That 5×5 patch is going to have several levels of detail, with level 0 being the most detailed and, in this case, level 2 being the least detailed. Look at Figure 2.10 if you need a visual explanation of what each patch looks like at its various levels. In the figure, black vertices are not sent to the rendering API, but the white ones are.

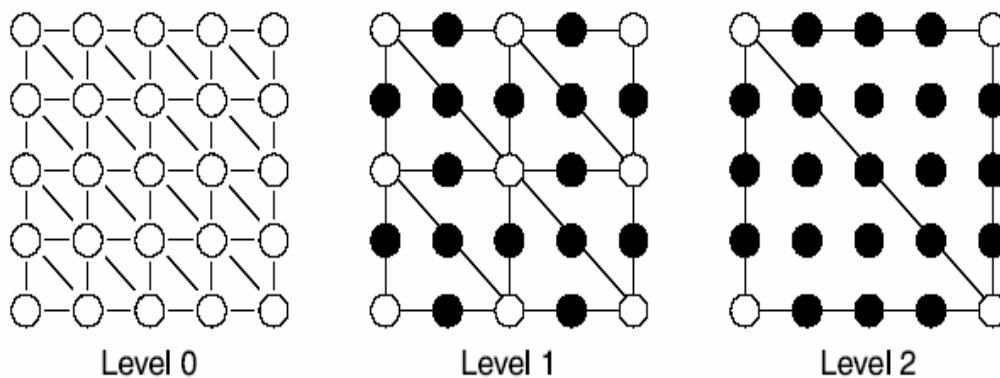


Figure 2.10. Triangle arrangement for a patch of terrain

The geomipmapping is similar to texture mipmapping except that we're using land patches instead of texture patches. What we need to do, starting from the user's point in 3D space (the camera's eye position), is make all of the patches around the viewer be the most detailed because those patches are what the user sees the most of. At a certain distance away, we'll switch to a lower level of patch detail. And, at another distance away, we'll switch to an even lower level of detail. Figure 2.11 explains this visually. As you can see in the figure, the patches in the immediate area of the viewer's position have a Level of Detail (LOD) of 0, which means that those patches are of the highest level of detail. As the patches become farther away, they change to a level of 1, which is the second highest level of detail. And even farther away from the viewer, the patches have a level of 2, which is the lowest level of detail presented in the image.

Often when you're dealing with CLOD terrain algorithms, you must deal with the subject of cracking. Cracking occurs, in the case of geomipmapping, when a highly detailed patch resides next to a lower detailed patch (See Figure 2.12). As you can see from the figure, the patch on the left is of a higher level of detail than the patch on the right. Our problem lies at points A and B. The problem is that there is a higher

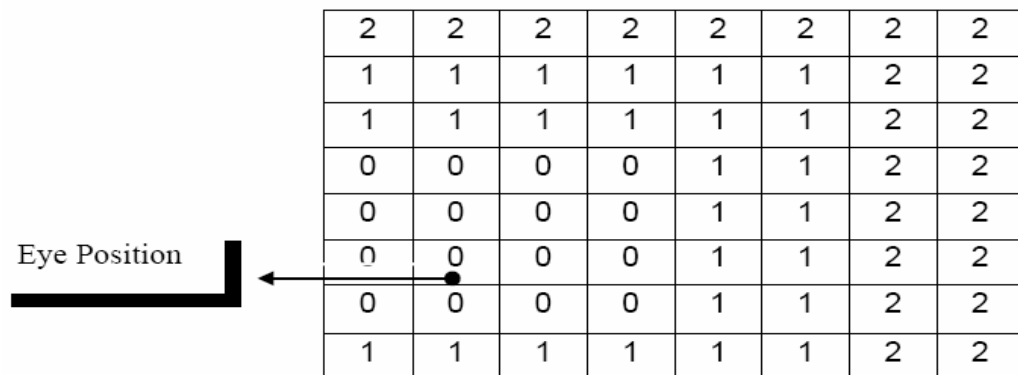


Figure 2.11. Patch of terrain for Geomipmapping algorithm

level of detail on the left side of point A than there is on point B. This means that the left patch is rendering the exact height at point A, but the right patch is just getting the average of the height above it and the height below it. This whole “cracking” thing might not seem like such a big deal, but check out Figure 2.13, which shows a screenshot of geomipmapping implementation without non cracking measures taken.

Crack-proofing your geomipmapping engine is a lot easier than it might sound. You have the added benefit of having someone explain this concept to you, which makes the whole process as easy as well, something easy. We have two possible ways of fixing the cracking problem. One way is to add vertices to the patch with the lower amount of detail so that the vertices in question will be of the same height as the higher detailed patch’s corresponding vertices. This solution could be ugly, though, and it means that we’d have to do some rearranging of the patch (add another triangle fan). The other way of solving this problem is to omit vertices from the more detailed patch. This solves the cracking problem seamlessly and effortlessly. Check out Figure 2.14 to see how easy it is to simply omit a vertex and fix the crack. Figure 2.13 a screenshot from a geomipmapping implementation, which does not implement anti-cracking measures [20].

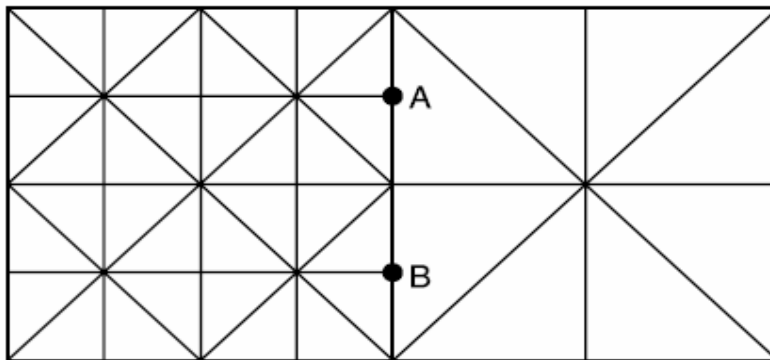


Figure 2.12 Two patches, side by side, with different levels of detail.

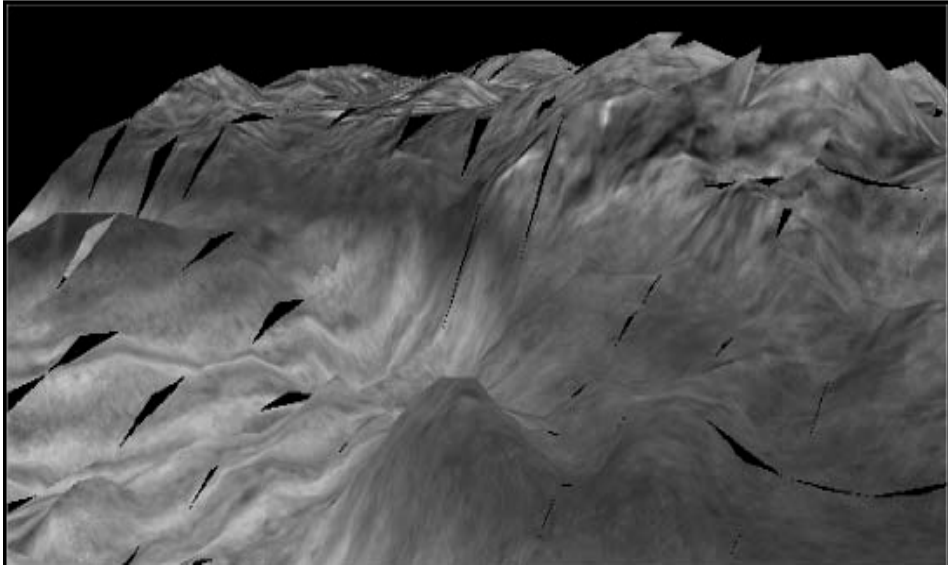


Figure 2.13 a screenshot from a geomipmapping implementation

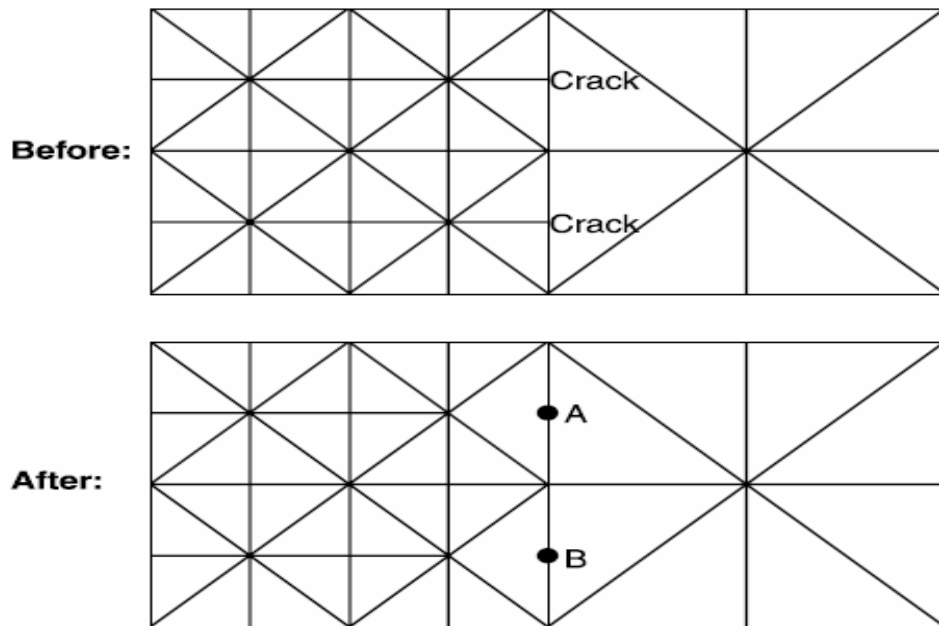


Figure 2.14. Crack elimination by omitting rendering the vertex at points A and B.

CHAPTER 3

EVALUATING ALGORITHMS

This chapter describes the methods and outcomes of evaluating the three algorithms for terrain level of detail that were studied in this thesis. Testing method is inspired from Derek Bradley [5] excellent work. The first two sections describe the testing environment and the height fields that were used for test data. The third section outlines the criteria for which the algorithms were evaluated, and finally the last section shows the actual test cases and results.

3.1 Testing Environment

The algorithms that were implemented and studied in this project were evaluated on a single machine with the specifications listed in Table 3.1.

Table 3.1 Test Machine Specifications

Processor:	Intel Pentium 4 CPU 2.40 GHz
Memory:	768 Mb DDR RAM
Video Card:	NVIDIA GeForce2 FX 5200
OS:	Windows XP Professional

Although machine specifications usually play a large role in the results of most 3D graphics tests, the evaluation of the algorithms in this project focuses on two areas that yield the same results independent of the processing power, hardware manufacturers or amount of memory in the test machine. These evaluation parameters are the polygon count and the terrain accuracy. Of course, the frame rate of the rendered scenes will vary significantly based on the machine that the software is running on, but frame rate comparisons were not part of the algorithm evaluation in this project.

Despite the fact that the hardware of the machine has no effect on the outcome of the algorithm evaluation (provided that any 3D accelerated graphics card does exist in the machine), there are two software restrictions in order to execute the accompanying software that contains the implementations for this project. These restrictions are that the machine must be running a sufficiently recent version of Microsoft Windows (the software was tested on versions 2000 and XP), and the machine must have OpenGL libraries available (which generally come by default on such versions of Microsoft Windows).

3.2 Test Data

To perform the evaluation of the algorithms in this thesis, twelve test data sets were used. Each data set represents a separate terrain with unique features in order to represent a broad range of different height fields. Of the twelve height fields used, five were generated in a shareware terrain generating application called TerraGen (version 0.9.19 by Planetside Software). The other seven terrains are real Shuttle Radar Topography Mission (SRTM) data of the Turkey. These real SRTM data's are converting to appropriate format (8 bit grayscale raw data) with using 3DEM (version 18.7 by Visualization Software) and TerraGen.

The twelve test terrains will now be described in Table 3.2 through Table 3.13. Terrain descriptions include the terrain title, size in pixels, topographic features, and the grayscale bitmap representation of the height field and a color bitmap of the texture used when rendering the terrain.

3.2.1 TerraGen Test Dataset

TerraGen is the very useful program for random terrain data generation. We use this program for obtain different kind of terrain data. TerraGen has four methods that are provide different featured terrain data.

Table 3.2 TerraGen - Subdivide & Displace Method terrain data.



Source	TerraGen - Subdivide & Displace Method	
Size in pixels	257 x 257	
Features	This method generates the data that is similar to real world.	
		
Height Map	Texture Map	

Table 3.3 TerraGen - Perlin Noise Method terrain data

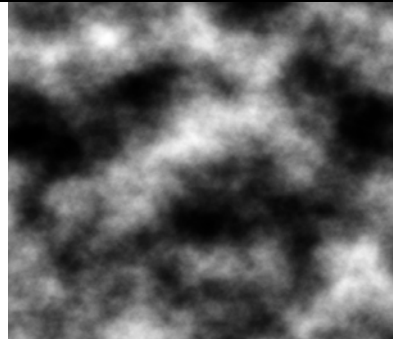
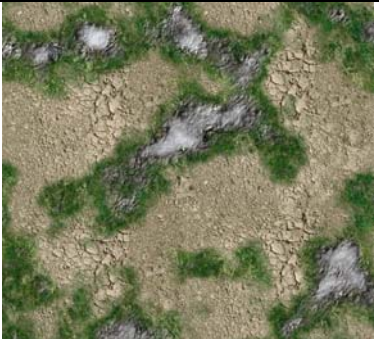
Source	TerraGen - Perlin Noise Method	
Size in pixels	513 x 513	
Features	This method creates data that is contains more flat terrain than the mountains.	
		
Height Map	Texture Map	

Table 3.4 TerraGen – Multi Perlin Method terrain data

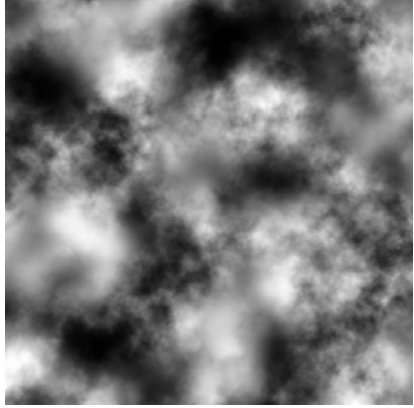
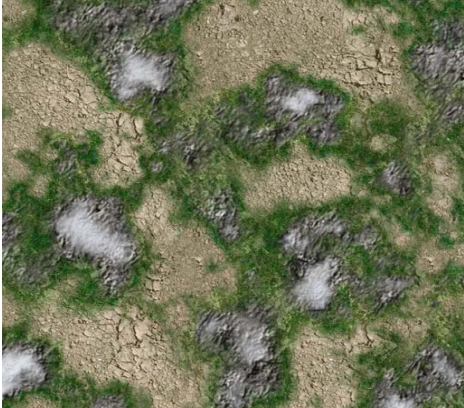
Source	TerraGen – Multi Perlin Method
Size in pixels	513 x 513
Features	This method is similar to Perlin Noise method but mountains close to each other.
	
Height Map	Texture Map

Table 3.5 TerraGen – Ridged Perlin Method terrain data

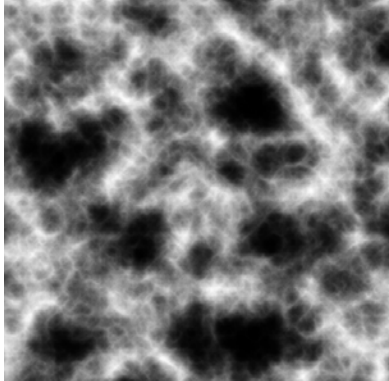
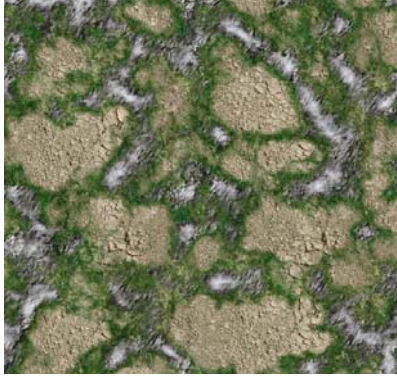
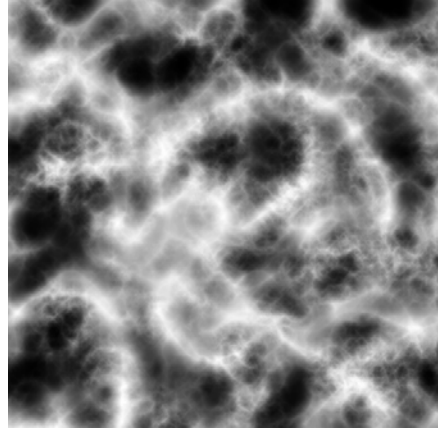
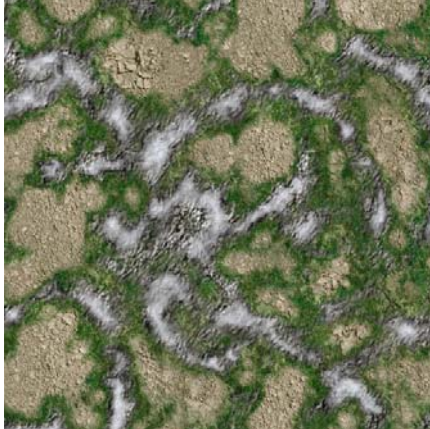
Source	TerraGen – Ridged Perlin Method
Size in pixels	513 x 513
Features	This method is opposites of Perlin Noise methods. It is increase the mountains percentage and decreases the flat terrain percentage.
	
Height Map	Texture Map

Table 3.6 TerraGen – Ridged Multi Perlin Method terrain data

Source	TerraGen – Ridged Multi Perlin Method
Size in pixels	513 x 513
Features	This method is similar to Ridged Perlin method but mountains close to each other
	
Height Map	Texture Map

3.2.2 Shuttle Radar Topography Mission (SRTM) Test Dataset

The Shuttle Radar Topography Mission (SRTM) is a joint project between the National Geospatial-Intelligence Agency (NGA) and the National Aeronautics and Space Administration (NASA). The NASA has provided Digital Elevation Model (DEM) data for over 80% of the globe. This data is currently distributed free of charge by USGS and is available for download from the National Map Seamless Data Distribution System web site (<http://seamless.usgs.gov/>) or the USGS ftp site (<ftp://e0mss21u.ecs.nasa.gov/srtm/Eurasia/>). The SRTM data is available as 3 arc second (approx. 90m resolution) DEM. A 1 arc second data product was also produced, but is not available for all countries. The vertical error of the DEM's is reported to be less than 16m. We have selected the seven SRTM dataset from Turkey map. The selected zones are mark by red frames at Figure 3.1.

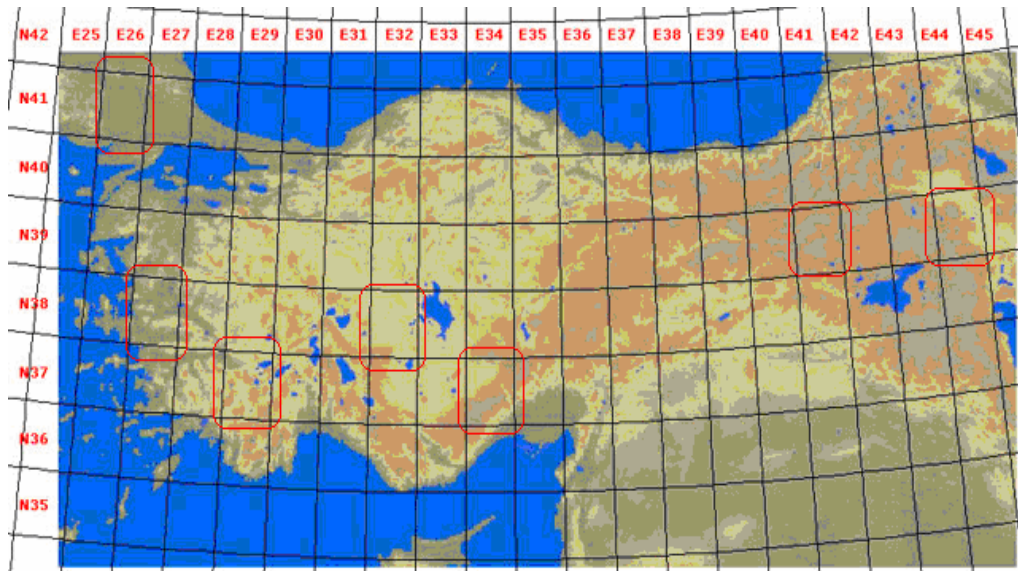


Figure 3.1 Turkey SRTM zone map

Table 3.7 N37E029 zone SRTM data

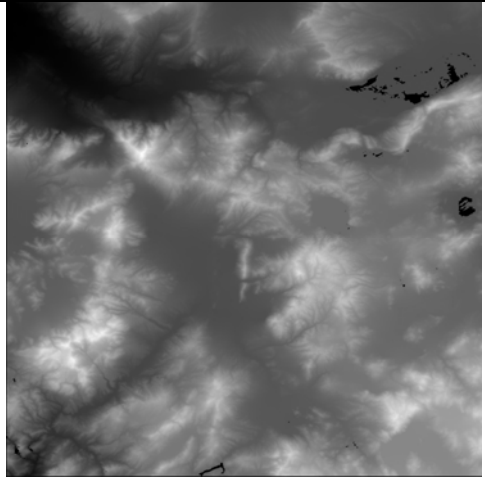

Source	N37E029 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large flat terrain and some hills. (around the Denizli territory)
	
Height Map	Texture Map

Table 3.8 N37E034 zone SRTM data



Source	N37E034 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large terrain with high mountains. (around the Niğde territory)
	
Height Map	Texture Map

Table 3.9 N38E027 zone SRTM data

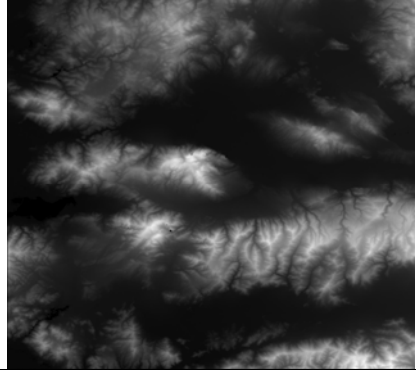

Source	N38E027 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large terrain with many jagged hills and rough areas; in other words, lots of elevation changes. (between the İzmir and Manisa territory)
	
Height Map	Texture Map

Table 3.10 N38E032 zone SRTM data

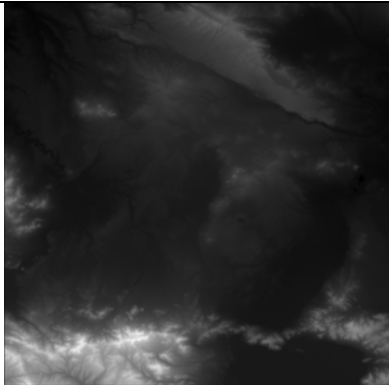

Source	N38E032 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large flat terrain with small height changes. (around the Konya territory)
	
Height Map	Texture Map

Table 3.11 N39E041 zone SRTM data

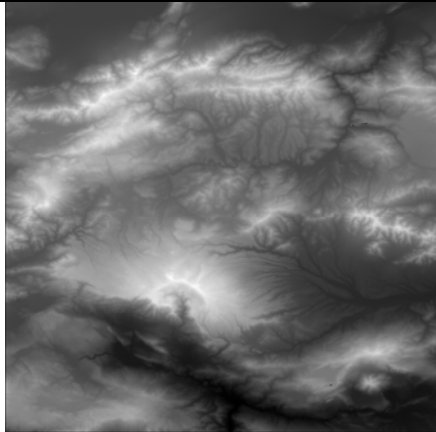

Source	N39E041 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large terrain with many jagged and high mountains. (around the Erzurum territory)
	
Height Map	Texture Map

Table 3.12 N39E044 zone SRTM data



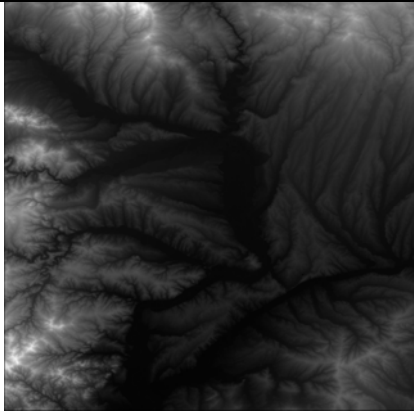

Source	N39E044 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large flat terrain with very high mountain. (around the Ağrı territory)
	
Height Map	Texture Map

Table 3.13 N41E026 zone SRTM data

Source	N41E026 zone SRTM data
Size in pixels	1025 x 1025
Features	This is a large flat terrain with no big elevation changes.(around the Edirne territory)
	
Height Map	Texture Map

3.3 Evaluation Criteria

In order to determine the usefulness of a terrain level of detail algorithm there are many factors that could be taken into account. For instance, from a 3D game point of view the algorithms may be evaluated based on speed and memory footprint, but from a GIS point of view the evaluation might be based on the accuracy of the approximated terrain and the number of polygons required to draw the terrain. This project takes on more of a GIS perspective and will evaluate the terrain accuracy and polygon count. It is for this reason that seven of the test data sets are actual SRTM.

3.3.1 Terrain Accuracy

Terrain accuracy is a measure of how close the approximated terrain resembles the original height field. This measure is calculated using the vertical distance between corresponding points in the rendered terrain and the height field. Now obviously the rendered terrain has far fewer data points than the height field, so geometric computations are performed to determine the corresponding point on the rendered surface for each point in the height field. Fig. 3.2 demonstrates the vertical distance calculation for two regions of terrain. The diagram on the left is a small region that covers a 5 x 5 area of the height field (i.e. a region close to the camera) and the one on the right covers a larger 17 x 17 area (i.e. a region farther away from the camera). For clarity, a blue line has been drawn from the height field point to the surface of the approximated terrain when the approximation is above the original and a red line has been drawn when the approximated point is below the original.

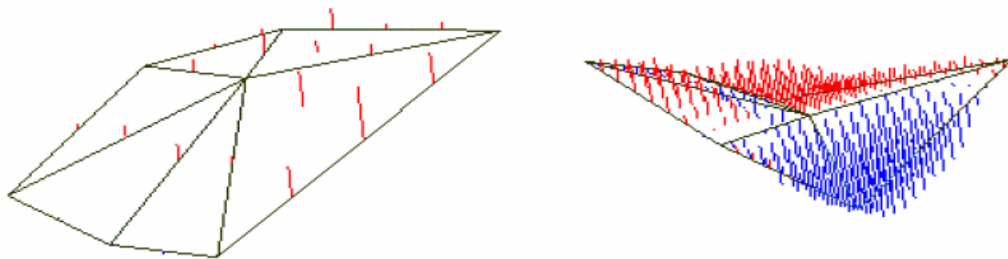


Figure 3.2 Calculating Terrain Accuracy

Once the vertical elevation difference has been computed for each point in the height field, the actual accuracy as a percent is calculated as follows:

$$\text{Accuracy} = 100 * ((\text{totalHeight} - \text{totalDelta}) / \text{totalHeight}) \quad (3.1)$$

In Equation. (3.1) the totalHeight is the sum of the heights of all the points in the height field and the totalDelta is the sum of the vertical differences of the points. This equation gives a global terrain accuracy measure that dynamically changes as the terrain changes and generally increases as the terrain is further and further refined since the totalHeight of a terrain is constant and the totalDelta decreases as more points are added to the approximated mesh. We use plane equation (Equation 3.2 and 3.3) while calculating totalDelta.

$$Ax + By + Cz + D = 0 \quad (3.2)$$

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \quad (3.3)$$

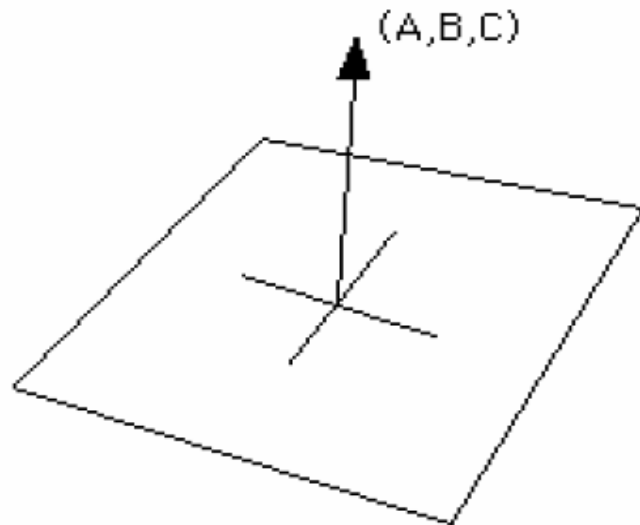


Figure 3.3. The normal to the plane is the vector (A, B, C).

3.3.2 Polygon Count

Determining the number of polygons in any given scene is a simple task that involves counting the numbers of triangles that are drawn each frame. To do this for the algorithms studied in this thesis there are three cases to consider.

The first case is demonstrated in the ROAM algorithm by Duchaineau [2] where each triangle is drawn one at a time. Here, the polygon count simply increases by one each time a triangle is drawn. The second case is demonstrated in the algorithm presented by Röttger [3] where triangles are sent to the hardware in the form of triangle fans. Here, when we create a triangle fan, the polygon count increases according to this triangle fans. The last case is demonstrated in the algorithm presented by Willem H. de Boer [4] where the triangles are sent to the hardware in one big triangle strip. Here when we create a triangle fan, the polygon count increases according to this triangle fans.

3.4 Test Cases and Results

This section describes the test cases that used to evaluate the implemented algorithms in this project and gives the results of these test cases. Testing will consist of two static viewpoint test cases; one to achieve a maximum terrain accuracy and another to achieve a desired number of polygons.

3.4.1 Terrain Accuracy Viewpoint Results

Our aim is reach to maximum terrain accuracy with minumum polygon count in this test case. There is no constraint for algorithm parameters. We try to run each algorithms at maximum limits. The parameters are set to maximum values for each algorithms.(See Table 3.14) Thus, each algorithm can be use the desired number of polygon. There is no constraints for polygon counts in this test case.

Table 3.14. Algorithm input parameters

Algorithm	ROAM	QuadTree	GeoMipmap
Max Detail Level	30	45	N/A
Min Resolution	N/A	10	N/A
Patch Size	N/A	N/A	33x33

We fixed camera position one point that is provide see entire terrain and apply the test cases to twelve data sets. The results can be shown at Table 3.15- 3.26.

Table 3.15. TerraGen - Subdivide & Displace Method results

Data Set Source :	TerraGen - Subdivide & Displace Method		
Data Set Size:	257 x 257		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	15	13	25
Vertices:	58017	48225	27364
Tris/s:	19939	34621	21860
Accuracy (%):	98,65	99,36	98,78
Render Time (ms):	95	109	47
Initialization Time (ms):	4596	1156	468

Table 3.16. TerraGen – Perlin Noise Method results

Data Set Source :	TerraGen – Perlin Noise Method		
Data Set Size:	513 x 513		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	10	5	8
Vertices:	50205	88880	76136
Tris/s:	16735	62659	60776
Accuracy (%):	97,60	98,89	98,79
Render Time (ms):	78	125	110
Initialization Time (ms):	4874	1297	406

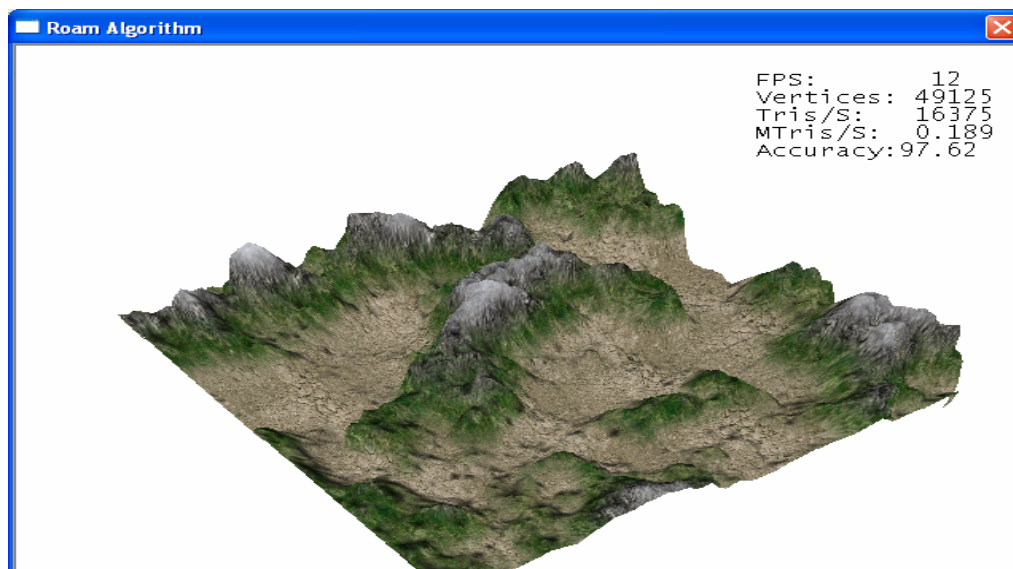


Figure 3.4 ROAM algorithm TerraGen – Perlin Noise data set texture map view

Table 3.17. TerraGen – Multi Perlin Method results

Data Set Source :	TerraGen – Multi Perlin Method		
Data Set Size:	513 x 513		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	11	5	7
Vertices:	53607	113166	89964
Tris/s:	17869	80854	71820
Accuracy (%):	97,53	99,17	98,93
Render Time (ms):	78	156	125
Initialization Time (ms):	4876	1328	531

Table 3.18. TerraGen – Ridged Perlin Method results

Data Set Source :	TerraGen – Ridged Perlin Method		
Data Set Size:	513 x 513		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	10	4	7
Vertices:	53115	197178	82192
Tris/s:	17705	143262	65584
Accuracy (%):	95,94	98,92	98,04
Render Time (ms):	78	250	125
Initialization Time (ms):	4731	1360	406

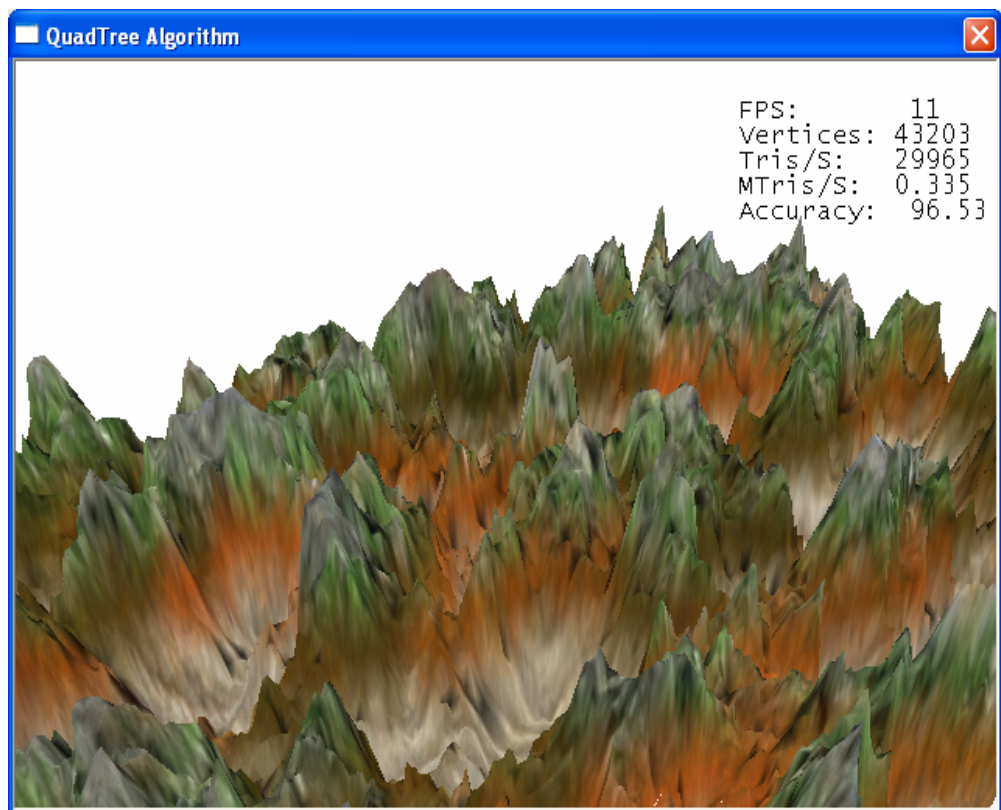


Figure 3.5 The Quadtree algorithm TerraGen – Ridged Perlin Method dataset view

Table 3.19. TerraGen – Ridged Multi Perlin Method results

Data Set Source :	TerraGen – Ridged Multi Perlin Method		
Data Set Size:	513 x 513		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	10	4	7
Vertices:	54051	156735	95172
Tris/s:	18017	113499	75972
Accuracy (%):	96,73	98,97	98,62
Render Time (ms):	80	235	141
Initialization Time (ms):	4855	1422	515

Table 3.20. SRTM – N37E029 zone results

Data Set Source :	SRTM – N37E029 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	3	3
Vertices:	34023	148786	186980
Tris/s:	11341	105536	149476
Accuracy (%):	96,15	98,48	98,97
Render Time (ms):	51	188	281
Initialization Time (ms):	4939	1922	672

Table 3.21. SRTM – N37E034 zone results

Data Set Source :	SRTM – N37E034 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	3	3
Vertices:	34785	148981	213452
Tris/s:	11595	106884	170700
Accuracy (%):	95,70	98,80	98,93
Render Time (ms):	51	172	265
Initialization Time (ms):	4997	1844	688

Table 3.22. SRTM – N38E027 zone results

Data Set Source :	SRTM – N38E027 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	3	3
Vertices:	31581	161448	147844
Tris/s:	10527	113180	118212
Accuracy (%):	92,22	96,80	97,62
Render Time (ms):	50	204	188
Initialization Time (ms):	5158	2000	672

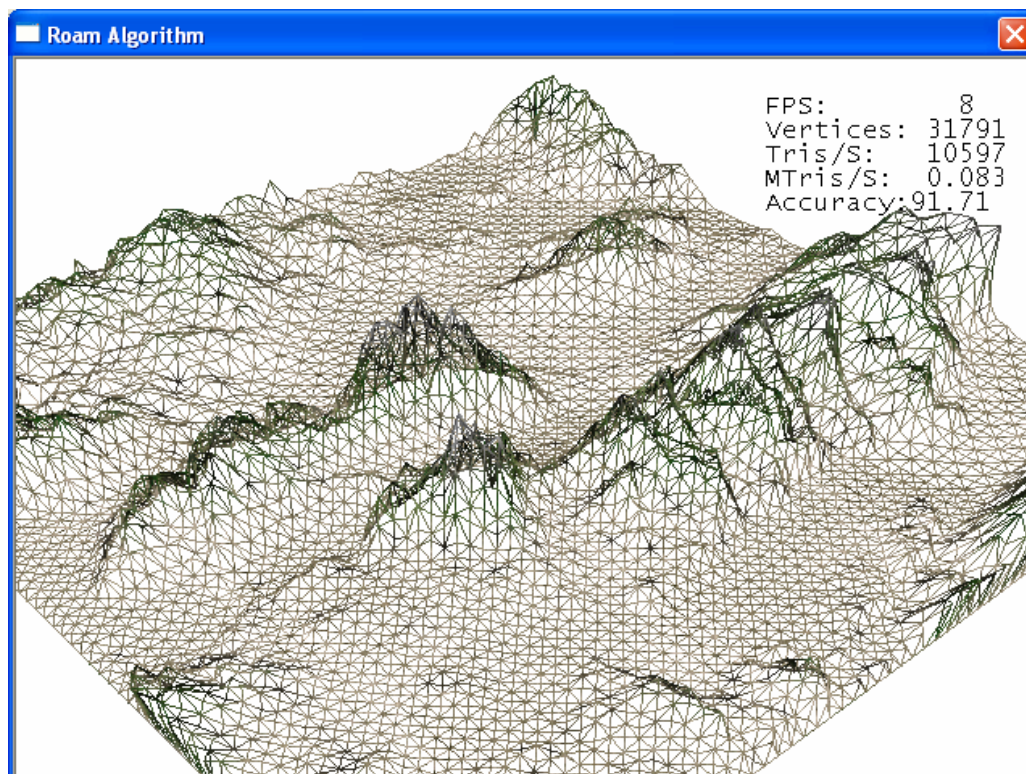


Figure 3.6. The ROAM algorithm SRTM – N38E027 zone results wire-frame view

Table 3.23. SRTM – N38E032 zone results

Data Set Source :	SRTM – N38E032 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	3	2
Vertices:	33024	100022	178668
Tris/s:	11008	71193	142892
Accuracy (%):	93,87	97,57	98,6
Render Time (ms):	49	142	267
Initialization Time (ms):	4908	2782	918

Table 3.24. SRTM – N39E041 zone results

Data Set Source :	SRTM – N39E041 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	3	3
Vertices:	32508	154005	189364
Tris/s:	10836	108106	151380
Accuracy (%):	96,24	98,61	98,92
Render Time (ms):	48	203	235
Initialization Time (ms):	4951	2578	734

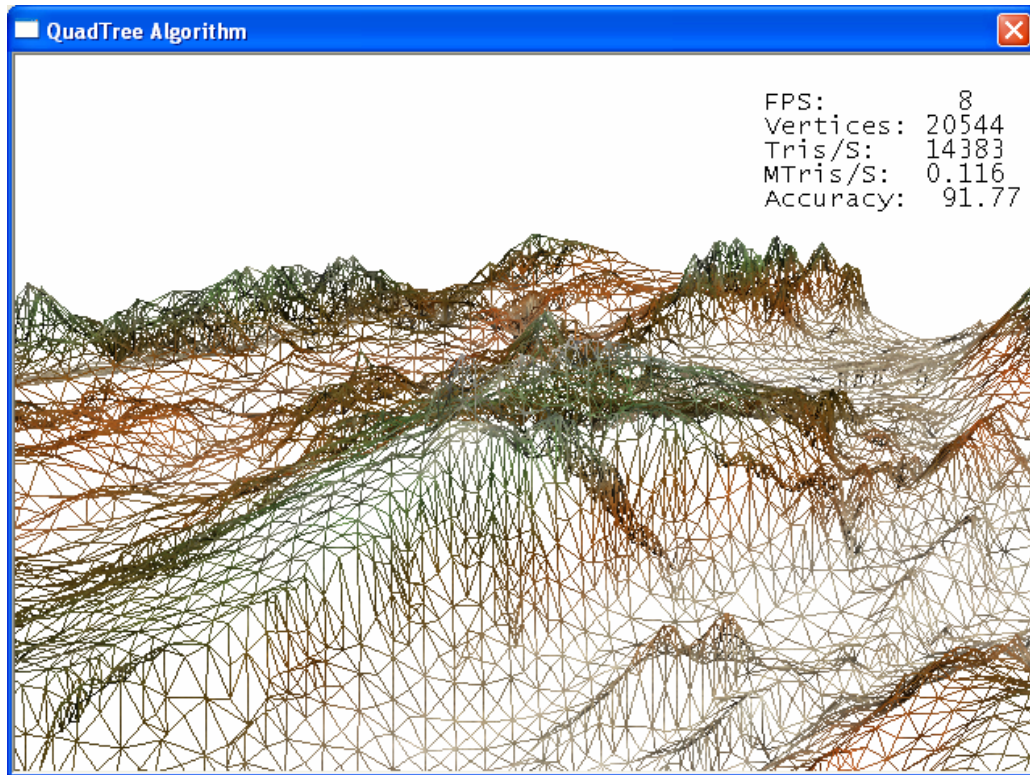


Figure 3.7 The Quadtree algorithm SRTM – N39E041 wire-frame view

Table 3.25. SRTM – N39E044 zone results

Data Set Source :	SRTM – N39E044 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	6	4	3
Vertices:	31878	98461	179892
Tris/s:	10626	71313	143860
Accuracy (%):	95,67	97,85	98,42
Render Time (ms):	62	125	235
Initialization Time (ms):	4961	1860	672

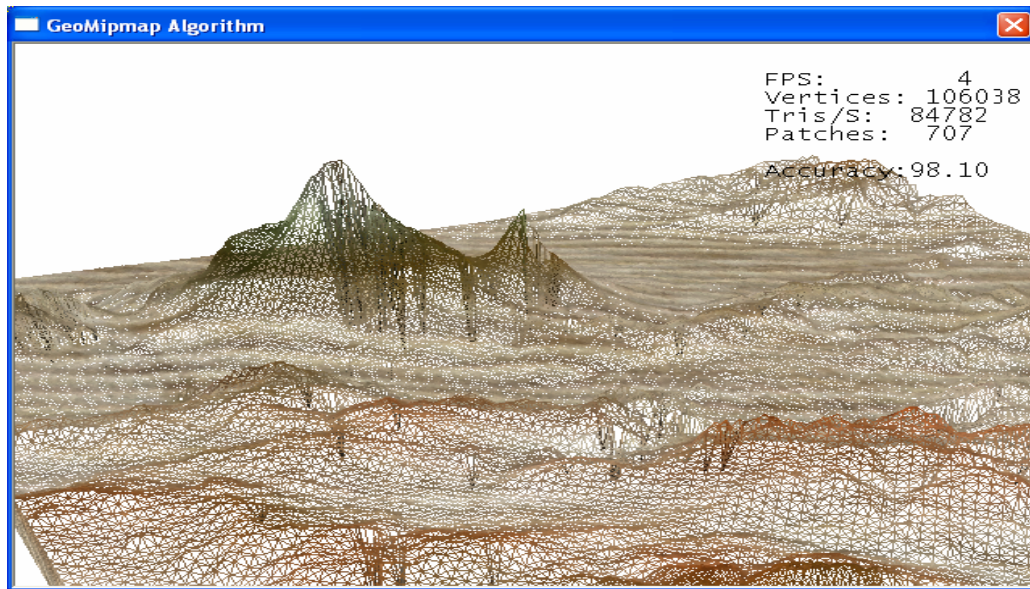


Figure 3.8 The GeoMipmap algorithm SRTM – N39E044 data set wire frame view

Table 3.26. SRTM – N41E026 zone results

Data Set Source :	SRTM – N41E026 zone		
Data Set Size:	1025 x 1025		
Algorithm :	ROAM	Quad Tree	GeoMipmap
FPS :	7	3	3
Vertices:	28695	189868	161660
Tris/s:	9565	135621	129276
Accuracy (%):	90,10	96,71	97,44
Render Time (ms):	50	216	219
Initialization Time (ms):	4987	1891	688

All of the results can be show at the below graphs.

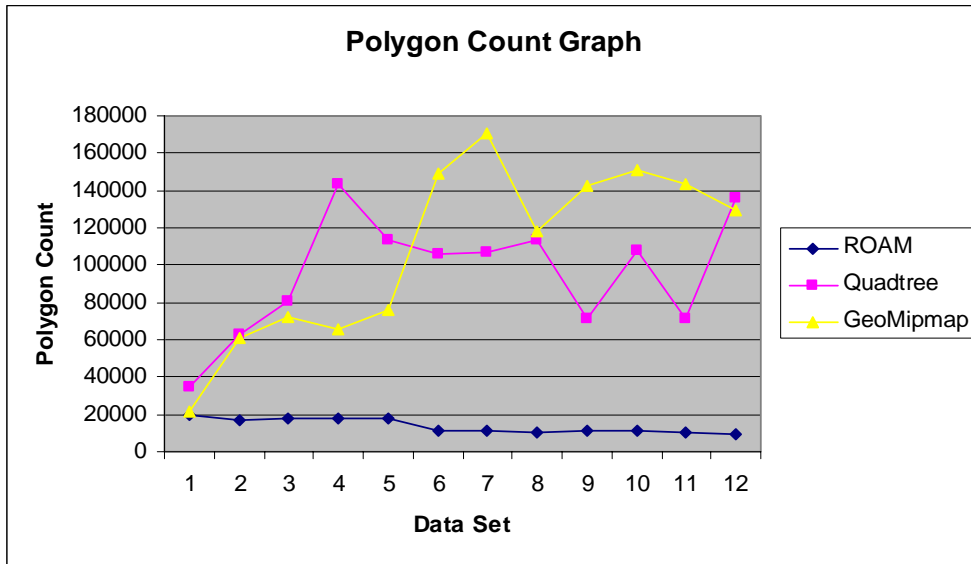


Figure 3.9. Polygon count result graph

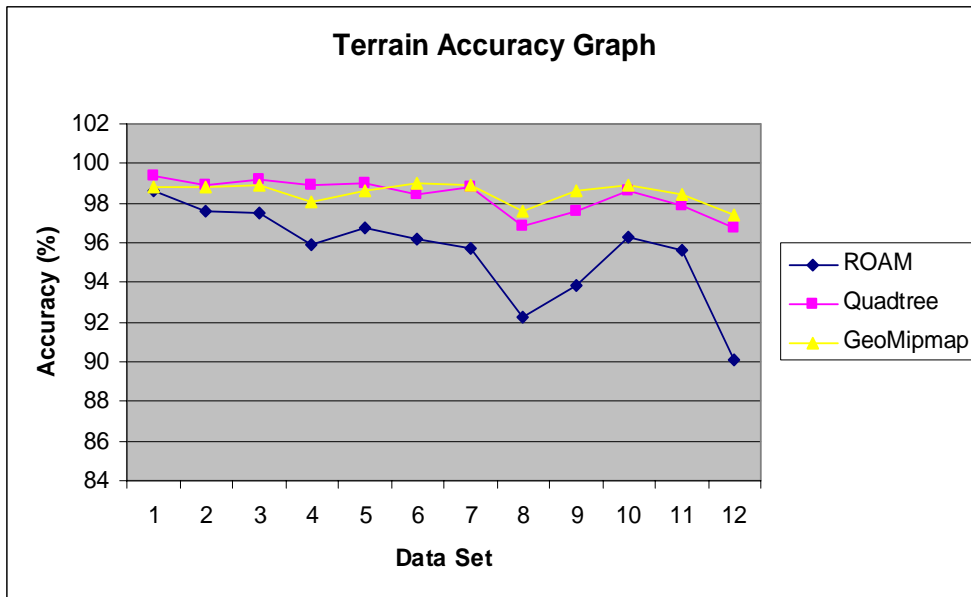


Figure 3.10. Terrain accuracy result graph

3.4.2 Desired Number of Polygons Viewpoint Results

Our aim is compare to performance of the LOD algorithms according to desired number of polygons . The polygon count is set to approximately 10000 and try to reach maximum terrain accuracy. We can't use Geomipmap algorithm for this test case. Because, minimum polygon count is approximately 40000 for our Geomipmap algorithm implementation.

Table 3.27. SRTM – N38E032 zone results

Data Set Source :	SRTM – N38E032 zone	
Data Set Size:	1025 x 1025	
Algorithm :	ROAM	Quad Tree
FPS :	7	8
Vertices:	29466	13613
Tris/s:	9822	9871
Accuracy (%):	94,29	93,68
Render Time (ms):	49	36
Initialization Time (ms):	4781	2127
Max Detail Level	20	20
Min Resolution	N/A	6

Table 3.28. SRTM – N39E041 zone results

Data Set Source :	SRTM – N39E041 zone	
Data Set Size:	1025 x 1025	
Algorithm :	ROAM	Quad Tree
FPS :	7	8
Vertices:	29565	13804
Tris/s:	9855	9904
Accuracy (%):	96,00	95,42
Render Time (ms):	59	35
Initialization Time (ms):	4982	2159
Max Detail Level	13	10
Min Resolution	N/A	6

3.5 Discussion of Results

Our main aim is compare the popular three LOD algorithms and find the best choice for terrain rendering according to polygon count and the terrain accuracy criteria. Firstly, we make the literature examination. As a result of this examination, we decided to study on the three LOD algorithms that are Real-time continuous terrain levels of detail algorithms described in the papers ROAMing Terrain: Real-time Optimally Adapting Meshes by Duchaineau [2], Real-Time Generation of Continuous Levels of Detail for Height Fields by Röttger [3] and Fast Terrain Rendering Using Geometrical MipMapping by Willem H. de Boer [4]. Each of the algorithms is very popular in the last decade and they use different polygon structure to make the polygon mesh. This is the reason why we choose these algorithms. After that we made an algorithm selection, implementation phase started. The selected algorithms have implemented using the OpenGL API. In order to test, we defined two test cases (See section 3.4). We prepared twelve different data sets and applied to algorithms.

In the paper by Duchaineau [2] the authors claim that the ROAM algorithm produces optimal meshes, and based on the test cases in section 3.4.1 and 3.4.2 this proved to be almost always true. According to polygon count viewpoint, this algorithm is the best algorithm. The authors claim that other two important feature in theirs algorithms;

Frame rate and triangle count bounding: The system may optimize its terrain simplification so that it always produces a set number of triangles in a given screen, or so that it optimizes to never take more than a set amount of time to optimize for the current screen. These are important factors the consider in most systems, since rendering the frame rate is often a bottleneck, from both the hardware and software perspective.

Guaranteed error bounds: An error-threshold metric is used by the system to determine approximation error, that is, the algorithm functions minimizing the difference between the rendered and actual landscape.

In Section 3.4.1 and 3.4.2, our implementation results verify the above facts. The ROAM algorithm is very successful about optimizing polygon count for twelve dataset (See section 3.4.1). The using polygon count always remains some bounds. There is no sudden increasing or decreasing at the polygon count (See Figure 3.9). Of the three main algorithms, ROAM is the only one that guarantees that the error will always fall between set parameters. Although, the ROAM algorithm's terrain accuracy performance is decreasing below the average values for dataset 8, 9 and 12 (See Figure 3.10); we say that our results again verify this fact. The reason of this performance decreasing, the purpose of a triangle bintree is to easily choose the local level of detail of the triangulation. If the terrain is flat or distant, only a few triangles are necessary to approximate a large area, while if the terrain is rough or close, more triangles are required [12]. The datasets 8, 9 and 12 are almost flat terrains and there are no abrupt elevation changes. Thus ROAM algorithm is reach to optimal mesh with minimum polygon count but this can't be reach to high terrain accuracy with respect to other algorithms. The ROAM algorithm outperformed both of the other two algorithms for all test data sets according to reached terrain accuracy with using minimum number of polygon. The main focus of the ROAM algorithm has been to minimize the total number of polygons displayed and guaranteed the error bounds on the screen at any point in time [1] [14]. This shows that the ROAM algorithm can be use even large terrain dataset.

The algorithm presented by Röttger [3], describes a continuous level of detail terrain-rendering algorithm using a quadtree data structure. This algorithm operates top-down which means that only the minimum numbers of computations are

performed to render the terrain at the desired resolution. This algorithm can be reached desired terrain accuracy values but it uses more polygons than the ROAM algorithms (See section 3.4.1- 3.4.2). The difference between the two algorithms comes from the using data structures. The bintree structure can generate more optimum mesh than the quadtree structure with respect to polygon count. The Quadtree algorithm should take into account the roughness of the terrain to ensure that flat areas use fewer polygons since less detail is required and bumpier areas use more polygons to show more detail [5]. Therefore, this algorithm proved to be the best algorithm to use in order to display large, sharp and jagged.

In 2000 Willem H. de Boer published Fast Terrain Rendering Using Geometrical MipMapping [4], introducing a new level-of-detail algorithm targeted for modern graphics cards. The authors claims that most of the algorithms were invented (long) before hardware rendering became the industry's standard, and therefore may not be suitable to be used in conjunction with 3D hardware rendering anymore. Therefore, new algorithms must be found that will give the best results when used together with 3D hardware rendering. Because, 3D hardware is able to process and render a large amount of triangles per frame, the algorithm can resort to more conservative culling methods, thereby not necessarily delivering the 'perfect set' of render-data, but pushing as much triangles through the pipeline as hardware can handle, with the least amount of CPU overhead.

The Geomipmap algorithm has two main advantages: it is relatively efficient on modern hardware and it is very simple. It is efficient since detail levels are selected at block-level rather than triangle level, which mean lesser CPU work per drawn triangle [21]. With geomorphs performed in vertex programs even more work is offloaded from the CPU onto the graphics hardware. But because of gaps the geomipmap needs to be re-triangulated each time the detail levels are changed,

which is not optimal. The simplicity of this algorithm is partly because of the simplification scheme. While simple, however, it is not the most polygon-efficient simplification scheme and this algorithm thus may require a higher polygon count than other algorithms to achieve a certain error threshold (See section 3.4.1). It can be reached desired terrain accuracy values but it uses too many polygons. This algorithm is the worst algorithm according to using the polygon count. Because of this, a more serious drawback of geomipmapping is its lack of scalability. As the terrain increases the number of geomipmap grows, and with a high count of geomipmap even at the lowest detail level, the polygon count may rise to unacceptable levels. The minimum number of polygon is approximately 40,000 for the Geomipmap algorithm, so that, we can't use this algorithm for second test case (See section 3.4.2). Although, it uses to five times more polygon than the others in

Table 4.1. Average performance results of the algorithms

Algorithm	Average Terrain Accuracy (%)	Average Polygon Count
ROAM	95,53	13814
Quadtree	98,34	95560
GeoMipmap	98,50	108484

some data set, this algorithm's average terrain accuracy value is the highest value (See Table 4.1). In conclusion, this algorithm does allow us to add details at runtime in a simple way and the algorithm provide acceptable, but not optimal, performance for terrains of limited size. We need to powerful graphic cards for using geomipmap algorithm. To make it able to perform well with large terrain sizes some modifications needs to be done. There is interesting fact from the results (See Table 4.1) that the algorithm presented by Röttger [3] and algorithm presented by Willem H. de Boer [4] met the highest terrain accuracy but these algorithms use too many polygons while the ROAM algorithm use fewer polygons and still maintain a acceptable terrain accuracy value.

CHAPTER 4

CONCLUSIONS

The results of evaluating the three algorithms studied in this project are very clear. The ROAM 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 an optimal mesh-generating algorithm.

FUTURE WORKS

While the ROAM algorithm is relatively old in computer graphics terms, it is still in use today in terrain rendering. One of the original developers is currently working on ROAM version 2[22], 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. We have used the classical binary tree structure in the ROAM implementation. If we change this structure with diamond structure, we will find better accuracy results.

The data culling techniques may apply to the quadtree and the geomipmap algorithm for reducing the polygon count. This data culling techniques will eliminate any scene information that will not directly contribute to the terrain accuracy.

REFERENCES

- [1]. C.P.Lo, A. K.W.Yeung, *Concept and Techniques of Geographic Information Systems*, Prentice Hall, 2002.
- [2]. M. A. Duchaineau, M. Wolinsky, D. E. Sietgi, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: Real-Time Optimally Adapting Meshes", in *IEEE Visualization*, 1997.
- [3]. S. Röttger, W. Heidrich, P. Slusallek, H.P. Seidel, "Real-Time Generation of Continuous Levels of Detail for Height Fields", *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization*, 1998.
- [4]. http://www.flipcode.com/tutorials/tut_geomipmaps.shtml
- [5]. D. Bradley, "Evaluation of Real-Time Continuous Terrain Level of Detail Algorithms", *Master Thesis*, 2003.
- [6]. D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, R. Huebner, *Level of Detail for 3D Graphics*, Morgan Kaufmann Publishers, 2003.
- [7]. T. A. Möller, E. Haines, *Real-time Rendering*, AK Peters, Ltd., 2002.

- [8]. P. Lindstrom, D. Koller, W. Ribarsky, L F Hodges, N Faust, and G Turner. “Real-Time Continuous Level of Detail Rendering of Height Fields”, Proceedings of SIGGRAPH, 1996.
- [9]. <http://www.longbowdigitalarts.com/seumas/progbintri.html>
- [10]. D.Hill, “An efficient, hardware-accelerated, level-of-detail rendering technique for large terrains”, Master Thesis, 2002.
- [11]. <http://www.magma.ca/~dhlf/downloads/detail-terrain-rendering.pdf>
- [12]. A. Ögren, “Continuous Level of Detail in Real-Time Terrain Rendering”, Master Thesis, 2000.
- [13]. B. András, “Real-Time Visualization of Detailed Terrain”, Master Thesis, 2003.
- [14]. <http://www.cs.sun.ac.za/~henri/terrain.html>
- [15]. http://www.gamasutra.com/features/20000403/turner_01.htm
- [16]. http://number-none.com/blow/papers/terrain_rendering.pdf

- [17]. http://www.gamasutra.com/features/20000228/ulrich_01.htm
- [18]. K. You, J. Tian, J. Liu, “Real-time Rendering of Large Terrain Using Quadtree Based Triangulation”, Proceedings of SPIE - Volume 4756, 2003.
- [19]. B. D. Larsen, N. J. Christensen, “Real-time Terrain Rendering using Smooth Hardware Optimized Level of Detail”, Journal of WSCG, 2003.
- [20]. T. Polack, *Focus on 3d Terrain Programming*, Premier Press, 2003.
- [21]. <http://www.terrain.dk/terrain.pdf>
- [22]. http://www.cognigraph.com/ROAM_homepage/ROAM2/