A Hybrid Algorithm for Terrain Simplification

by

Xuanying Li

B.E., Xiamen University, 1998

M.E., Xiamen University, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

May 2003

Abstract

Terrain surface simplification or approximation is very important in many terrain-related applications. Due to a variety of reasons such as the data acquisition method and more detailed features information, terrain models are extremely, and not necessarily, large nowadays. A fast algorithm that can produce compact yet highly accurate model would be desirable.

In this thesis, a hybrid algorithm integrating greedy insertion and quadric error based approximation is given. This algorithm is designed to provide a high-quality approximation of original model, yet reasonably fast. Experiments show quality improvements of the approximated models over previous methods.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I wish to thank my supervisor Dr. James J. Little. I wish to thank him for his inspiring guidance and his magnanimousness. For these two years, Jim has been a great help to me in both my academic and personal life. This work would have been impossible without his efforts.

I would also like to express my sincere gratitude to my cosupervisor – Dr. Wolfgang Heidrich. Dr. Heidrich is very patient and full of new ideas, and I benefited a lot from many stimulating conversations with him. I also want to express my special thanks to Dr. Will Evans for carefully reading my thesis draft and for his many helpful suggestions. Thank you all for all your kind help and suggestions.

I also want to thank my many wonderful friends, Jihong Ren, Li Rui, Kejie Bao, Qi Wei, Juan Li, Shaofeng Bu, Fahong Li, Peng Peng, Cai Kan, Zhimin Chen, Xiushan Feng and many other friends. Thank you all for making my life in UBC a wonderful memory.

I also owe a lot to my parents and brothers. Although far from home, I can always feel their love and support. Knowing that they are always there backing me up, I'm afraid of nothing.

My final, and most heartful, thanks must go to my husband Tianrun. He always sees the best in me, and gives me strength and hope. His support, encouragement, and companionship have helped me though the hard time studying alone in Canada. For all these, and for everything I'm now and will be in the future, he has my everlasting love.

XUANYING LI

The University of British Columbia
May 2003

Chapter 1

Introduction

The ability to create terrain models with increasingly greater details has increased these years. As a result, terrain models tend to be more and more complex, and extremely large; hundreds of millions of triangles are now common – many models have billions of triangles. However, hardware performance cannot match the increasing complexity of terrain models in many terrain-related applications, such as realtime rendering, network transmission and terrain analysis.

Theoretically and ideally, the complexity of a terrain model should be decided by the frequency of the terrain surface. However, nowadays, the complexity of a terrain model depends largely on the data acquisition methods (which will be discussed later) due to the fact that the frequency of a terrain surface is usually unknown. Therefore, a complex representation is not always necessary, or at least not necessary for the purpose of a specific application. For example, in computer rendering applications, for a certain frame, if the viewpoint is far from the terrain, then a very coarsely representation will suffice; for a closer viewpoint, the part that lies outside the view-frustum will be totally unnecessary.

Therefore, although high complexity usually means rich details, it is not always helpful; sometimes some parts of a complex model are totally unnecessary and redundant, and thus become a burden for applications. Meanwhile, in many applications involving terrain data, such as flight simulation, computer games and virtual environments, the time and resource demanded by complex terrain models are intimidating. Therefore, it would be ideal to reduce the complexity of terrain models, while at the same time maintaining desirable fidelity in the final results.

1.1 Terrain Applications and Terrain Simplification

Terrain data are used in a variety of applications, such as digital topographic maps, three-dimensional display of terrain for pilot training, virtual reality, games, landscape design and planning, viewshed analysis, planning of road or dam locations, computing of slope and aspect maps to assist geomorphologic studies, and estimation of erosion and run-off. A huge terrain data set will certainly slow down all applications, therefore the applications will benefit from a compact yet accurate terrain model. For different applications, the accuracy requirements may be different, and they may have other requirements such as the preservation of slope, aspect, visibility and drainage.

Generally, terrain simplification can be put as: given a set H of points (x, y, z) that represent a terrain, try to find a set of points H' (often, but not necessarily, a subset of H) to replace the old set H, so that H' is smaller and best approximates H, i.e, for a given application, the result using H' is most like the result using H itself.

1.2 Motivation

A model with a fixed resolution is not good for all applications and all users, due to their different requirements, computer resources, and available time. Therefore, it would be ideal for users to have the ability to control the size of the model, the time needed to process the model, and the accuracy of the model.

Research in terrain simplification field has been active and fruitful for the past years. However, achieving balance in the compactness, fidelity of the simplified model and the speed of simplification still remains an open issue.

The motivation is to provide rendering applications (but also useful for other applications) affordable (either in time or resource) terrain models yet with as high fidelity as possible. We have analyzed pros and cons of many excellent surface simplification algorithms, and decided to experiment on a hybrid algorithm which integrates greedy insertion with quadric error based surface simplification[GH97]. We hope that this integration will improve the quality of approximation models over the individual methods.

1.3 Thesis Outline

Chapter 2 will review some background knowledge, indispensable issues and recent works in terrain simplification. Chapter 3 will give the details of our hybrid terrain simplification algorithm. The experiment results and algorithm performance will be given in Chapter 4. Finally, Chapter 5 will discuss the possible applications and improvements of the hybrid algorithm.

Chapter 2

Background and Related Work

2.1 Goal of Terrain Simplification

For different applications, the goal of terrain simplification is different. For example, for some viewshed analysis applications, the goal could be to preserve the inter-point visibility of the simplified model, and thus accuracy in other aspects may be sacrificed. Some applications or algorithms may emphasize other properties such as color or texture as well. However geometric accuracy is the most common goal for all applications, because geometric accuracy is indispensable to the representation of the terrain either for rendering or for other terrain feature analysis. Geometric accuracy gives the applications much versatility to do different analysis and yield good results compared to original terrain models.

There could be two ways to achieve the goal of terrain simplification while trying to preserve geometric accuracy. First, given a terrain surface T with n vertices, find the most accurate approximation T' to T using only m ($m \leq n$) vertices. Second, given terrain surface T with n vertices and an absolute global error bound ε, find the most compact approximation T' satisfying $|T'(u,v) - T(u,v)| \leq \varepsilon$ for all $(u, v, T(u, v)) \in T$. An approximation T' satisfies $|T'(u,v) - T(u,v)| \leq \varepsilon$ for all $(u, v, T(u, v)) \in T$ is called an ε-bound approximation of T. In [AS], Agarwal and Desikanhas have proved that finding the smallest ε-bound approximation of a surface is NP-hard. Therefore, many algorithms have tried different heuristics methods to find the best possible results, which, of course, may not be optimal.

2.2 Representation of Terrain Surfaces

We have only finite storage and finite resource to process terrain surfaces. Therefore, an accurate yet compact representation of a terrain surface would be necessary. Many representations of the terrain surface have been proposed, and they can be divided into two categories. The first category uses mathematical functions to represent a terrain surface. Because usually one mathematical function is far from enough to represent a natural terrain surface, this method divides the terrain surface into many small patches, and fits a mathematical function to each patch. This method is complex, involving many technical difficulties. The second category is much more popular. This method uses a set of sample points, usually this is called **Digital Terrain Model** or **DTM**, to represent the surface. However, the frequency of a terrain surface is usually unknown. Therefore, this method tends to over-sample the terrain surface.

In [Mar78], Mark has given a very detailed classification for DTM based on the method of elevation representation. Regarding the composing elements of DTM, he divided DTM (what he called tabular representation) into line representation (DLG) and point representation. According to point distribution, he further divided point representation into TIN and DEM.

In this section, the three most common DTMs: DLG, DEM and TIN, are introduced and their pros and cons compared.

2.2.1 Data Acquisition Methods

Usually terrain surfaces in the real world are continuously variant. How do we solve the conflict between finite storage resource and the infinite complex real world terrain surface? Usually DTMs are collected from sampling and digitalizing.[DEM] divides the data source into primary data source and secondary data source. Primary data sources get terrain data by direct measurement using field sampling or remote sensing, such as stereo photogrammetry. The resolution of the data is determined by the density of sampling. Secondary data sources obtain terrain data from existing maps, tables or other databases. This method is restricted in many ways, because it needs to know the data acquisition and interpolation methods from which these maps are made.

2.2.2 Digital Line Graph(DLG)

A Digital Line Graph (DLG) refers to the digital vector data representing cartographic information. DLGs contain a wide variety of information depicting geographic features (for example, hypsography, hydrography, boundaries, roads, utility lines, etc). DLGs are derived from hypsographic data (contour lines) using USGS standards[DLG].

In this method, a terrain surface is represented mainly by a set of contour lines. Contour lines are curves that connect surface points with the same elevation value. They can be defined the as projection on the XY plane of intersections between the surface and horizontal planes[con].

DLG is a very popular, human-oriented terrain representation for maps. It is widely available, but not flexible for different terrain analysis and machine rendering. Therefore, this representation has limited direct machine applications.

2.2.3 Digital Elevation Models(DEMs)

A DEM is defined as a raster or regular grid of sampled height values. It is usually represented by a matrix of z(height) values for each point in a uniform grid. Each point in the grid represents a point (x, y, z), where (x, y) is implicitly specified by the position of the point in the grid. For example, the coordinates of top-left point are usually $(0, 0)$, indicating the origin of the coordinate system. There is a scale along the x axis and y axis to specify the real distance between grid points. The smaller the scale, the higher the resolution of the DEM, and also the more accurate the representation of the real terrain surface. Currently, the best resolution commonly available is $10m$ along the x and y axes, with $1m$ vertical resolution. The height of a point in a terrain can be taken directly from DEM if it lies exactly on the grid or by interpolation if it lies elsewhere.

The DEM representation is perhaps the simplest method for representing a terrain surface. DEMs can be derived from both primary data sources, such as stereo photogrammetry, and secondary data sources, such as conversion of printed contour lines. DEMs are easy to access, flexible to processing and efficient to store. Different DEM models cover most of the earth. The wide availability of DEMs makes them very popular in many commercial or non-commercial applications and software for features (drainage basin, watershed, drainage network and channels, peaks and pits, modelling of hydrologic functions, energy flux and forest fires) extraction and other applications.

However, the DEM model also has its disadvantages. Most DEMs come from uniform sampling; the sampling density is the same everywhere in the terrain. However, terrain surfaces are usually highly irregular. This feature of terrain surfaces usually demands a very dense sampling in order to make a reliable and accurate DEM representation. The sampling rate is the same for the whole terrain surface, either jagged surface or flat surface. This results in many unnecessary large and redundant DEM models.

2.2.4 Triangular Irregular Networks(TINs)

An alternative representation of a terrain, the Triangular Irregular Network(TIN), was proposed by Fowler and Little in [FL79] to address the limitation of DEM representation. TIN (Triangulated Irregular Network) is a vector data structure which depicts geographic surfaces as contiguous non-overlapping triangles. Within each triangle, the terrain surface is approximated by a plane defined by the three vertices of the triangle.

TINs do not come directly from most data acquisition methods. They are usually extracted from DEM models using some algorithms. Basically two dependent issues emerge to construct a TIN from a DEM. The first issue is how to choose a set of sample points as the vertex set of the TIN. The second issue is how to triangulate this vertex set. Almost all programs tend to choose sample points that are most important. However, they vary in the measurement of sample points. For the second issue, several triangulation schemes are available. Different triangulation algorithms have different preferences when choosing the shapes of the triangles (i.e., how to triangulate the vertex set) too. Some may choose to minimize the angle between the two normals of adjacent triangles in order to enforce a smooth transition, others may choose to maximize the minimum angle of a triangle, and still some others try to minimize the total length of the triangles. The most popular triangulation is the Delaunay triangulation. The Delaunay triangulation favors fat or equiangular triangles and tends to choose a triangulation that minimizes total edge length. The theory behind this is that the surface is more accurately represented at the vertices than at interpolated points far from a vertex. Therefore, Delaunay triangulation tries to make all points close to the vertices of the triangle where they are located.

Because the sample points in a TIN are irregularly distributed, more sample points can be used for jagged areas and fewer for flat areas. Therefore, a TIN may need fewer sample points than a DEM does to represent a terrain with the same accuracy. Moreover, the triangle is the most popular primitive for drawing – almost all graphics libraries and hardware support triangles. From this aspect, a TIN is very useful for rendering for most computer graphics applications. One more key advantage of TINs over DEMs is that the edges of triangles in TINs can be adapted to be aligned with the breaklines (ridges, cliff lines) of the terrain to better represent the real features of the terrain[FL79, LS01c]. Moreover, TINs are usually more flexible and useful for feature analysis such as silhouette extraction, visibility analysis and other possible applications. Currently there are also a lot of commercial and non-commercial software using TINs as input models.

In DEMs, only one coordinate for each point needs to be stored, while for TINs, three coordinates, and other connectivity information about the triangulation, are required to aid the access and processing of the whole TIN for each vertex. Therefore, it is hard to say which one is more efficient in terms of storage. If a DEM is highly redundant, then its TIN would be more storage-efficient[Mar78].

Compared to DEMs, TINs are more complex, and not so widely available as DEMs. However, many algorithms are available to convert a DEM to a TIN. So availability should not restrict the use of TINs. Because TIN models have many other advantages, they have also become popular input models for many programs.

2.3 Evaluation of the Quality of Simplified Models

There are several ways to evaluate how good a simplified model is. Different applications have different requirements. Therefore, their measures for the similarity between the simplified model and the original model are also different. Some may use one of the measures, while others may combine several of them.

2.3.1 Perceptual Similarity of Simplified Model

If the final purpose of a simplified model is for rendering, then the appearance on the screen is the thing that matters. Both an over-simplified approximation and changed topology will cause visual degradation of the simplified model. A very simple way to measure this similarity is to compare the pixel difference between the rendered images of the simplified model and the image of the original model. For a viewpoint, let the image rendered from the original model be I_o, and the image rendered from the simplified model be I_s. Suppose the size of the image is $n \times m$, and $I(u,v)$ is the intensity or RGB vector of a point (u,v) in image I, let $\parallel I_o(u,v) - I_s(u,v) \parallel$

be the difference of the two intensity values or RGB vectors (L_2-norm) $I_o(u,v)$ and $I_s(u,v)$, then perceptual similarity measure between the two images can be given by Equation 2.1.

$$E_{img} = \| I_o - I_s \| = \frac{1}{n \times m} \sum_u \sum_v \| I_o(u,v) - I_s(u,v) \|^2 \tag{2.1}$$

Of course, E_{img} is viewpoint dependent, so for the same pair of a model and its simplified model, the error measure is different for different view points. If a simplified model is used for rendering from all directions, then an error measure considering all view points will be necessary. However, this would not be feasible, because we have an infinite number of viewpoints. How to select the set of viewpoints to measure them remains an important problem.

2.3.2 Geometric Error of Simplified Model

Perceptual similarity defined above is only applicable to rendering applications. For other applications such as feature computing, geometric similarity is the most important measure. Geometric accuracy also helps improve the perceptual similarity.

Vertical RMS Error(VRMS)

The vertical RMS error for a terrain surface approximation is like the L_2 norm for function approximation. For a function $f(t)$ and its approximation $f'(t)$, the L_2 norm is given by Equation 2.2.

$$\| f - f' \| = \sqrt{\int_a^b (f(t) - f'(t))^2 dt} \tag{2.2}$$

Assume that a TIN is constructed to approximate the terrain surface represented by a DEM. To measure the error of the TIN, TIN_s, compared to the DEM model D_o with size $n \times m$, the VRMS (or L_2-norm) can be given by Equation 2.3.

$$\| TIN_s - D_o \| = \frac{1}{n \times m} \sqrt{(\sum_u \sum_v (H_{TIN}(u,v) - H_D(u,v))^2)} \tag{2.3}$$

The definition of H_{TIN} is given by Equation 2.4.

$$H_{TIN} = \begin{cases} z & \text{if (u,v) is a vertex in the TIN, and z is the height of this point;} \\ \frac{-(au+bv+d)}{c}, & \text{if } (u,v) \text{ is not a vertex, but it is in triangle} \\ & \text{ with plane equation } ax+by+cz+d=0 \end{cases} \tag{2.4}$$

Maximum Vertical Error

The maximum vertical error is a very strong error measure, providing a bounded error control over the whole approximated surface. It limits the approximation surface to lie between two offset surfaces.

The maximum vertical error for terrain surface approximation is similar to the L_∞-norm for function approximation. Again, the L_∞-norm for function $f(t)$ and its approximation $f'(t)$ is given by Equation 2.5.

$$\parallel f - f' \parallel_\infty = Max_{a \le t \le b} | \; f(t) - f'(t) \; | \tag{2.5}$$

Equation 2.6 gives the maximum vertical error (or L_∞-norm) of the approximation TIN_s for original model D_o.

$$\parallel TIN_s - D_o \parallel_\infty = Max_{0 \le u \le n} Max_{0 \le v \le m} \mid H_{TIN}(u,v) - H_D(u,v) \mid \tag{2.6}$$

The definition of $H_{TIN}(u,v)$ is given by Equation 2.4.

Signed Vertical RMS Error(SVRMS)

In addition to the two tradition measures mentioned above, we also tried signed vertical RMS error, which is given by Equation 2.7. For a point (p_x, p_y, p_z) and a plane $ax+by+cz+d=0 (c>0)$, if $ap_x + bp_y + cp_z + d > 0$, then its error is given a + sign ($SIGN_{p_x,p_y} = +1$), otherwise a − sign ($SIGN_{p_x,p_y} = -1$). The signed L_2

norm is very similar to L_2 norm, except that we add a sign before every error item.

$$\| TIN_s - D_o \|_{SVRMS} = \frac{1}{n \times m} \sqrt{\left| (\sum_u \sum_v (SIGN_{u,v})(H_{TIN}(u,v) - H_D(u,v))^2) \right|} \tag{2.7}$$

Vertical Mean Absolute Error

Vertical mean absolute error or VMAE is also a measure to evaluate the accuracy of simplified models. VMAE gives the average of the absolute values of elevation differences[JS98]. The VMAE between a DEM terrain H_D and its TIN approximation H_{TIN} is given by Equation 2.8.

$$\| TIN_s - D_o \|_{VMAE} = \frac{1}{n \times m} (\sum_u \sum_v |H_{TIN}(u,v) - H_D(u,v)|) \tag{2.8}$$

Vertical Mean Error

If errors with signs are used instead of the absolute errors as in the previous section, we have vertical mean error. Because signs are used, some errors may cancel each other; this makes the final error appear smaller than vertical mean absolute error. The VAE between a DEM terrain H_D and its TIN approximation H_{TIN} is given by Equation 2.9.

$$\| TIN_s - D_o \|_{VME} = \frac{1}{n \times m} (\sum_u \sum_v (H_{TIN}(u,v) - H_D(u,v))) \tag{2.9}$$

Comparison of Different Errors

So far there is no ideal error measure that claims to fully reflect the degree of fidelity of simplified models. Therefore it is necessary to understand the strength and weakness of the measures we have right now.

As we can see from its definition, the maximum vertical error is a very strong error measure, since it gives a global error bound over the simplified model. It guarantees that the deviation between the approximated surface and the original surface will be no further than the error bound. Compared to the maximum vertical

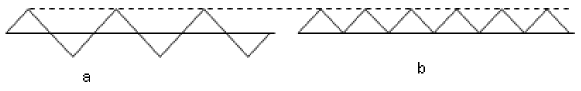


Figure 2.1: Comparing signed L_2 norm and L_2 norm

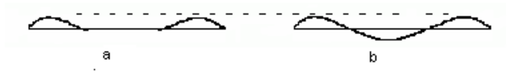
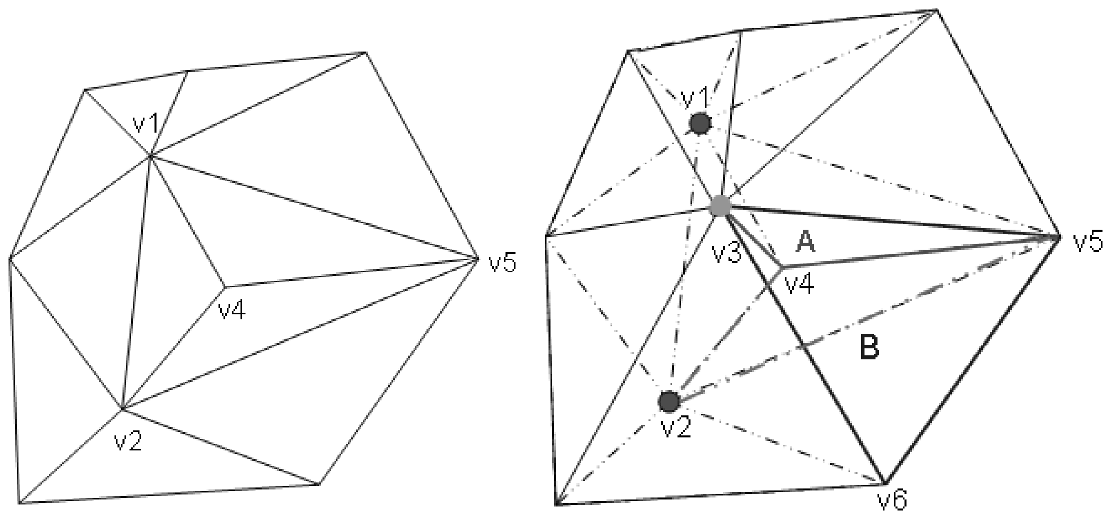


Figure 2.2: Comparing signed L_2 norm and L_2 norm

error, the RMS vertical error is not so strong, as it allows some big local errors, and the RMS vertical error is more robust to noise than the maximum vertical error. It is a good measure for the overall fitting quality of the simplified surface and the original terrain. In [Gar99a], Garland has carefully compared these two error measures.

Compared to the other two measures, signed vertical RMS error shares some properties with RMS vertical error. The signed vertical RMS error favors a more balanced approximation surface. RMS error is sensitive to noise and outliers. In [JS98], Jünger and Snoeyink argued that VMAE is a good measure since it is more robust against isolated errors and infrequent large errors, because VMAE weights all error equally, while RMS error tend to emphasize large, maybe not infrequent, errors.

For example, in Figure 2.1, for maximum vertical error, the errors of a and b would be the same, for signed vertical RMS error, the error of a is zero while the error of b is large. In Figure 2.2, teh maximum vertical error is teh same, vertical RMS error favors a, and signed vertical RMS error favors b. Therefore, the signed VRMS error favors an overall balanced approximated surface while allowing a larger deviation from the original surface.



(a) OriginalMesh (b) Mesh after Contraction

Figure 2.3: Invalid Mesh Structure

2.3.3 Evaluating the Quality of Mesh

Sometimes maximum vertical error and vertical RMS error cannot fully indicate the quality of an approximation. Because a DEM terrain surface is a 2.5D model, overlapping is not allowed. Therefore, the validity of the topology of approximated mesh is also an important measure.

Valid Topology

Sometimes certain operations in the simplifying process will result in unreasonable mesh structure. Consider the following case in Figure 2.3. The left picture is the original mesh, which is represented by dashed lines in picture *b*. After the contraction operation $(v1, v2) \rightarrow v3$, the two vertices $v1$ and $v2$ contract to the new point $v3$. Then triangle $(v2, v4, v5)$ will flip over to a new triangle $A = (v4, v5, v3)$; Then $(v4, v5, v3)$ will overlap with another triangle $B = (v3, v5, v6)$. This overlapping is not a valid topology, because we use a 2D surface in coordinates - not a 2D manifold in 3D.

Therefore, for simplification, it is necessary to check the validity of all operations to insure the consistency and validity of the topology of the simplified terrain surface.

Triangle Shape

As we mentioned before, triangulation methods all try to avoid sliver triangles while favoring fat triangles. There are several reasons behind this bias. One reason is that the points in a fat triangle have shorter distance to the vertices of the triangles, where the height value is exactly the same as original DEM. Another reason is that sliver triangles may cause other problems. In [MP97], the author points out, in addition to the potential inaccuracy caused by interpolation over a long distance, sliver triangles may also result in distorted visual effects due to aliasing and ill-conditioned linear systems which is not easy to solve.

In the triangulation process, there are several strategies to optimize the shape of local triangles, including

1. Minimize the maximum (or the sum) of $\frac{\text{longest edge}}{perimeter}$
2. Minimize the maximum (or the sum) of the aspect ratios of triangles.
3. Maximize the maximum (or the sum) of the minimal angle of triangles
4. Minimize the maximum (or the sum) of the angles between the normals of adjacent triangles

The Delaunay Triangulation, introduced by Delaunay[Del34] in 1934, has enjoyed great popularity for many years. In a Delaunay triangulation, for any triangles *abc* and *abd*, *d* is not inside the circumcircle of *abc*. For a vertex set V, the Delaunay triangulation for it is unique. In 2D, the Delaunay triangulation of a vertex set maximizes the minimum angle among all possible triangulations of that vertex set.

2.4 Review of Terrain Simplification Algorithms

The topology of a height field is relatively simple. Therefore, most surface simplification methods should be applicable. However, because general surface simplification algorithms need to consider many complex properties of different kinds of surfaces, they are more complex and less efficient compared to those algorithms designed to deal with only terrain surfaces.

In [HG97], Garland gives a very detailed survey of simplification algorithms (including those on terrain surfaces) covering the literature before 1996. He divided them into six categories:

1. Refinement Methods
2. Decimation Methods
3. Feature Methods
4. Regular Grid Methods
5. Hierarchical Subdivision Methods
6. Optimal Methods

Most algorithms accept DEM as input models, construct a compact representation from it, and usually output the result as a TIN. Refinement methods start from a very coarse model, and keep seeking the most important terrain point to refine the model until requirements are met. Decimation algorithms start from the full model, and keep killing unimportant points to get a compact model. Feature methods consider the structure lines as the most important part when constructing a compact representation[FL79], and they can start from both directions. Regular grid methods are similar to subsampling. Hierarchical subdivision methods start from the full model, and adaptively subdivide areas that have large errors. These five categories of algorithms are all heuristic. The algorithms in the last category are optimal methods. They try to find out the optimal approximation to the original model using a representation with a certain size or satisfying certain requirements.

The six categories of algorithms mentioned above are the basic algorithms for terrain simplification; there are still other algorithms that combine two or several of these methods. One kind of algorithm is a hybrid refinement/decimation method. In [HDD$^+$93], Hoppe et al. proposed an algorithm which uses both edge split (for refinement) and collapse (for decimation) to optimize a mesh. In [SC91], Schmitt and Chen adopted a two-stage process - split-and-merge for range segmentation applications. The split stage refined the triangulated approximation, while the merge stage merged adjacent triangles with similar normals. In [JS98], Jünger and Snoeyink present a two-side decimation/refinement. Decimation is done at the server by ordering the vertices so that the most important vertices are transmitted to the client first to support fast reconstruction over the network. Then those less important vertices are transmitted to the client to refine the reconstruction until it meets the resolution requirement of the user. In this way, decimation and refinement are combined to support progressive transmission. The algorithm we propose is also based on a hybrid idea.

In [CMS97], Cignoni et al. give a detailed characterization of mesh simplification algorithms and compare the performance of those algorithms. Another survey on general surface simplification is [Lue01]. Luebke surveys the field from the viewpoint of a developer, and gives the strengths and weaknesses of different approaches. However, both [CMS97] and [Lue01] mainly work on general surface simplification. Therefore, many recent works and trends on terrain surface simplification are not reviewed.

In this section, we will review terrain simplification algorithms after 1996. Since 1996, many new algorithms addressing new issues in terrain simplification have been proposed. Some tried to address terrain simplification for different application purposes, and others proposed new triangulation methods and new data structures.

For rendering, many terrain applications also adopted other techniques to reduce the popping artifacts between frame transitions. Basically the main stream in terrain simplification includes:

1. View-Dependent multi-resolution representation and terrain simplification
2. Appearance(or other feature) preserving terrain simplification
3. Multi-processor parallel terrain simplification
4. Out-of-core terrain simplification

2.4.1 View-Dependent Multi-Resolution Terrain Simplification

For real-time rendering systems, the requirement on resolution varies with viewpoint. For a close look, the resolution should be high; for a far viewpoint, the resolution could be very low. An early solution is LOD (Level of Detail)[Cla76]. Representations with different resolutions are handmade or precomputed and stored. Then at runtime, the system chooses the resolution according to how far the viewpoint is from the terrain. This method only offers a fixed number of resolutions, and only one resolution for the whole model at one time. LOD could be discrete, continuous or view-dependent. View-dependent multi-resolution representation is a more advanced solution. It allows variant resolutions across one model, and for each viewpoint, and the most economical and appropriate combination of resolutions will be computed. In recently years, hardware developments can hardly match the rendering demands of many applications. As a good solution to this conflict, view-dependent multi-resolution representation has been an active research field these years.

View-dependent Combined LOD based on TIN

TIN models are compact and flexible, and have many nice properties. Therefore, some researchers try to construct multi-resolution TIN representations.

In [BFM95], Bertolotto et al. propose a TIN hierarchy. In [dBD95], deBerg et al. first give a simplification algorithm that builds a hierarchical Delaunay triangulation and combines different levels of detail into a single representation. Surfaces with different LODs can be extracted in linear time. The algorithm divides the initial surface into many small groups of Delaunay triangles. Starting from the full model, the algorithm removes a number of **non-adjacent** vertices by selecting at most one candidate point from each group in one simplifying pass. In this way, one group of Delaunay triangles can be replaced by another group of triangles with lower resolution. Although the way the candidate points are selected allows preserving Delaunay properties and the construction of a hierarchy, the error of the next level is not guaranteed. In [EKT01], Evans et al. also analyze this kind of algorithm, and suggest dividing them into two subcategories based on whether the algorithm will preserve the boundaries of coarse triangulations in a finer level representation. Pros and cons of these two subcategories are also given.

View-dependent Multi-Resolution Representation based on Semi-regular Subdivision

Semi-regular subdivision restricts the replacement of vertices. It is not as flexible as a TIN; with the same number of triangles, the accuracy of a semi-regular subdivision is never superior to the best TIN (because the semi-regular subdivision is a TIN). But semi-regular subdivision is much simpler and faster to compute than a TIN. Therefore, it has a better chance to run in a realtime environment than a TIN. Because of these advantages, semi-regular subdivision terrain simplification has been a very active research field. For these algorithms, the most popular subdivision method is **Longest Edge Bisection** of isoceles right triangles[LKR$^+$96, RHS98, Ger03, LS01a, LP02]. Similar works include RTIN[EKT01, SP03], quad-tree triangulation[Paj98, CE01], or bintree[DWS$^+$97]. This subdivision refines an isoceles right triangle by bisecting the hypotenuse when higher resolution is necessary for this coarse triangle. Cracks are avoided by some dependent splitting.

In [DWS^{+}97], Duchaineau et al. propose an algorithm that allows real time rendering of models with thousands of triangles. During preprocessing, a nested view-independent error-bounded bintree is computed and stored. Then at run-time, view-frustum culling, view-dependent splitting and merging of triangles are performed. The triangulation is longest-edge bisection, splitting the hypotenuse of a right-isosceles triangle to get two new smaller right-isosceles triangles, and doing the inverse to merge two triangles. The transformation from one triangulation to another is a sequence of these splits and merges. Two monotonic priority (usually related to the error bound) queues, one for split and one for merge, are maintained to sort the error bound of the triangles in the mesh. The split and merge processes are driven by a greedy strategy. View-frustum culling, incremental T-Stripping, Deferring Priority recomputation and progressive optimization are used to optimize the algorithm.

In [CE01], Cline and Egbert propose a decimation algorithm called Q-morph. Q-morph is very fast, and can run in a real-time interactive environment. The DEM is subdivided to build quadtrees. When the viewpoint changes, these quadtree nodes are clipped against the view frustum. Those quadtree nodes that are inside the view frustum are further subdivided according to the LOD parameter which is controlled by the viewpoint. As usual, cracks are prevented by enforcing a common boundary between neighboring nodes. Popping artifacts are eliminated by a position-based morphing.

In [EKT01], Evans et al. propose an approximation algorithm based on a hierarchical RTIN(or right-triangulated irregular network). It is fast and supports multi-resolution representation well. The algorithm in [LKR^{+}96] is similar to this one. Compared to [EKT01], the method in [LKR^{+}96] is more like a combination of RTIN and a sub-grid. [LKR^{+}96] may be more storage efficient than [EKT01], because it stores an error for each vertex, while the method in [EKT01] stores an error per triangle. The benefit of this storage cost is that [EKT01] can guarantee a given tolerance of the original elevation model. The price for the speed and simplicity is accuracy. Evans et al. also compare the performance between RTIN and TIN, finding that the accuracy achieved by a RTIN as a function of the number of points in the approximation is worse than the accuracy of a TIN.

In [PAL02], the algorithm of Pajarola et al. combines the TIN model with the Quad tree structure. It constructs a quadtree triangle hierarchy on TIN terrain to provide adaptive LOD triangulation. The real time rendering of the height field uses additional Steiner points. To speed up rendering, triangle strips are used to represent the result.

In [SP03], Surez and Plaza present a very simple refinement and coarsening algorithm of complexity $O(n)$ and $O(logn)$ based on the RTIN. The error metrics they use are very interesting. They propose two error measures. The first one compares the area included in two contour lines between the original mesh and the simplified mesh. The larger the area difference, the worse the quality of the mesh. The second one uses a measure called Signal Noise Relation. The time complexity is similar to those algorithms in [EKT01, LKR+96]. The implementation of both data structure and algorithms are very simple.

2.4.2 Multi-Processor Parallel Terrain Simplification

Simplifying extremely large models demands a huge amount of memory and other computer resources. This demand, however, often cannot be satisfied by current hardware. Therefore, many researchers now try to partition a huge data set into small workable pieces, and simplify them either separately or sequentially, and then stitch the results from each small piece together.

In [GS], Ghodsi and Sack propose a solution to use more than one processor to compute terrain simplification. The algorithm begins with partitioning the terrain into small pieces, and simplifies them separately, and then combines them together.

In [KLYP00], Kang et al. partition a DEM into rectangular blocks of the same size, and simplify each of them separately with greedy insertion and Delaunay triangulation. The process for each block only refers to the points in the local block, rather than points in the whole DEM. They claim that to reach the same accuracy as greedy insertion for the whole DEM, their algorithm is about four to more than twenty times faster than greedy insertion. However, they did not give a scheme on how to choose the number of blocks. Also, we do not know which greedy algorithm they are comparing against.

2.4.3 Out-of-Core Terrain Simplification

Due to the huge amount of memory needed to simplify extremely large models, several "out-of-core" or "memory-insensitive" simplification algorithms are proposed. These algorithms use disk to replace memory, and store some intermediate results on disk. Because disk access is usually slow, external sorting and sequential access are used to minimize random access and minimize disk access time. A typical algorithm is OoCSx by Lindstrom et al. in [LS01a, LP02].

In [LP02], scalable out-of-core simplification and view-dependent visualization of general surfaces using external memory are presented. The whole process has three phases: the first two are off-line simplification and off-line LOD hierarchy construction, and the last phase is run-time view-dependent refinement and rendering. In the algorithm, an Octree is constructed over a uniform grid, each node of the Octree representing an expandable vertex node for certain LOD level. Simplification and refinement are done by collapsing or expanding vertex nodes.

2.4.4 Appearance(or other feature) Preserving Terrain Simplification

In [BMKN02], Ben-Moshe et al. propose a new method for terrain simplification based on a new quality measure. Because it addresses applications such as facility location which relies on inter-point visibility relationships, the most important goal of their algorithm is to preserve the inter-point visibility. Their basic idea is that the view between points is usually blocked by ridges. Therefore, they compute and approximate the ridge network first. This ridge network forms a natural subdivision of the terrain. Then each patch in the subdivision is simplified independently using one of the standard terrain simplification methods.

In rendering application, some terrain simplification algorithms also try to preserve some appearance features such as color or texture, not only geometric accuracy. The idea behind this is that color or texture is more important than purely geometric accuracy. In [Sug00], Suglobov sacrifices a little geometric accuracy for the sake of better textures of the simplified model in computer entertainment applications. However, these algorithms do not eliminate the demand on geometric accuracy; instead, they are usually based on geometric accuracy.

Miscellaneous Algorithms

In [AD97], Agarwal et al. propose a randomized algorithm to compute an $\varepsilon-$approximation of size $O(c^2 log^2 c)$ in $O(n^{2+\varepsilon} + c^3 log^2 c log \frac{n}{c})$ expected time where c is the size of the smallest ε-approximation and n is the number of terrain vertices. They formulate the terrain simplification problem as a 2D hitting set problem and adapt the randomized algorithm of Clarkson[Cla87] to compute a small hitting set. This is a suboptimal algorithm that guarantees an error bound. It has high time complexity and is hard to code. This kind of algorithms does not seem to guarantee better results than heuristics methods.

2.4.5 Summary of Previous Work

View-dependent multi-resolution representation is the trend of future terrain simplification, because it lowers the demand on computer hardware of many applications with huge terrain data sets. However, the basic simplification techniques remain the same. They are combined with other techniques to allow fast view-dependent rendering and multi-resolution in a single representation.

Chapter 3

A Hybrid Algorithm for Terrain Simplification

The hybrid algorithm is motivated by two basic terrain approximation algorithms. The first one is a refinement method – greedy insertion[FL79]. It starts from a very simple model, usually a model with only two triangles, and then keeps looking for the worst fit point and inserts it into the mesh.

The second algorithm is **Qslim** by Michael Garland in [GH97]. It is quadric error based decimation. Starting from the full DEM model, it calculates the cost to contract all valid vertex pairs (mostly pairs connected by an edge), and then contracts the vertex pair with lowest cost, usually, the least important points.

Decimation removes details whereas greedy insertion adds points to represent detail. Both methods are very popular, and QSlim is a good balance between accuracy and speed. However, both algorithms make irrevocable decisions as they go, and they never get a chance to review their decisions, while backtracking is too expensive. We suspect that there may be some shortsighted moves.

We thus propose a hybrid terrain simplification algorithm. The hybrid algorithm integrates the above two algorithms with the hope that this integration can retain the advantages of both while avoiding some disadvantages.

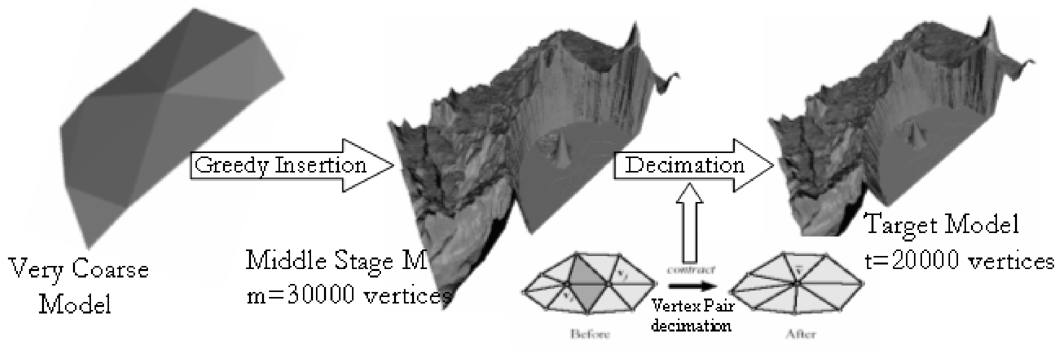


Figure 3.1: Hybrid Algorithm: A two step process

We assume the input data set is a DEM - a discrete two-dimensional set of samples D. The output of the algorithm will be a TIN. The hybrid algorithm is a two-step simplification process. The target model T has t vertices. First, a middle-stage model M with m vertices is constructed using greedy insertion; usually, the size of the middle stage model M is bigger than the target model T ($m > t$). In the second stage, the middle stage model M is shrunk to the target model T using iterative quadric error based vertex pair contraction. This process is shown in Figure 3.1. In the first step, we adopt Delaunay triangulation; for the second step, because of the optimal target vertex placement strategy, we do not enforce Delaunay constraints any more.

3.1 Greedy Insertion For Middle-Stage Model

Greedy insertion terrain simplification is a kind of refinement algorithm. This kind of algorithm has been developed for over two decades, and has many variants. Depending on how many points are inserted in one pass, it can be divided into two subcategories, *sequential greedy insertion* and *parallel greedy insertion*. Sequential greedy insertion inserts only one point in one pass, as in [FFP83], while parallel greedy insertion inserts a set of points in one pass as in [FL79]. Usually the candidate points are selected as the worst fit point in the current model, but some other feature points (points located in ridges, peaks, or valleys) are also added to the mesh by some algorithms[FL79, Hel90]. Greedy insertion is a basic idea in terrain construction, which can be combined with other techniques to allow hierarchical representation and view-dependent LOD representation of terrain models.

3.1.1 Triangulation Data Structure

There could be several ways to store a triangulation, such as a list (array) of triangles. However, this list does not provide topological information that records connectedness (adjacencies) between vertices, edges, and faces. Without topological information, many operations would be very clumsy. For example, finding the vertices adjacent to a given vertex and edge flipping. In [GS85], Guibas and Stolfi proposed the Quad-Edge data structure and discussed it together with its topological operators in great detail. In our implementation, we adapt the Quad-Edge structure to store the basic triangulation results.

3.1.2 Naive Greedy Insertion Algorithm

Let the target model T have t vertices, and the input model D be a DEM with n vertices. The pseudo code of a naive greedy insertion algorithm controlled by vertex number is:

Algorithm 3.1.1: GREEDYINSERTION(D, t)

```
/* initialize mesh (using the 4 corners of D) to 2 triangles */
$mesh(D.se, D.ne, D.nw, D.sw)$;
/* set current vertex number */
$i \leftarrow 4$;
while $(i \leq t)${
    $max\_error \leftarrow 0$;
    $candidate \leftarrow NULL$;
    for each (p∈D){
        $tri \leftarrow locate(mesh, p)$;
      /* Calculating the error of the point with respect to the triangle where it is located */
        $p.error \leftarrow tri.error(p)$;
        if $(p.error > max\_error)${
            $max\_error \leftarrow p.error$;
            $candidate \leftarrow p$;
        }
    }
    $mesh.insert(candidate)$;
    $i++$;
}
```

Obviously, this naive algorithm is very slow and inefficient. Each time we insert a new point, all the points in the DEM will be recalculated to update their errors. To calculate the error of a point, first, we need to find out which triangle this point is located in. This is done by a routine called $locate(mesh, p)$. A simple solution is to use the "walking method" by Guibas and Stolfi. Starting from an edge in the mesh, this method walks toward the target along the edges in the mesh. In the implementation, the algorithm will maintain a starting edge to start the walking. For a mesh with i vertices, the worst time for locating p is $O(i)$. Since there are

n points in the DEM, therefore, the cost will be $O(ni)$ in the worst case for each insertion. Usually, the starting edge is set to be an edge of the triangle that has the newest inserted point. This does not guarantee good performance. To estimate the time cost, let us assume that a random starting edge is selected. Guibas and Stolfi estimate the expected cost for this random locating operation to be $O(\sqrt{i})$ for a Delaunay mesh with i vertices. A better solution is to set the starting edge to be the edge that is returned by the last locating operation. This reduces the time to near constant, because we follow the raster order in the DEM when we process the points. In this case, we can expect the time for one location to be constant $O(1)$, and for one loop, the total cost for n points would be $O(n)$.

Then, the interpolation of each point in its corresponding triangle needs to be calculated to get its error. After all errors are calculated, they need to be sorted to find out the k points with highest errors in parallel greedy insertion. Without using a heap, the total cost per pass (including cost for quick sorting) would cost us $O(nlog(n))$ in the average case, and $O(n^2)$ in the worst case. However, if we only need to find the point with maximum error, the cost will be reduced to $O(n)$ per pass.

The last step is to insert the point(s) with highest error to refine the current coarse mesh. When inserting a point, edge swappings are necessary to maintain the Delaunay triangulation. The time for an insertion depends on how many edge we need to swap. In the worst case all the edges are swapped, which will take $O(i)$ time for a mesh with size i. Therefore, for sequential single-point greedy insertion, the worst case time complexity per pass would be $O(ni)$ for locating points, $O(n)$ for selecting the worst point, and $O(i)$ for insertion. The entire algorithm takes $\sum_{i=1}^{t} O(in) = O(t^2n)$ time. In practice, we expect the time per pass will be $O(n)$ for locating points, $O(n)$ for selecting candidate point, and $O(i)$ for inserting a point, making the total time $\sum_{i=1}^{t} O(n) = O(tn)$. For a large terrain model, this cost is still too prohibitive. Therefore, we adopted several optimizations to speed up the algorithm. After all the optimizations, the algorithm is very similar to the greedy insertion in [GS95].

Exploiting Locality using Triangle Structure

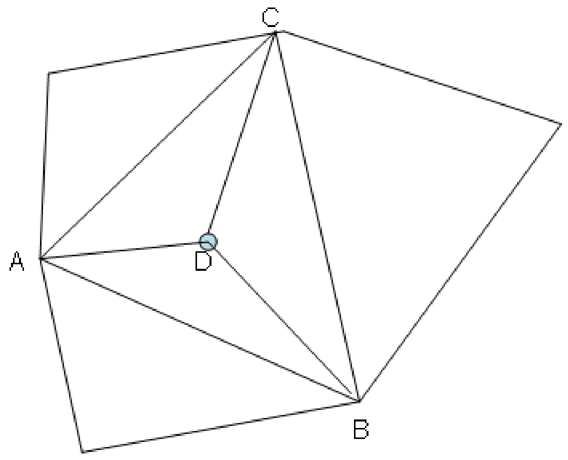
In addition to the mesh (quad-edge) structure to store the topological information, we add one more basic data structure – triangle. The triangle is a very basic element in our implementation. We make the triangle an "intelligent agent". Each triangle is responsible for selecting its candidate point, updating the errors of the points when the triangle is newly created, and maintaining its error measure.

When a new point is inserted into the mesh, the affected region is called the update region. The size of the update region varies with different meshes and different parts of the mesh, but usually only a very small part of a mesh needs to be updated for one insertion. For example, consider the insertion in Figure 3.2.

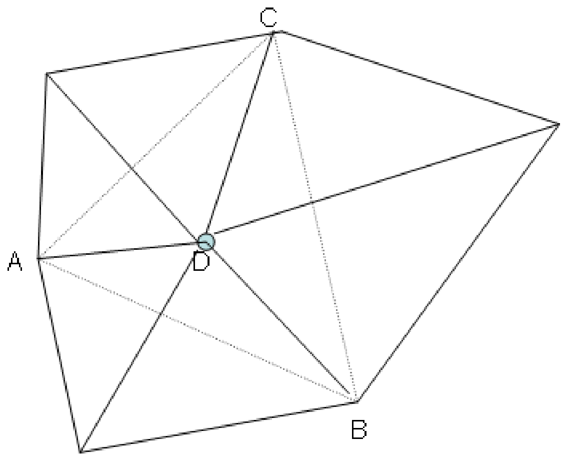
When point D is inserted, only triangle ABC and the three triangles around it need to be retriangulated, and only the points that are in these four previous triangles need to update their errors.

Using triangles, the update region can be easily tracked. Each time we insert a new vertex, some old triangles, which contain the inserted point or had their edges swapped, will be killed, and their contribution to the total error statistics (such as highest error, total volume error) will be cancelled, and a new set of triangles will be there to replace them. The update region, obviously, consists of only those newly created triangles. Upon the creation of a new triangle, the triangle will scan itself to update the errors of those points located in its territory, search for its highest error and corresponding candidate point, keep a record of its volume error, and also update the error statistics with its appropriate parts. Therefore, after we finish the triangulation of the update region, the update region has already been updated.

By using this mechanism, upon each insertion, the huge effort to recalculate every triangle's error at every pass is avoided.



(a) Insertion



(b) After Swapping

Figure 3.2: Locality of Insertion

Heap Structure for Faster Candidate Selection

In Section 3.1.2, without heap, the smallest cost for one selection in a sequential greedy insertion algorithm will be $O(n)$ for an input model with n vertices. For a parallel greedy insertion, the cost will be $O(nlogn)$ per selection when using quick sort. We need an efficient way other than scanning the entire grid to find the ideal candidate in each pass, for both sequential and parallel greedy insertion.

A heap is very appropriate structure here because of its relatively low cost for operations such as insertion and deletion and constant time to extract the top element. For a heap with size i, the cost for an insertion and a deletion will be $O(logi)$, and cost for extraction of top element will be $O(logi)$. Therefore, we use a heap to store necessary information to select candidates for insertions. Usually a heap node stores a pointer to a triangle, and the error of the triangle's candidate as its key value. Each time a new vertex is inserted, some obsolete triangles will be replaced by some new triangles, and the heap structure will maintain the sorting information of all the current triangles in current mesh.

So how many insertions and deletions will we do in one insertion? Basically, for an insertion, one big triangle (the top triangle in the heap) where this point is inserted will be deleted from the heap, and three new triangles will be inserted, and for each edge swap, two old triangles will be deleted and two new ones will be inserted. Therefore, for each insertion, the size of the heap will grow by two. Letting the number of total edge flips be k, the total deletion and insertion operations will be $(3 + 4 * k)$. However, we expect this k to be a small constant number, as we have seen from tracking how many times *heap_delete* and *heap_insert* are called. Therefore we can expect that for each pass, the cost for heap operations will be $O(log(i))$.

Algorithm 3.1.2: OPTIMIZED GREEDY INSERTION(F, t)

```
/* initialize mesh using the 4 corners of F to a 2 triangles */
$mesh(D.se, D.ne, D.nw, D.sw)$;
/* set current vertex number */
$i \leftarrow 4$;
while ($i \leq t$){
        /* process all the changed edges to retriangulate update region;
        New triangles will be created here; */
        $Process\_UpdateRegion()$;
        $toptri \leftarrow heap.extracttop()$;
        $toptri.insertcandidate()$;
        $i++$;
}
```

Algorithm 3.1.3: PROCESSUPDATEREGION()

```
Edge e $\leftarrow mesh.UpdateEdgeList.extract\_first()$;
while ($e! = NULL$){
        /* remove the next edge in its triangle */
        $UpdateEdgeList.remove(e.next)$;
        $UpdateEdgeList.remove(e.prev)$;
        $ntri \leftarrow CreateNewTriangle(e)$;
        $ntri.ScanTriangle()$;
        $heap.insert(ntri)$;
        $e \leftarrow mesh.UpdateEdgeList.extract\_first()$;
}
```

Worst Case Time Cost

In the worst case, when we insert a new point, all the old triangles are affected by this new insertion, therefore, the update region will be $O(i)$. In this case, the cost of heap operations can reach $O(ilog(i))$. Since these triangles cover all the points in the input model, we need to calculate the new errors of all n points, taking $O(n)$ time. Therefore, the worst case time cost will be $\sum_{i=1}^{t} O(ilog(i)+n) = O(max\{t^2lgt, tn\})$.

Expected Case Time Cost

How big is the update region? In [GS95], Garland pointed out that, for most surfaces, the update region has expected area $O(n/i)$, since n points divided among approximately $2i$ triangles gives an average area of $O(n/i)$. As we analyzed previously, in each pass, the cost of the heap operations is $O(log(i))$, and the cost for updating the mesh is $O(1)$. Therefore, the cost on the average will be $\sum_{i=1}^{t} O(n/i + logi) = O((n+t)logt)$

3.1.3 Error Measure

In Chapter 2, we have given two commonly used error measures (vertical maximum error and vertical root mean squared error) and a short comparison between them. In this section, the error measures we use in our algorithm will be discussed in full detail.

We have experimented with ten error measures to evaluate the quality of the mesh and to select the next insertion candidate, based on work of Little and Shi[LS01c]. In Table 3.1, a summary of these measures are given.

Orthogonal Error

Orthogonal error (or normal error) is defined as the perpendicular distance between a point and a plane.

In Figure 3.3, the normal distance between point p and plane A is N_a, and N_b is the normal distance between point p and plane B, while V_a and V_b are their vertical correspondents.

Name	Meaning
M2	Max Vertical Error
V2	Sum of Norm Error
V3	Sum of Signed Norm Error
V4	Sum of Vertical Error
V5	Sum of Signed Vertical Error
V6	Triangle Area
V7	Sum of Squared Norm Error
V8	Sum of Squared Vertical Error
V9	Sum of Signed Squared Vertical Error
v10	Sum of Signed Squared Norm Error

Table 3.1: Volume Error Criteria

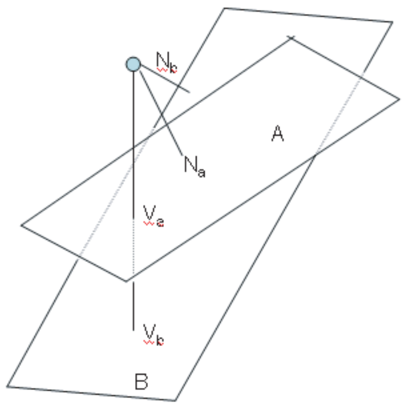


Figure 3.3: Comparing Normal Error and Vertical Error

Vertical Error

In Figure 3.3, V_a and V_b are vertical errors.

The vertical error of a triangle is the highest vertical error among all the points in its territory. If the vertical error is chosen to be the error criteria, then the key value of the heap will be the vertical error of each triangle. The algorithm will choose the triangle that has highest vertical error and insert its candidate point.

Volume error

The two methods mentioned above consider the point that has the highest error in a triangle, while volume errors take all the points in a triangle into consideration. In Table 3.1, letter V means volume error. When one volume error is selected, it will be the key value in the heap, and the top triangle in the heap has the highest volume error. However, the criteria for selecting a candidate in a triangle still remains the same – the candidate point always has the highest vertical error among all the points in this triangle.

$V2$ uses the sum of normal error as the volume error of the triangle. $V3$ uses the sum of signed normal error. $V7$ is the sum of squared normal error. $V10$ is the sum of signed squared normal error. $V4$ and $V5$ are the correspondents of $V7$ and $V10$ with vertical error. $V8$ is the sum of squared vertical error and $V9$ is the sum of signed squared vertical error. $V6$ uses the area of a triangle as the key value in the heap. The algorithm will always select the biggest triangle to insert a new point. However, for flat area, a big triangle is not a problem. We do not expect the performance of $V6$ to be high. We are curious to know which one is best.

3.1.4 Insertion Strategy

With the heap and triangle data structure, the hybrid algorithm can support both sequential and parallel greedy insertion very easily.

Sequential Greedy Insertion

On each pass, the top triangle in the heap will be selected to insert its candidate point. Then after insertion, the update region is retriangulated and the errors of those points in this area will be updated.

Partially Parallel Greedy Insertion

In one pass, the strategy for parallel greedy insertion is to insert a predefined number of points or insert points that have error bigger than some threshold all together. This may not be good, as many researchers have pointed out, because it is easy to cause unnecessary dense sample points in one area. One insertion in this area may bring down all the local errors. However, because we insert at most one point for one triangle, the parallel greedy insertion in our hybrid algorithm is not the same as the parallel greedy insertion in the traditional meaning.

In the hybrid algorithm, we will select a bunch of triangles, either a predefined number or according to their highest error, and insert their candidate points. Because our new set of triangles is created after all new points are inserted, this parallel insertion of points is seamlessly supported by the hybrid algorithm. We are curious about how well this partially parallel greedy insertion works. Because of the previous analysis on parallel greedy insertion, we expect the result of partially parallel greedy insertion to be inferior to sequential insertion. We will give our results in a later chapter.

3.1.5 Feature Preserving Greedy Insertion

Based on work of Little and Shi in [LS01c], feature preserving is also supported in our algorithm. To make our program more flexible, we separate the feature extraction from mesh reconstruction. As a result, our algorithm would accept a feature file consisting of feature vertices and features edges as input, and those features will remain intact during mesh reconstruction.

3.1.6 Greedy Insertion Summary

Our algorithm is similar to the optimized algorithm III in [GS95]. However, our algorithm is more powerful and flexible, and yet maintains a good time complexity.

3.2 Top-down Decimation simplification

Many simplification algorithms employ decimation. Refinement mostly works on curves and height fields. However, decimation can work on more general surface such as polygonal surfaces. Decimation methods include vertex clustering, region merging, wavelet decomposition and vertex decimation. In [GH97], Garland and Heckbert proposed a new vertex decimation algorithm using quadric error metrics. Their algorithm is kind of "vertex-pair contraction", but the error measure they use is new and interesting. This algorithm is widely acknowledged because of its good balance between accuracy and speed.

3.2.1 Vertex Pair Contraction

The algorithm begins with the original full model. It iteratively removes vertices and degenerate faces from the mesh to simplify the original model. The whole procedure is performed in a greedy way; each time, it selects the best candidate pair of vertices, and contracts them. To perform contraction on a pair of vertices (v_i, v_j), it simply performs the following steps, as illustrated in Figure 3.4.

1. Based on error metrics, compute an optimal position using quadric error metrics for this pair of vertices, and change the coordinates of v_i to that optimal position.

2. Replace all v_j with v_i.

3. Delete v_j and the degenerate faces (usually two triangles that share edge $< v_i, v_j >$)

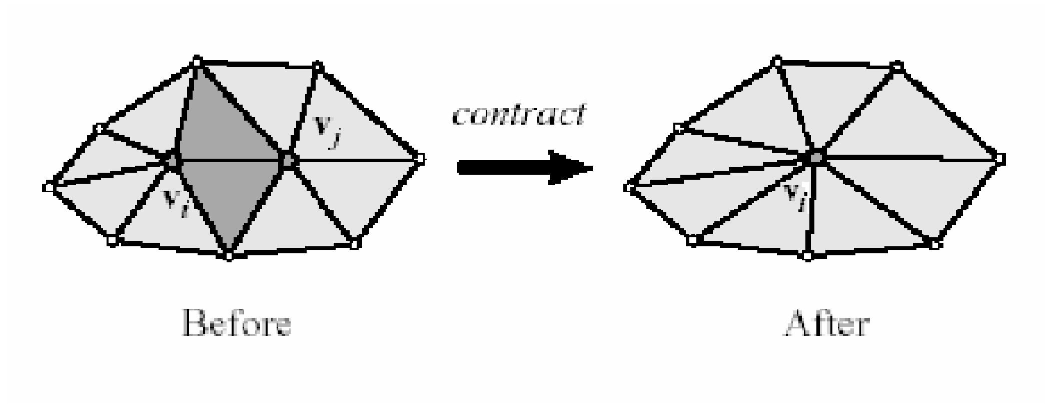


Figure 3.4: Vertex Pair Contraction

Vertex pair contraction is a generalization of edge contraction. In Garland's algorithm, contraction of a vertex pair that is not connected is allowed. This will sometimes change the topology of the surface. However, our hybrid algorithm only allows vertex pairs that are connected by an edge to be contracted. Therefore, in our case, it is in fact an edge contraction. This preserves the topology of the simplified surface.

3.2.2 Quadric Error Metrics

To evaluate the cost to contract a vertex pair, Garland et al. propose a new error measure – Quadric Error Metrics. Their idea is to find an error measure that is cheap to evaluate yet still good enough to provide a sufficient accuracy of approximation. They associate a set of planes with every vertex of the model. Those planes represent the faces that are incident to this vertex.

For a vertex in the original surface, the distance from this vertex to all the incident planes in the original surface will be zero. As many vertex pairs are contracted, the resulted vertices from these contractions may move away from the original surface. The further these resulted vertices move away from the original planes, the farther the approximation surface deviates from the original surface. Let us assume that those original planes are represented by (n_i, d_i), where $n_i = (a_i, b_i, c_i)$ and $a_i^2 + b_i^2 + c_i^2 = 1$, and the equation for this plane is $a_i x + b_i y + c_i z + d_i = 0$. Therefore, the sum of the distances between the vertex and the planes will be a good measure for how far the point (x, y, z) has deviated from the original surface. This measure is given by $Q(v)$ in Equation 3.1.

$$Q(v) = \sum_i (n_i^T + d_i)^2 = \sum_i (a_i x + b_i y + c_i z + d_i)^2 \tag{3.1}$$

When two vertices (v_i, v_j) are contracted to a new vertex v_i', the plane set for v_i' should be $planes(v_i) \cup planes(v_j)$. One solution is to explicitly track all the planes associated with each vertex. However, this may consume too much storage and time. Garland invented a very convenient way to represent this error using a matrix. Let $p_i = [a_i\ b_i\ c_i\ d_i]^T$, then $Q(v)$ can be rewritten as in Equation 3.2.

$$\begin{aligned} Q(v) &= \sum_i (n_i^T + d_i)^2 \\ &= \sum_i (p_i^T v)^2 \\ &= \sum_i v^T (p_i p_i^T) v \\ &= v^T \left(\sum_i K_i \right) v. \end{aligned} \tag{3.2}$$

where K is called fundamental error quadric matrix for a plane $p = [\text{a b c d}]^T$, given by Equation 3.3.

$$K_i = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{3.3}$$

To avoid tracking the planes that are associated with a vertex, when a edge (v_i, v_j) is contracted to a new vertex v_k, the algorithm assigns the fundamental error quadric matrix of the new vertex v_k to be the sum of the fundamental error quadric matrix of its two parents, that is, $Q(v_k) = v_k^T(K_i + K_j)v_k$. This method introduces some inaccuracy, because if the two plane sets of the two parent vertices are not disjoint, then the two planes that have (v_i, v_j) as one of their edges will be counted twice; Garland has analyzed that in the worst case, a plane could be counted three times. The benefit from this compromise is the great savings in storage and time used for tracking the plane set that is associated with a vertex.

It is ideal that with the same surface, different tessellation will not affect the quadric error metrics. However, with the above quadric error matrix, this is not the case. For example, in Figure 3.5, because all the triangles come from the same plane, therefore, the plane equations are the same, and their fundamental error quadric matrices will also be the same, let it be K. For the left tessellation, the sum of all the plane matrices will be $2K$, while for the right tessellation, the sum will be $4K$. As a result, for the same vertex v, and the same new target position v', the error of the right tessellation will be two times that of the left tessellation, which is not reasonable. To solve this problem, Garland uses *area_weighted* quadrics. Letting Q be the quadric determined by plane f, he divides the face f into three fragments, and constructs three fundamental quadrics w_1Q, w_2Q,w_3Q for the three fragments. Each vertex v of f will receive the fundamental quadrics w_iQ of the fragment adjacent to the vertex. He also proposes using area as the weight, that is, w_i is assigned a value according to the area of the fragments. His solution is to divide the face f into 3 equal parts. Therefore, each fragment will receive weight $w_i = area_f/3$. Back in Figure 3.5, both tessellations will receive a quadrics $area_fQ$ and the quadrics will be tessellation-independent.

3.2.3 Vertex Placement Strategy

As we mentioned in 3.2.1, in vertex pair contraction, we are going to use a new vertex v_k to replace the original vertex pair (v_i, v_j). The choice for v_k will be a critical part

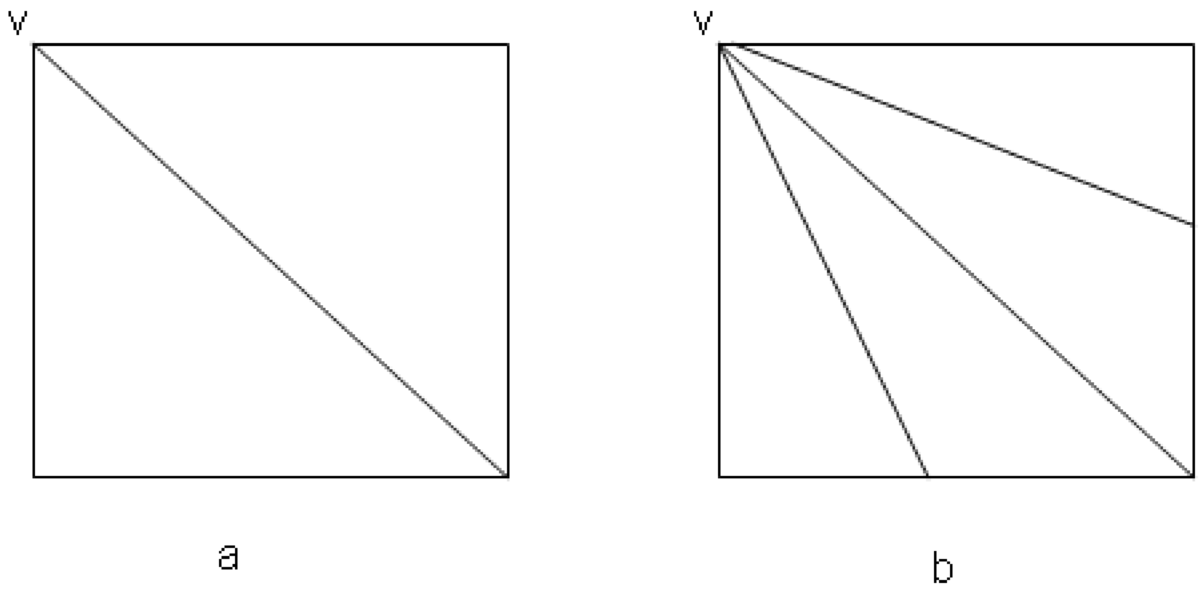


Figure 3.5: One Surface with Two Tessellations

to ensure a good approximation. In [GH97], one strategy is *subset placement.* The target position v_k is chosen to be either v_i or v_j, depending on which one, Qv_i or Qv_j, is better (smaller). *Subset placement* tends to be more storage efficient compared to other strategies we will discuss later, because we can store the contraction pair in such a way that the first vertex will be the vertex that replaces the original vertex pair. For example, if we are going to contract vertex pair (v_i, v_j) to (v_j), we will save the vertex pair as (v_j, v_i), so that we know that v_j replaces the pair. By doing so, no extra storage is needed. Meanwhile, using *subset placement* will produce an incremental representation.

An alternative strategy is *optimal placement.* We know that Q is a matrix, so an optimal placement will try to find a vertex v_k which makes $Q(v_k)$ minimal. Let

$$Q = \begin{bmatrix} A & b \\ b^T & c \end{bmatrix}$$

then we will have Equation 3.4.

$$v_k = -A^{-1}b \tag{3.4}$$

However, A could be singular when all the planes in Q are parallel. In this case we can not get v_k using Equation 3.4. For this case, Garland et al. suggest to find the best position along line segment between v_i and v_j. If this does not produce a unique solution, then use *subset placement*. Compared to *subset placement*, *optimal placement* needs extra space to store the new vertex v_k. However, the approximation produced by *optimal placement* tends to deviate less from the original, and mesh quality is better too–with better shaped triangles.

Garland has demonstrated the connection between the quadric measure and surface curvature. He visualizes the quadric isosurfaces, which reflect the local shape of the surface. The isosurfaces are long in the direction of low curvature while short in the direction of high curvature. The author also demonstrates that the solution given by Equation 3.4 is the center of the quadric isosurface, and also coincides with the optimal position derived using the least squares method to solve the normal equations [Gar99b]:

$$Nv_k = -d$$

where d is a vector of length k which satisfies $n_i^T v + d_i = 0$(the plane equation) and N is the $k \times 3$ matrix of the normals of the set of the planes:

$$N = \begin{bmatrix} n_1^T \\ n_2^T \\ \vdots \\ n_k^T \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ \vdots & & \\ a_k & b_k & c_k \end{bmatrix}$$

and v_k is the target placement of the vertex. His further analysis finds out that the eigenvectors of the quadric matrix A (which is given in Section 3.2.3) are roughly in the directions of the average normal, the direction of maximum curvature, and the direction of minimum curvature. These will be the axes of the resulting ellipsoidal isosurface.

3.2.4 Dealing with Discontinuities and Boundaries

Discontinuity usually indicates a significant feature such as a crease or a ridge. Decimation should avoid destroying these features to preserve visual fidelity. Quadric error metrics implicitly preserve discontinuity, because the algorithm is designed to be strongly biased against altering the shape of sharp edges. Let us take a sharp edge of a cube for example. A point on the edge will have two or more perpendicular planes adjoining it. Therefore, quadric metrics will favor the target point moving along the edge while punishing (giving high cost to) the contraction that moves the target point outside the edge. As for boundaries, special constraints should be enforced, because boundary the itself does not mean any natural discontinuity. Garland solves this problem by making it act as if it were a discontinuity edge. We make them sharp edges by adding a "virtual" perpendicular plane to a boundary face. To make it even stronger, we can add a big penalty factor. To avoid the tessellation affecting the weight for the quadrics of the boundary edges, the author uses squared edge length instead of using area to weight it. The author also generalizes this method to preserve arbitrary contours and points.

3.2.5 Mesh Validity Check

Sometimes a vertex pair contraction may result in undesirable changes, such as overlap, to the mesh. For example, in Figure 2.3, after contraction, triangle A and B fold over each other. Because all of our faces are pointing up with the z component of the normal positive, each time we plan to do a contraction, we will do a false contraction first, and if we find that one of the resulted faces will flip over, then we add a huge penalty factor to its cost, and reinsert it into the heap.

3.3 Combined Strategy

We have tried several strategies to integrate greedy insertion with quadric error based decimation. The two-step process we mentioned is just one strategy that works best. Let the target model have t vertices, let m satisfies ($m > t$), we test three strategies. The first one is to do the shrinking once, that is, grow the coarse model to size m, then shrink it to size t. The idea of the second strategy is to aim to t, the steps are given as following:

1. Grow to m_0
2. Shrink to t_0
3. Grow to m_i
4. Shrink to t_i
5. Repeat (3) and (4) until $t_i = t$

The third strategy is given as following steps:

1. Grow to m_0
2. Shrink to t_0
3. Grow to m_i
4. Shrink to t_i
5. Repeat (3) and (4) until $t_i = k$ ($k > t$)
6. Shrink the model from step (5) to size t

Chapter 4

Results and Performance Analysis

4.1 Time Complexity

In Section 3.1.2, we have analyzed the time complexity for the greedy insertion stage. To construct a middle stage model with size m from a DEM with size n, the worst case time cost for greedy insertion is $O(max\{m^2logm, nm\})$, and the "expected" time cost is $O((m+n)logm)$. For the second stage, the job is shrinking the middle stage model with size m to target size t. In [GH97], Garland has stated that the time complexity to shrink a model of size m to target size t is expected to be $O(mlogm - tlogt)$. Therefore, the overall expected time complexity will be $O((m+n)logm)$. Section 4.3.3 will further demonstrate that m will usually be no more than $2t$, which makes the final expected time complexity be $O((t+n)logt)$.

Figures 4.1 and 4.2 give a comparison of empirical running times between the Greedy Insertion, Qslim and hybrid algorithms for the cbaker terrain model. Figure 4.1 compares the time of the three methods when the target size is no larger than 19000 for the cbaker model, while Figure 4.2 compares the time of the three methods when the target size is larger than 19000 for the cbaker model. The hybrid algorithm

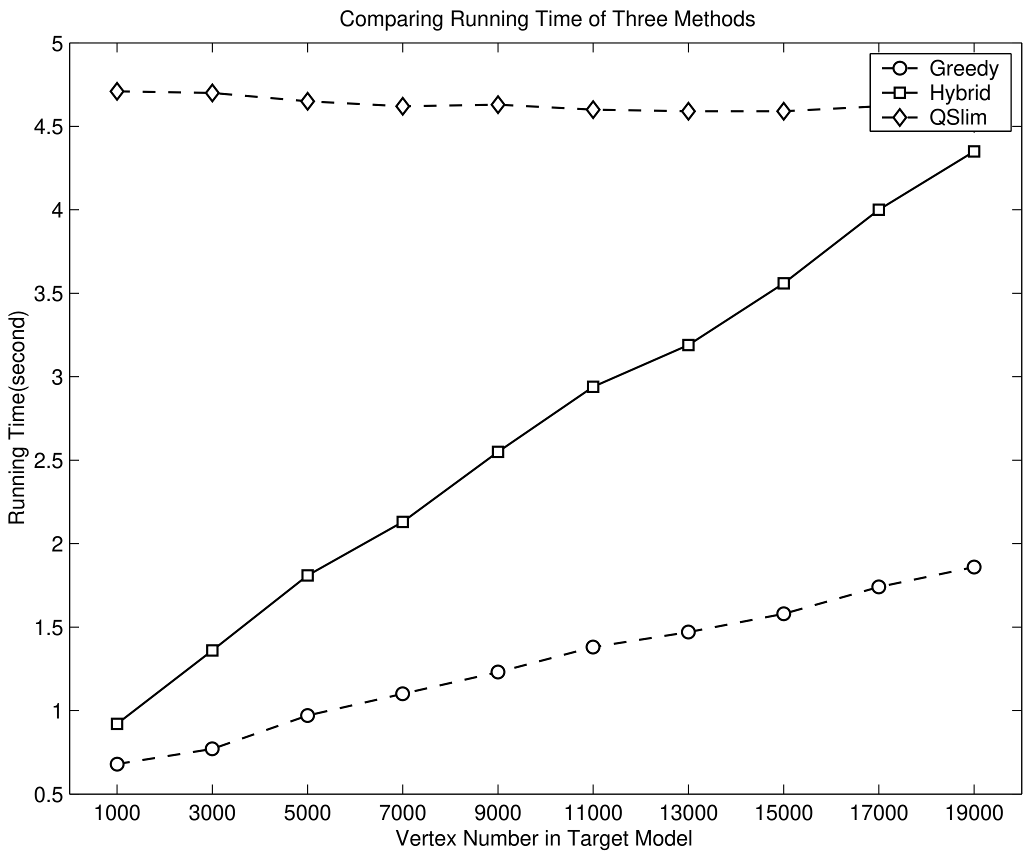


Figure 4.1: Cbaker: Comparing Running Time of Three Methods, with $t < 0.15n$

is faster than QSlim when we use less than 15% of the vertices of the original full model. However, after that, QSlim runs faster than the other two methods.

4.2 Space Complexity

In the greedy insertion stage, the memory usage is distributed among four data structures: they are the height field, the mesh, the heap, and the triangles. Let the size of the height field be n; the size of mesh, heap and triangles are all proportional to the number of vertices of the mesh m, therefore, the memory cost is $O(m + n)$.

For the second stage, in [Gar99b], Garland has analyzed the space cost. He gives his storage cost as 268 bytes per vertex, which is huge, and it is uniquely

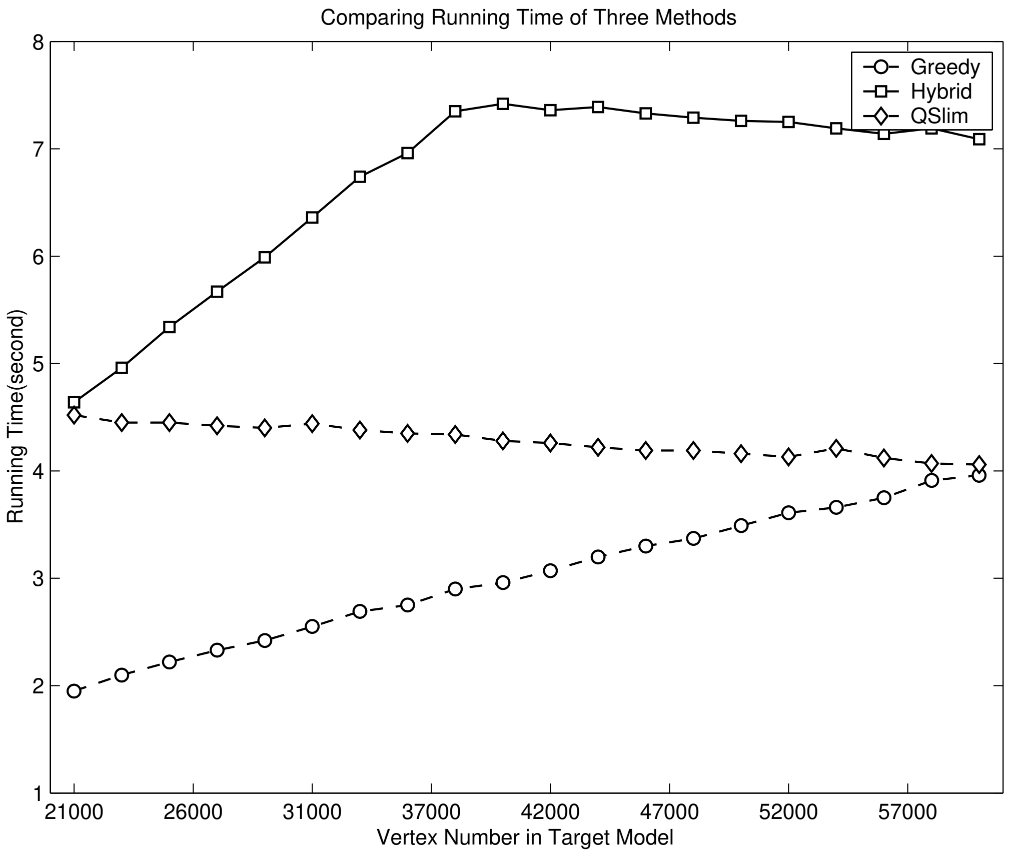


Figure 4.2: Cbaker: Comparing Running Time of Three Methods, with $t > 0.15n$

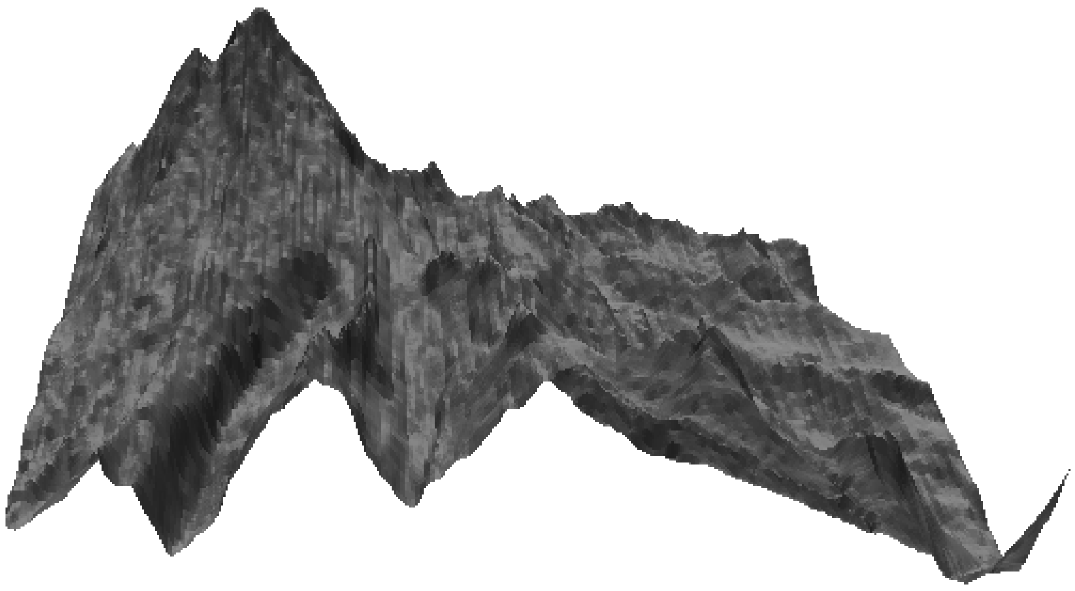


Figure 4.3: Full Model of cbaker; 127600 vertices, 253742 faces

decided by the input model itself. In the hybrid algorithm, the size of middle stage model is usually much smaller than n itself. As a result, the hybrid algorithm has an advantage in the space cost. The overall space cost will still be $O(n + m)$.

4.3 Quality of Simplified Models

One very important goal of terrain simplification is the high fidelity of the simplified models. Any algorithm, however fast it is, would be meaningless if it does not resemble the original model at all. In this section, I will give several pictures of the cbaker terrain model and its simplified versions produced by the hybrid algorithm, QSlim and greedy insertion.

The full model converted from DEM file is given in Figure 4.3. The mesh is obviously over-dense for the current view. Because the density of vertices is so high, the mesh can hardly be recognized. The model in Figure 4.4 is about half of the size of the full model. For the current view, it is hard to tell the difference between

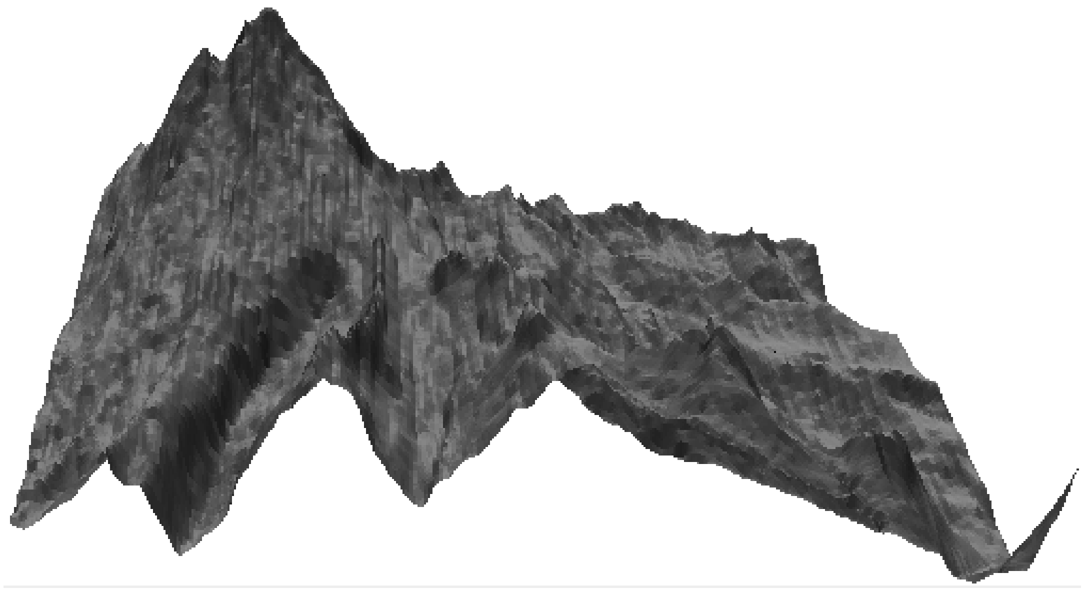
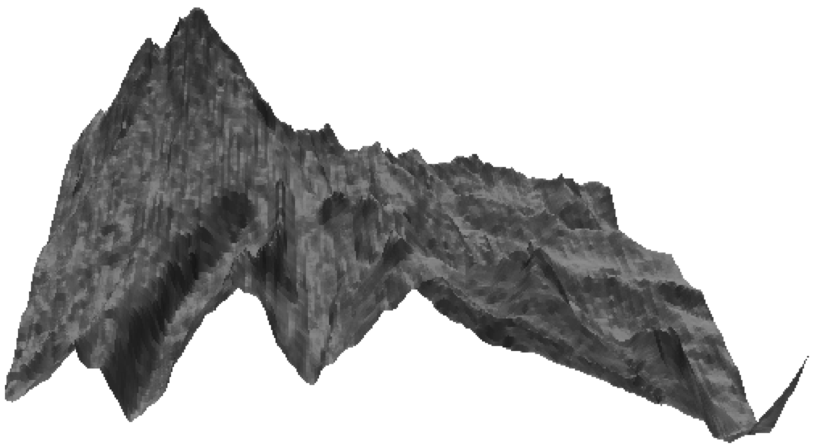


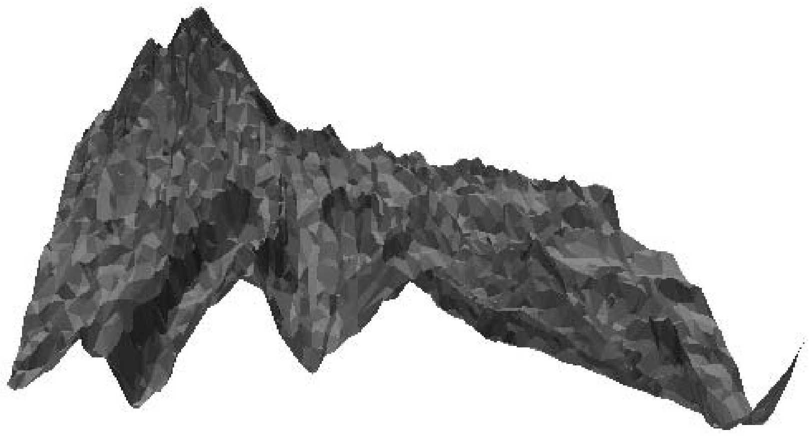
Figure 4.4: Simplified cbaker; 77445 vertices,153742 faces

it and the full model.

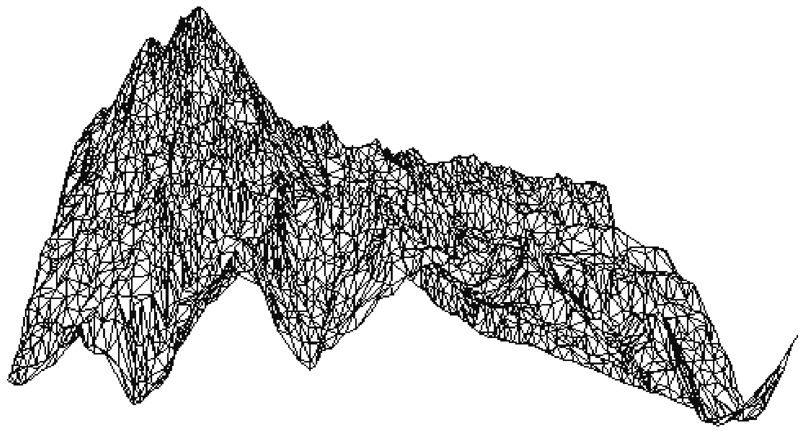
To compare the simplified models from three methods, the full model is simplified by a large degree to make the difference between the three methods distinguishable. Figures 4.5, 4.6 and 4.7 give the three simplified models with size about 4% of original model. In these three pictures, all of the three simplified models have good visual fidelity to the original model. It is hard to tell the difference by shading. Careful examination shows that the silhouettes from Figure 4.7 (by greedy insertion) and Figure 4.5 (by hybrid) are very similar and accurate compared to the original model, while the silhouette in Figure 4.6(by QSlim) has one missed peak and one incorrect round curve patch in the silhouette. Figures 4.9, 4.10 and 4.11 give the simplified models using only 1000 vertices. All three silhouettes are not accurate any more, but for the overall shading, Figure 4.9 is the best among the three. Figures 4.13, 4.14 and 4.15 give the simplified models using only 500 vertices. Greedy insertion captures one peak better than the other two, and the hybrid works better than QSlim, because the simplified model from QSlim has degraded very much. Again, for the overall shading, Figure 4.13 is the best among the three. Better shading means the model has more accurate curvature along the terrain surface, therefore, the fidelity of surface is higher. Those differences can be even more easily seen in Figures 4.8, 4.12, 4.16.



(a) FullModel

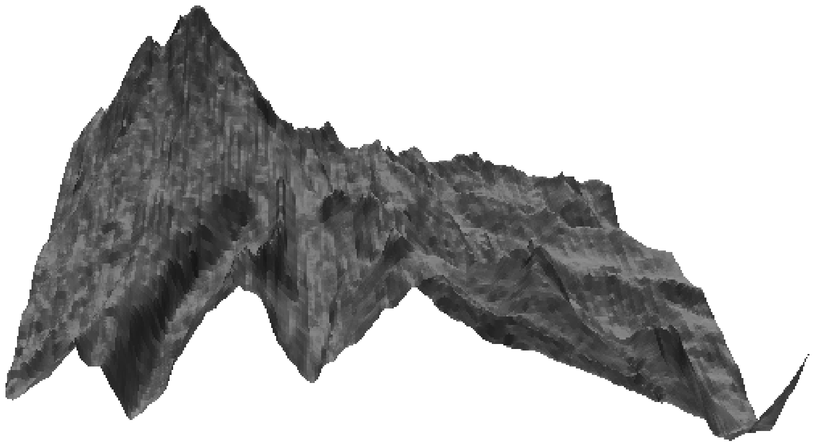


(b) Surface

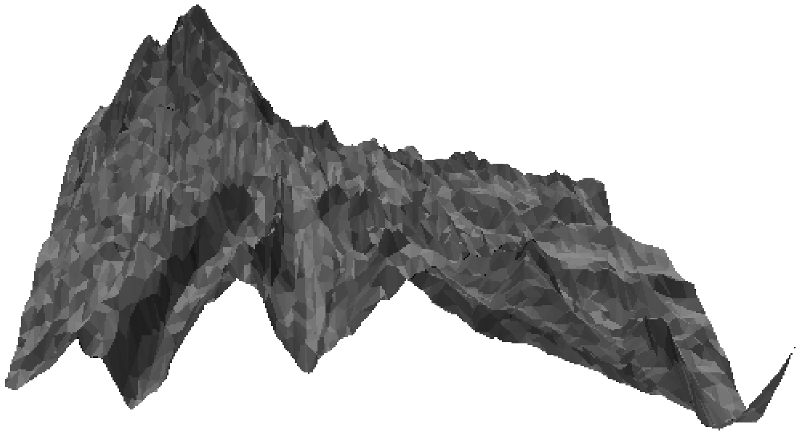


(c) Mesh

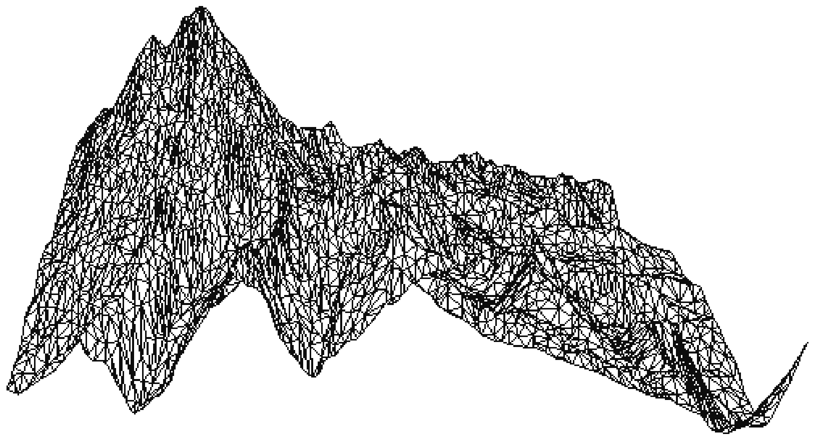
Figure 4.5: Hybrid Simplified cbaker; 5000 vertices, 9814 faces



(a) FullModel

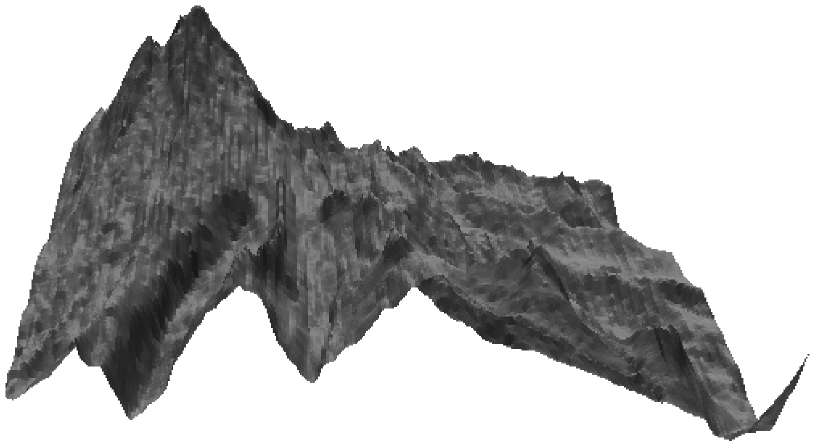


(b) Surface

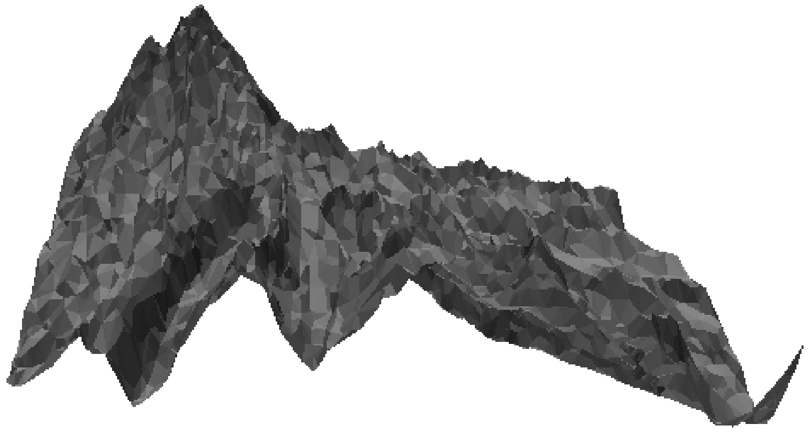


(c) Mesh

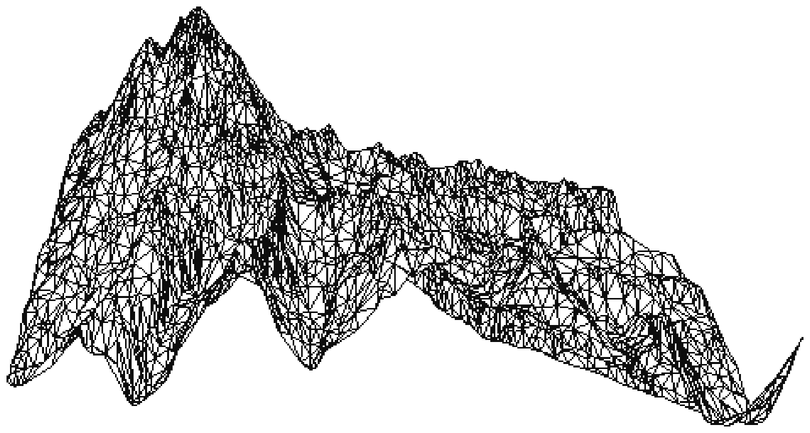
Figure 4.6: QSlim Simplified cbaker; 5000 vertices, 9515 faces



(a) FullModel

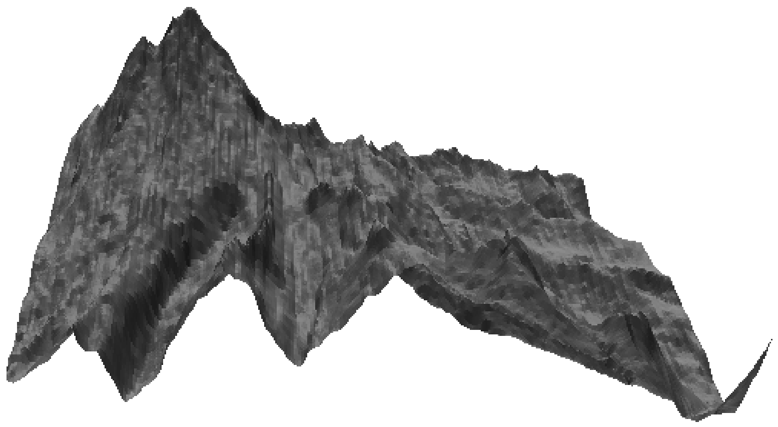


(b) Surface

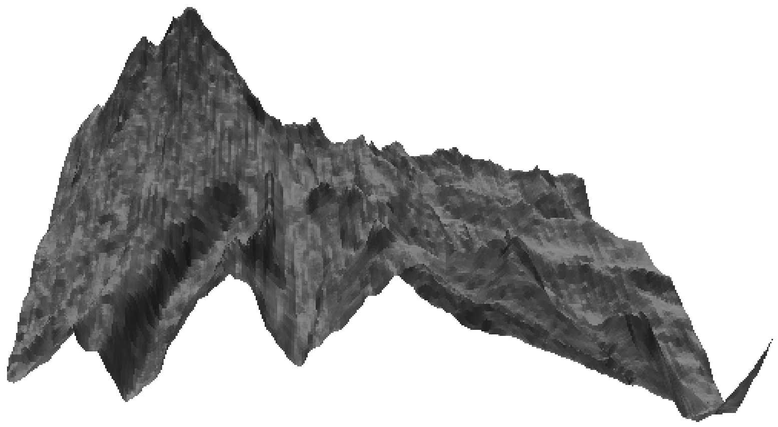


(c) Mesh

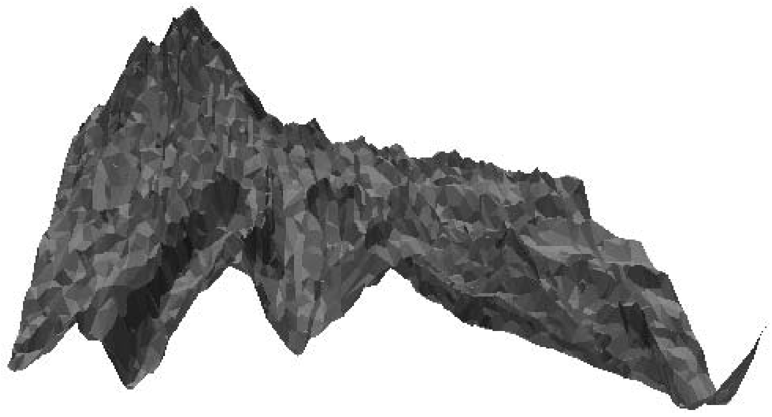
Figure 4.7: Greedy Insertion Simplified cbaker; 5000 vertices, 9826 faces



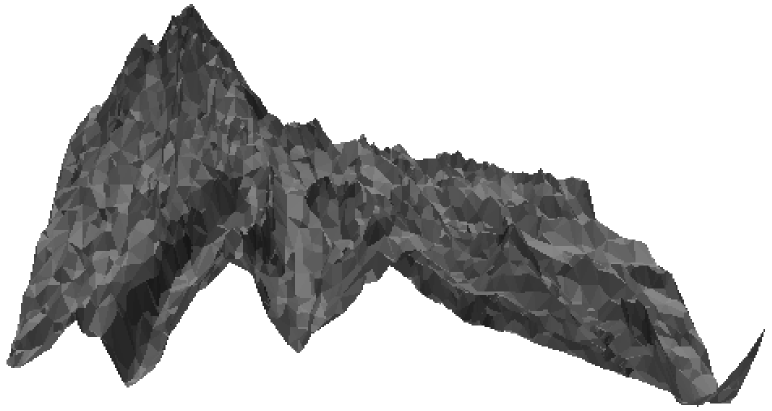
(a) FullModel



(b) Hybrid

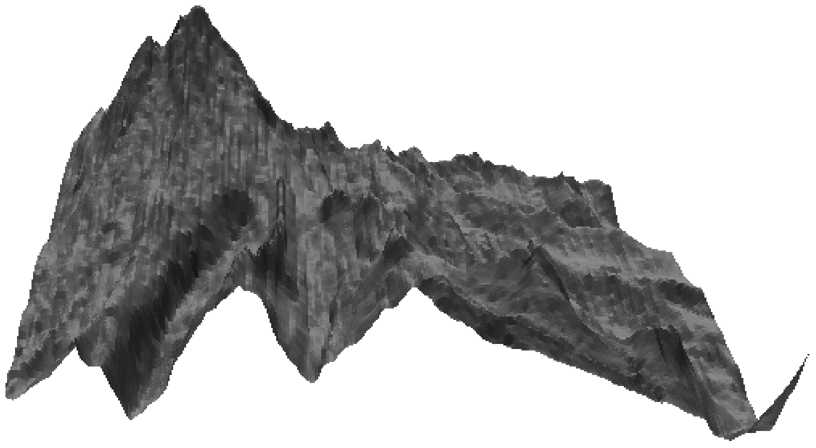


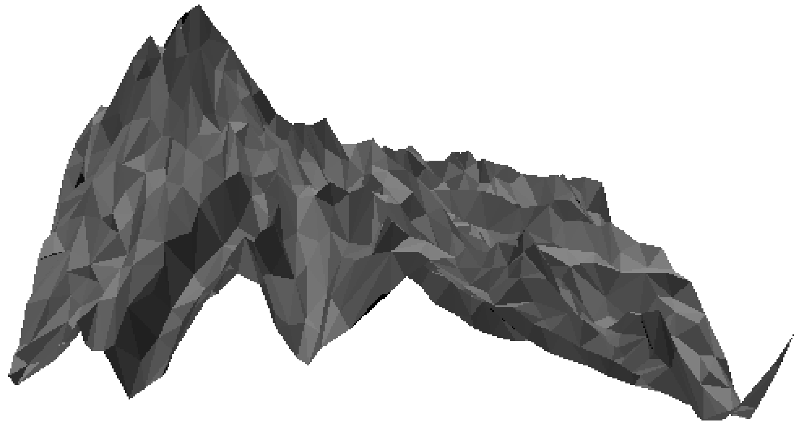
(c) QSlim



(d) Greedy

Figure 4.8: cbaker: Comparing all 5000 models


(a) FullModel


(b) Surface

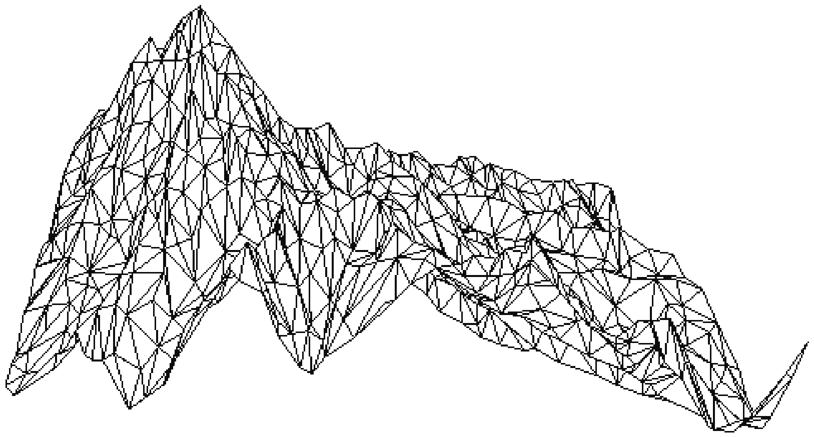
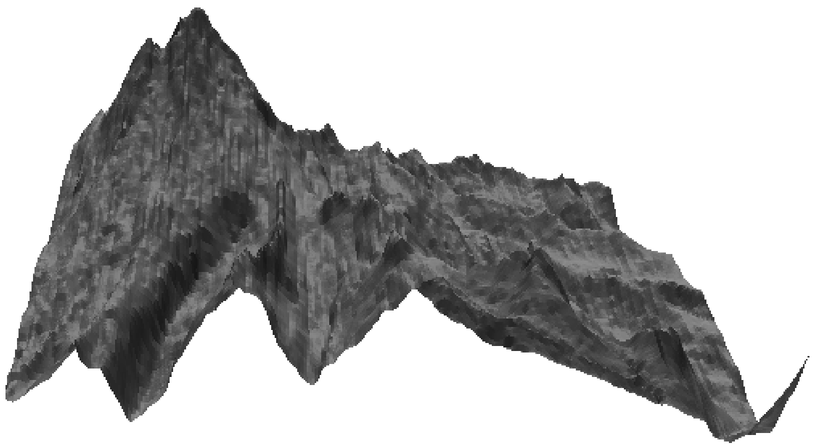
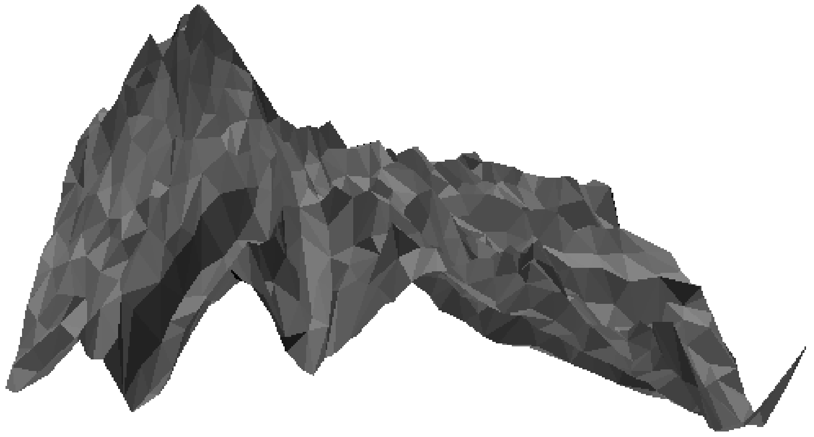

(c) Mesh

Figure 4.9: Hybrid Simplified cbaker; 1000 vertices, 1865 faces


(a) FullModel


(b) Surface

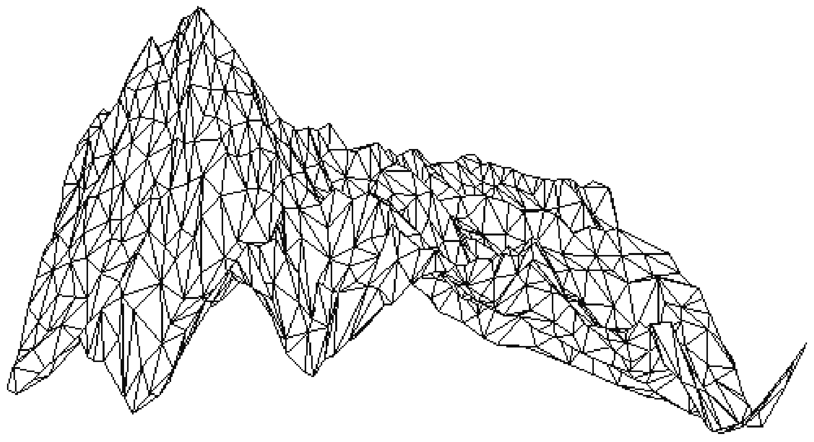
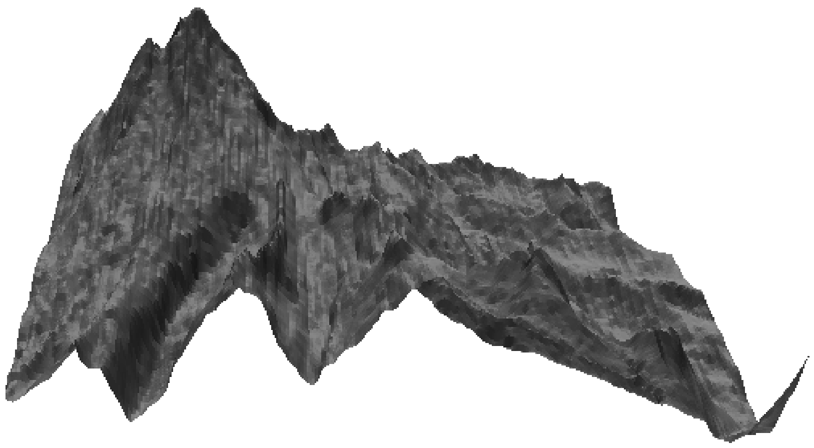
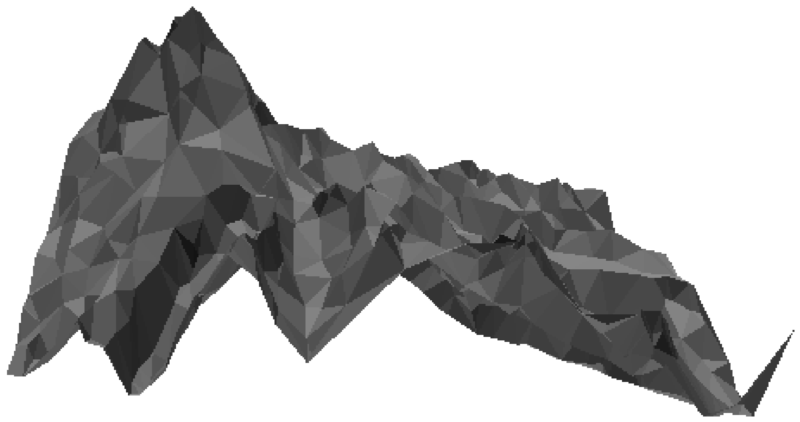

(c) Mesh

Figure 4.10: QSlim Simplified cbaker; 1000 vertices, 1805 faces


(a) FullModel


(b) Surface

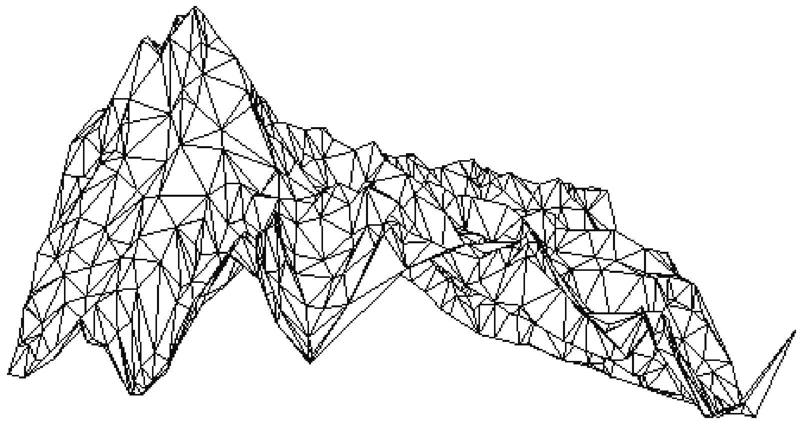
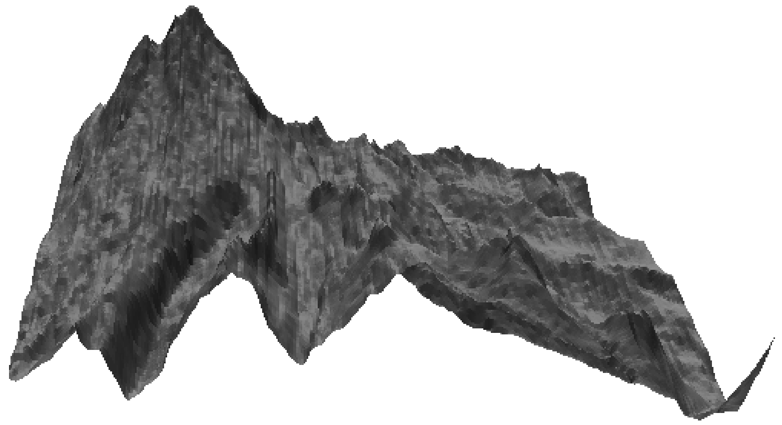
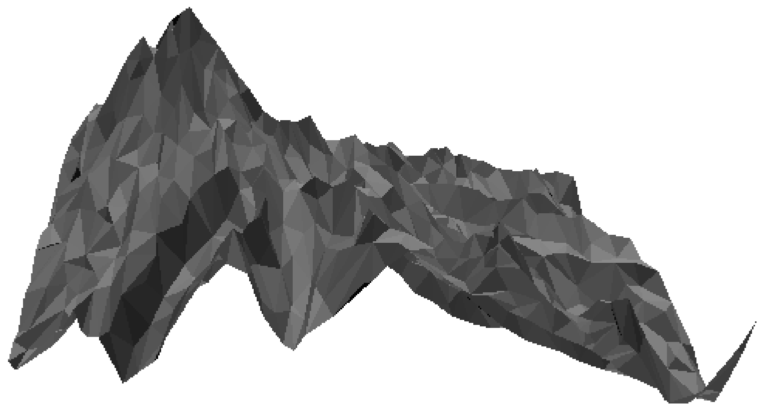

(c) Mesh

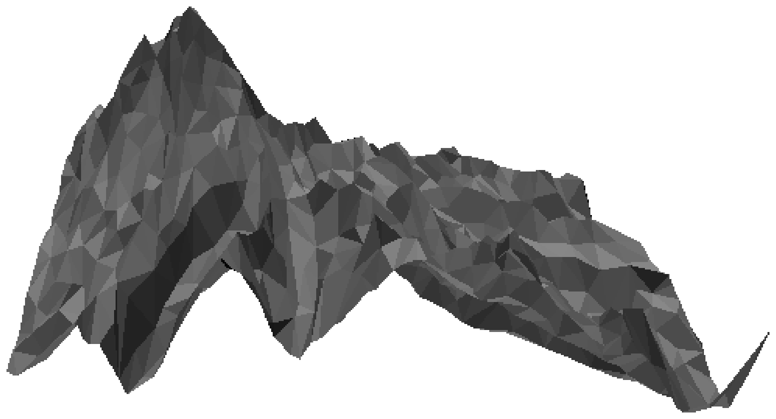
Figure 4.11: Greedy Simplified cbaker; 1000 vertices, 1927 faces



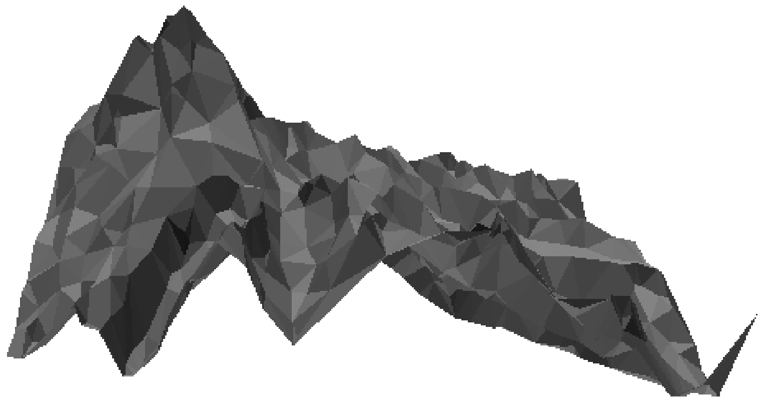
(a) FullModel



(b) Hybrid

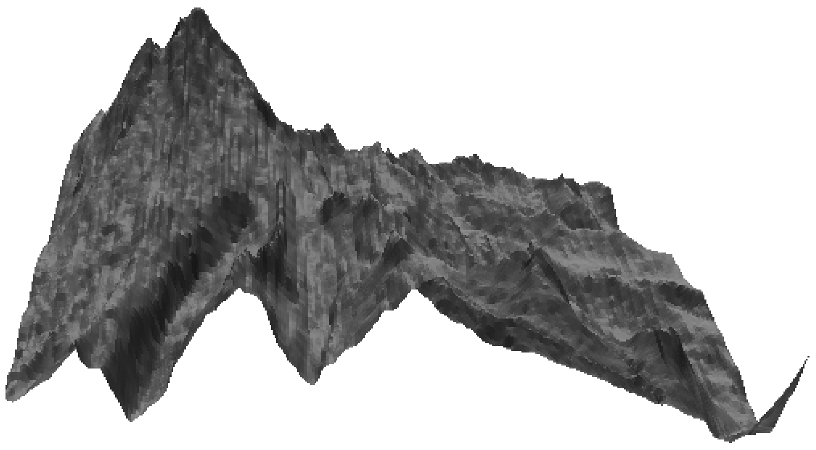


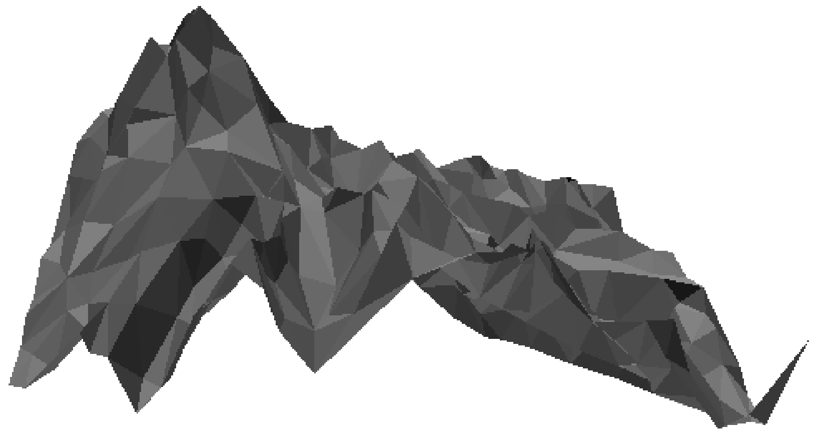
(c) QSlim



(d) Greedy

Figure 4.12: cbaker: Comparing all models with 1000 vertices


(a) FullModel


(b) Surface

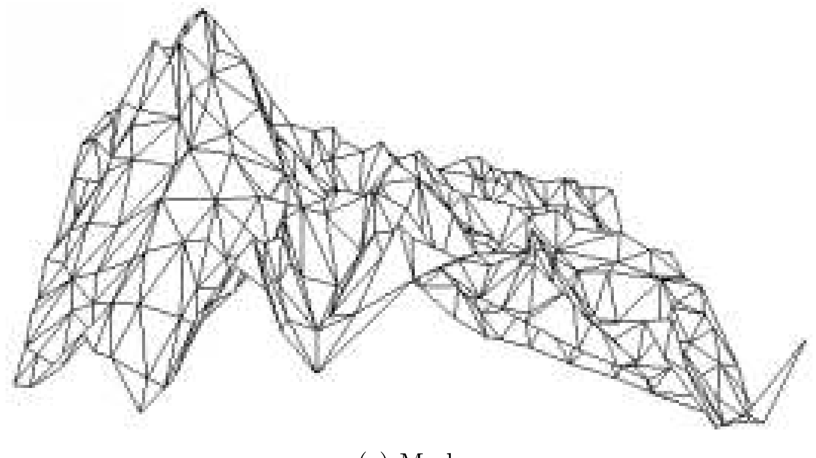
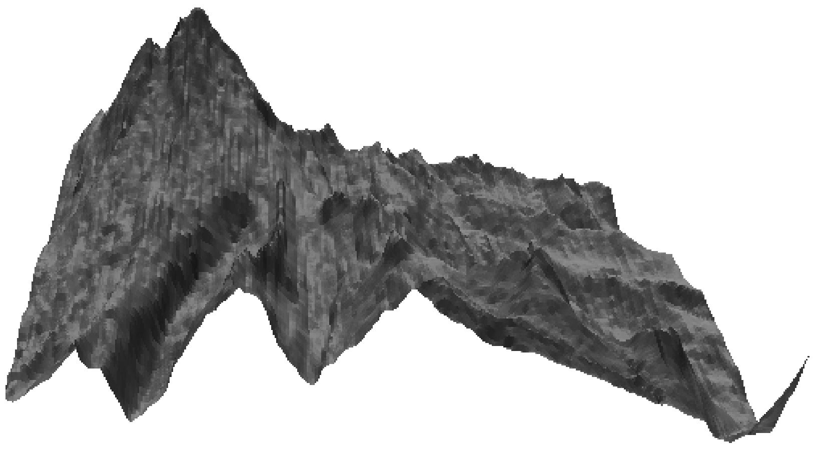
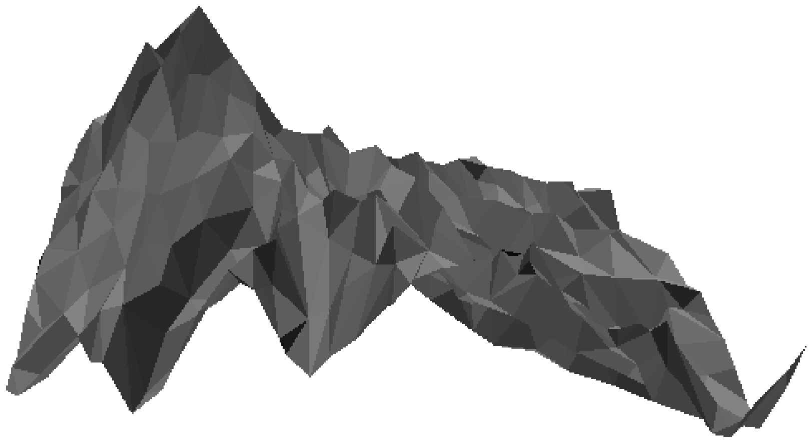

(c) Mesh

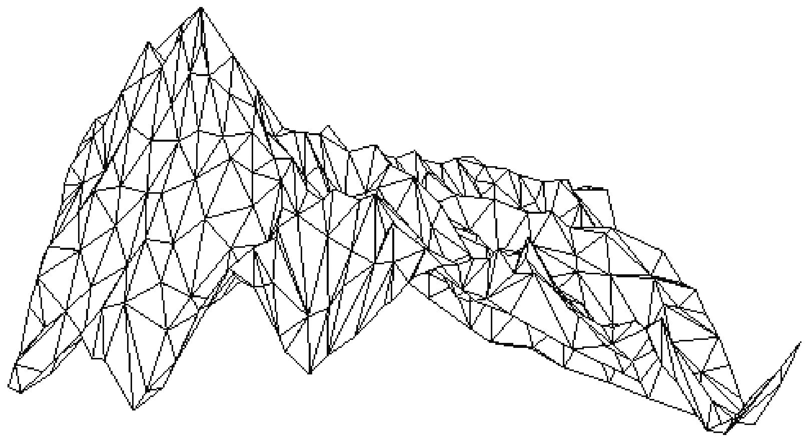
Figure 4.13: Hybrid Simplified cbaker; 500 vertices, 930 faces



(a) FullModel

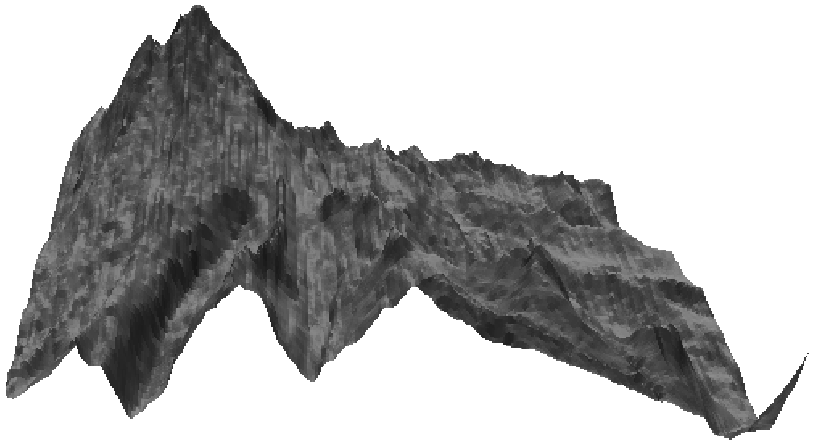


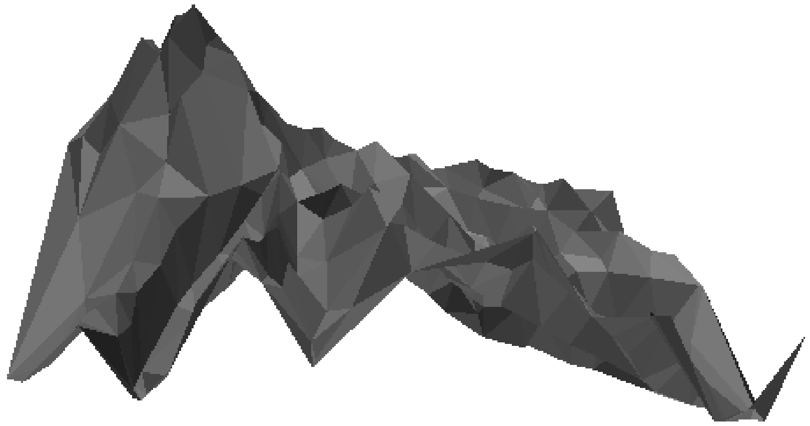
(b) Surface



(c) Mesh

Figure 4.14: QSlim Simplified cbaker; 500 vertices, 869 faces


(a) FullModel


(b) Surface

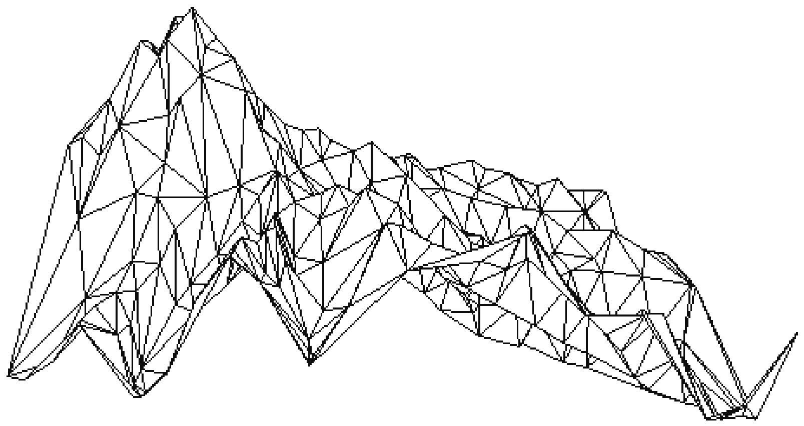
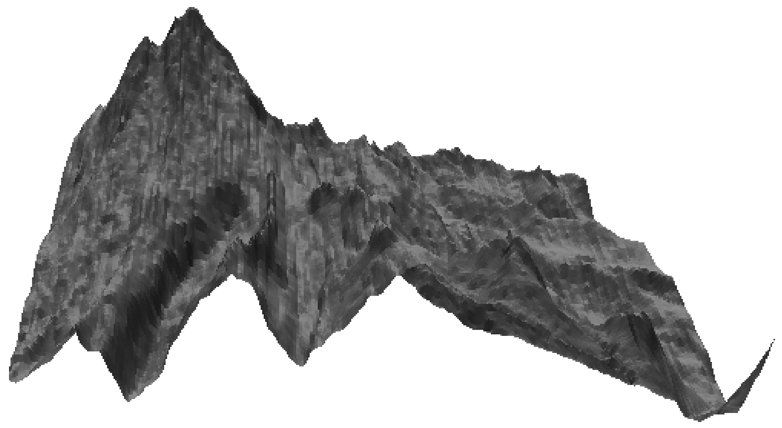
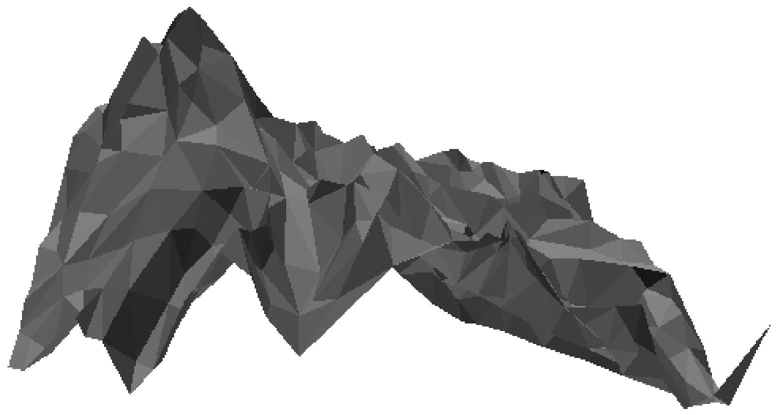

(c) Mesh

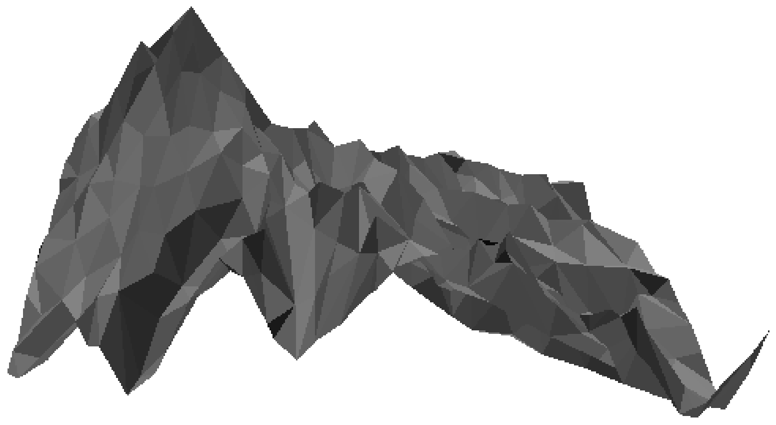
Figure 4.15: Greedy Simplified cbaker; 500 vertices, 954 faces



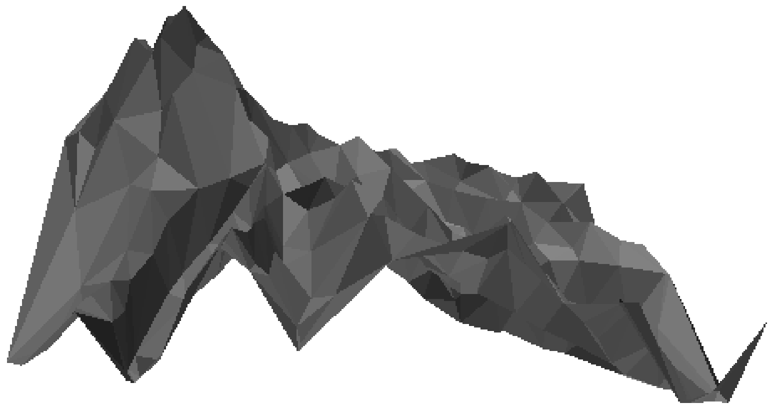
(a) FullModel



(b) Hybrid



(c) QSlim



(d) GreedyInsertion

Figure 4.16: cbaker: Comparing all models with 500 vertices

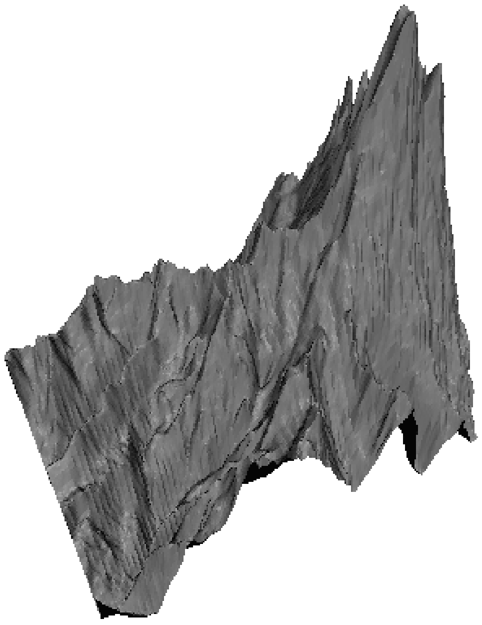
A similar result can also be derived from another model, as shown in the figures in the Appendix.

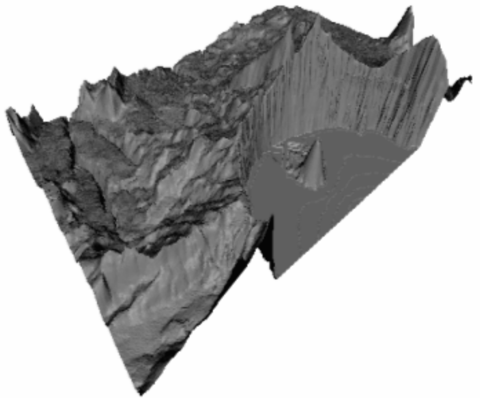
4.3.1 Geometric Error of Simplified Model

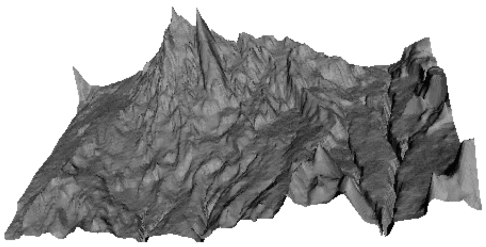
Visual similarity is only one aspect of fidelity, and it is very subjective and vague. Geometric error provides a more objective and more precise way to measure the fidelity of simplified models.

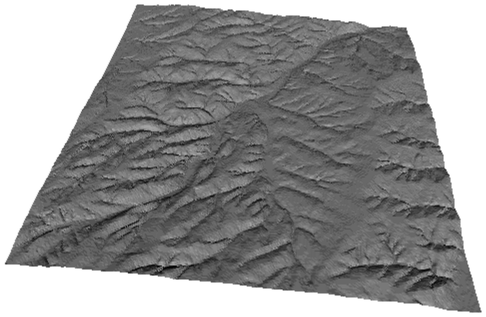
Some typical sample terrain models and their sizes are given in Figure 4.17. *cbaker* is a very jagged model. *crater* is a model full of different features such as valleys, peaks and lakes. *lcrater* is part of crater, only without the lake. *Ritz1* is very flat. *Yakima2* is a terraced model, and *yakima3* is a model that is mostly flat, but with a small number of high peaks with steep cliffs.

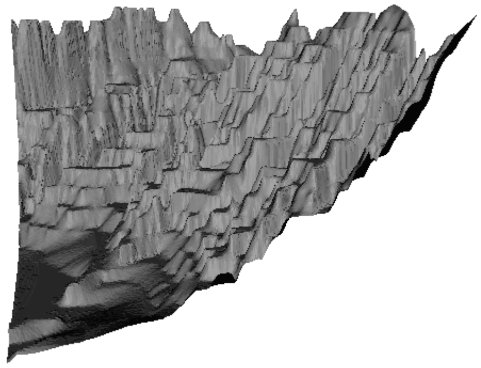
Table 4.1 gives a comparison of the VRMSs from the three methods for the model cbaker. Column “Target size” gives the size of the target models. The VRMSs from the hybrid algorithm are the smallest among the results of the three methods. Table 4.2 gives similar results for the terrain model crater. Table 4.3 gives a comparison of the VMAEs from the three methods for the terrain cbaker. The results of hybrid are the best for all the three tables.


(a) cbaker, 290x440


(b) crater, 336x459


(c) lcrater, 230x459


(d) ritz1, 300x300


(e) yakima2, 256x256

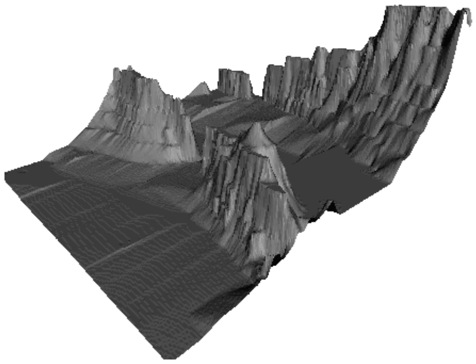

(f) yakima3, 256x256

Figure 4.17: Sample Terrain Models

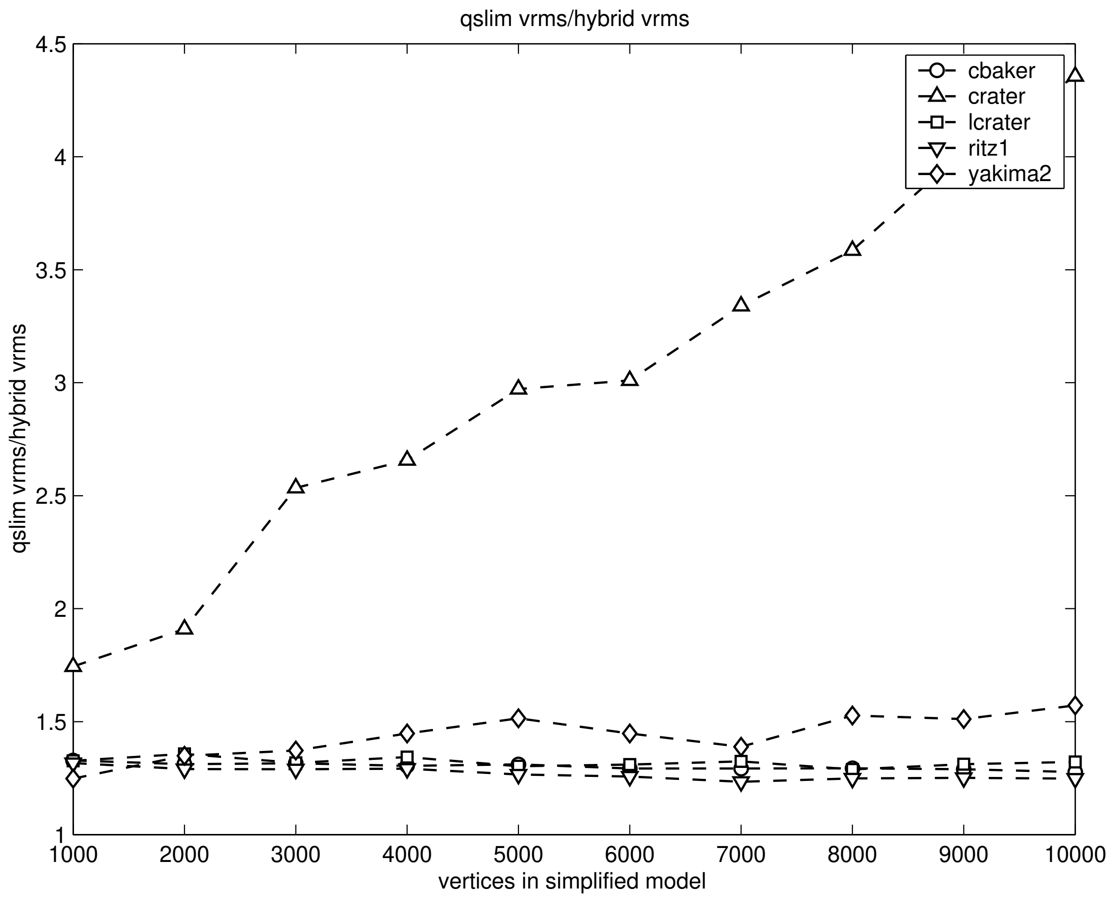


Figure 4.18: QSlim VRMS over improved Hybrid VRMS, 5 models

Target size	hybrid VRMS	QSlim VRMS	greedy insertion VRMS
1000	9.4299	10.6784	13.7980
2000	5.7039	6.5290	8.5539
3000	4.2354	4.7840	6.5259
4000	3.4276	3.8018	5.3646
5000	2.8636	3.1661	4.5771
6000	2.4956	2.7283	3.9834
7000	2.1976	2.4000	3.5104
8000	1.9645	2.1411	3.1738
9000	1.7837	1.9323	2.8832
10000	1.6301	1.7660	2.6300

Table 4.1: Comparing the VRMS of the cbaker model

Target size	hybrid VRMS	QSlim VRMS	greedy insertion VRMS
1000	5.1621	6.0398	7.0222
2000	3.2191	3.6889	4.5065
3000	2.3989	3.2581	3.3853
4000	1.9968	2.9158	2.7544
5000	1.7096	2.5659	2.3520
6000	1.5437	2.2677	2.0555
7000	1.3667	1.7380	1.8441
8000	1.2568	1.5869	1.6709
9000	1.1702	1.4903	1.5332
10000	1.0729	1.4049	1.4106

Table 4.2: Comparing the VRMS of the crater model

Let us define *error ratio* as the ratio of the VRMS of the QSlim result to the hybrid result. Figure 4.19 shows the error ratio of the VRMS of the simplified model produced by Qslim to that produced by the hybrid algorithm. The numbers on the x axis are the size of the simplified models, while the y axis gives the error ratio. Note that the y axis starts with coordinate 1.05, which means for all models, the result of the hybrid algorithm is superior to that of QSlim. Improvement varies

Target size	hybrid VMAE	QSlim VMAE	greedy insertion VMAE
1000	0.7450	0.8569	1.3890
2000	0.4622	0.5237	0.8715
3000	0.3552	0.3746	0.6645
4000	0.2865	0.3110	0.5635
5000	0.2434	0.2614	0.4835
6000	0.2123	0.2303	0.4210
7000	0.1898	0.2025	0.3677
8000	0.1681	0.1814	0.3343
9000	0.1543	0.1630	0.3064
10000	0.1414	0.1484	0.2822

Table 4.3: Comparing the VMAE of the cbaker model

with different models, ranging from 7.5% to 10 times less error. Figure 4.19 only gives the error ratios for five sample models; the ratio of *yakima3* reaches 10.00, much bigger than the other models. Figure 4.20 gives the error ratios of all six models. Great improvement is obvious for models with very huge cliffs, while the improvement for flat models with low curvature is relatively small.

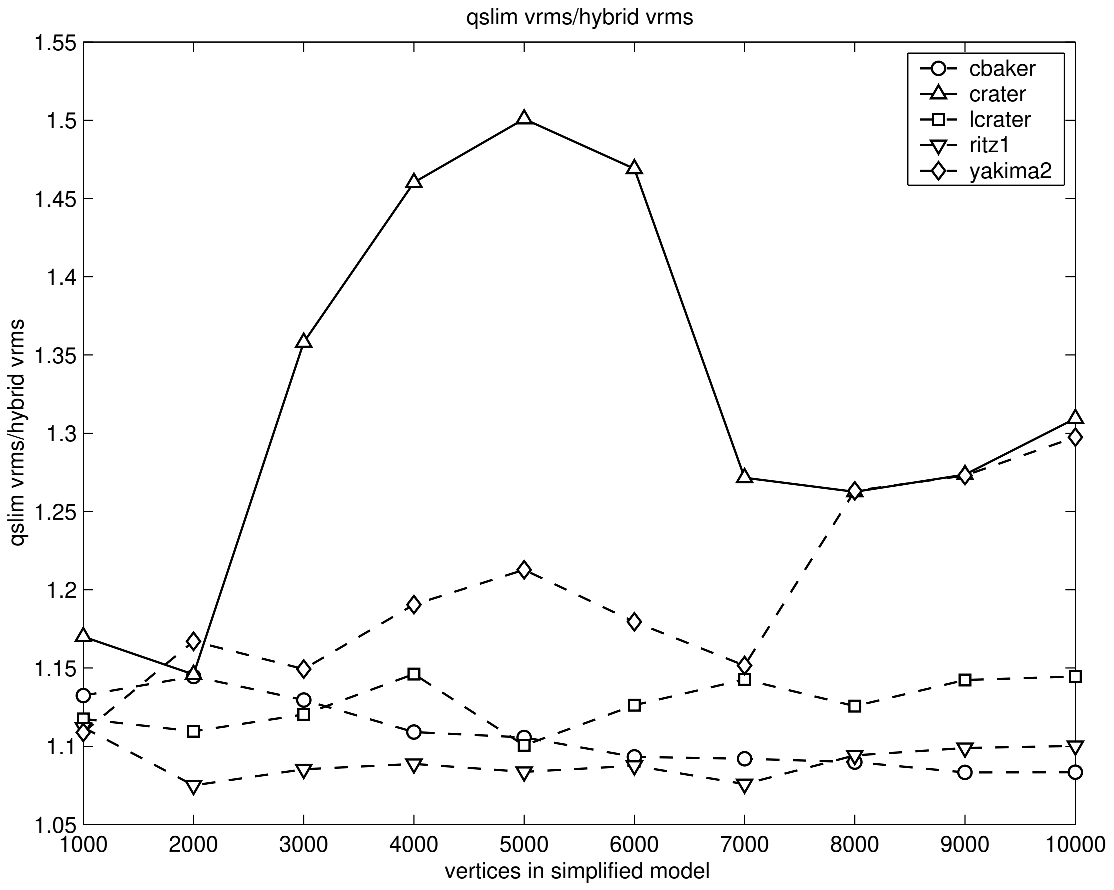


Figure 4.19: QSlim VRMS over Hybrid VRMS, five models

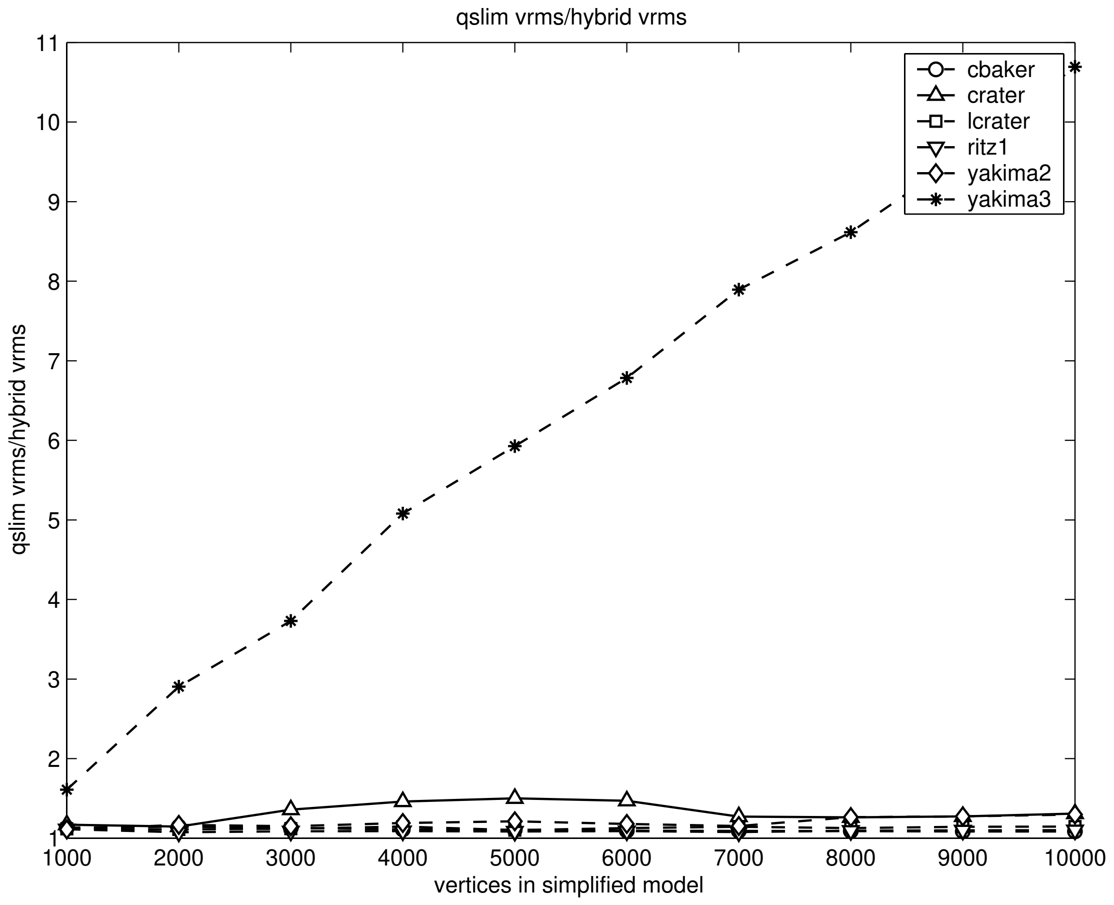


Figure 4.20: QSlim VRMS over Hybrid VRMS, six models

Figure 4.21 shows the error ratio of the VRMS of the simplified model produced by greedy insertion to that produced by the hybrid algorithm. The x axis and the y axis are set up the same as in Figure 4.19. The y axis starts at 1; the simplified models from the hybrid algorithm have smaller error than their corresponding error produced by greedy insertion; the improvement ranges from 5% to 80%. On the contrary to Figure 4.20, the two models with the highest improvement – yakima3 and crater – have the lowest improvement in Figure 4.21. The reason why this happens still needs to be explored.

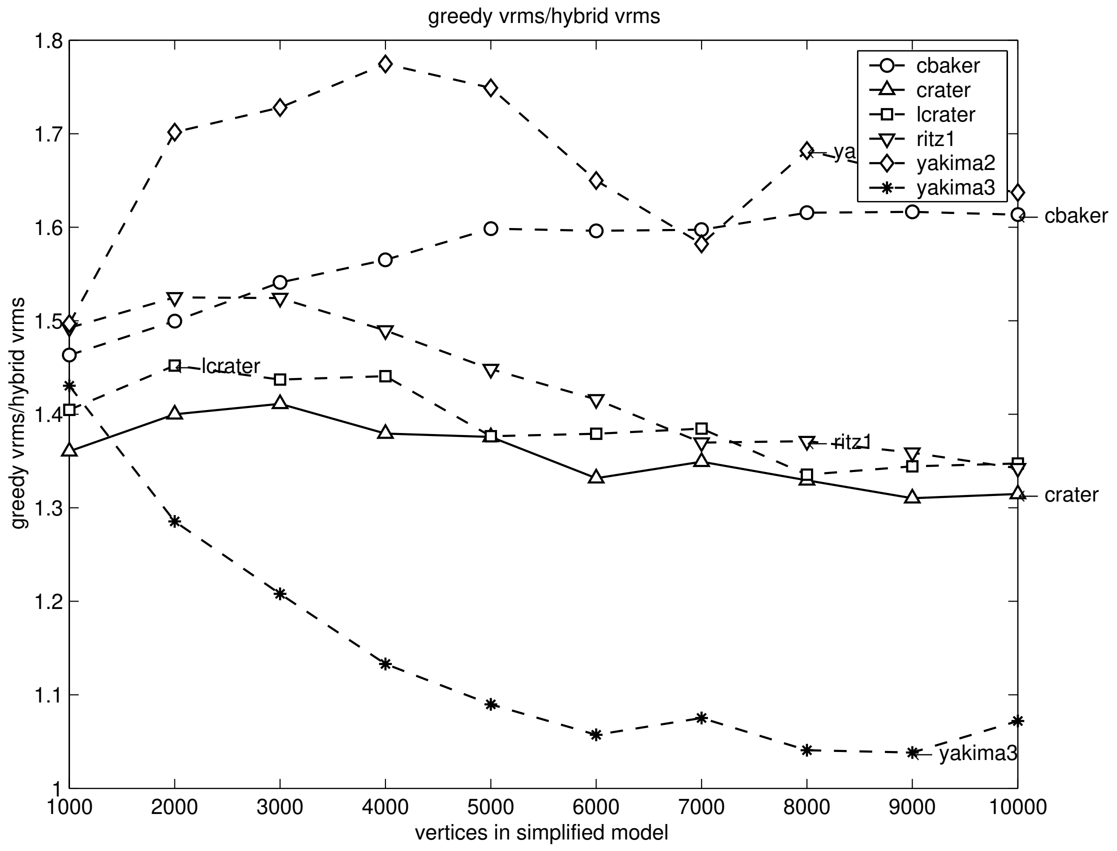
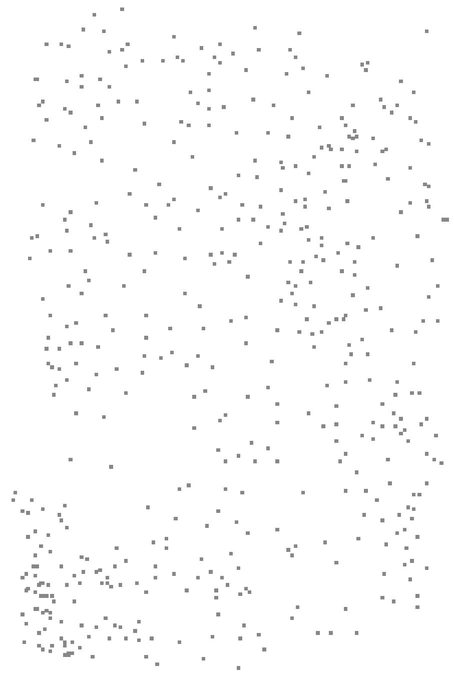
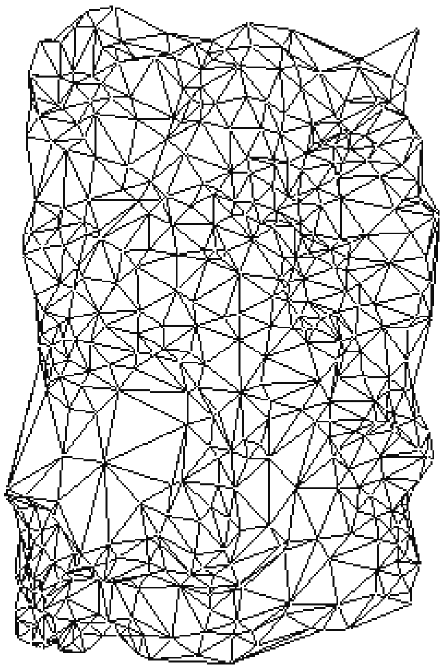


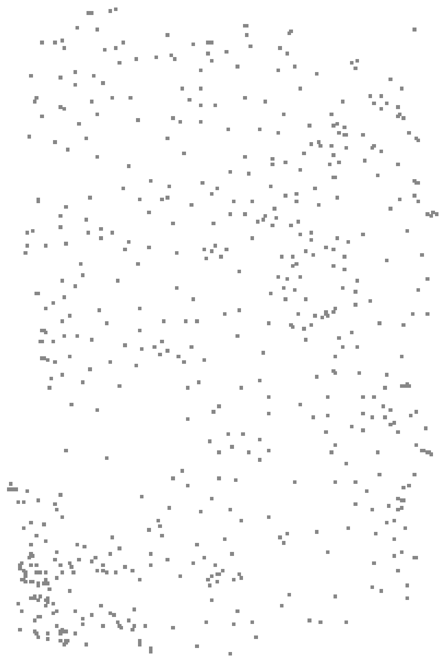
Figure 4.21: Greedy VRMS over Hybrid VRMS

4.3.2 Shrinking Effects

The hybrid algorithm has an overall better performance than the other two algorithms. We are curious about why the hybrid algorithm performs better than the other two. Figure 4.22 gives a comparison between two point sets of the cbaker model: one is before shrinking, the other is after shrinking. *a* is the point set shrunk from a point set of size 600 from the Greedy method, which is shown in subfigure *c*, using QSlim; *b* and *d* are their corresponding meshes. A more detailed comparison is given in Figure 4.23. *a* gives the original point set before shrinking; vertices are represented by black dots. *b* plots out the two point sets in the same picture with a grey background, with black dots representing points before shrinking, white dots representing points after shrinking. Comparing these two sets of points, we can see that Qslim only makes changes where there are small triangles, i.e, where edges are short and vertices are densely distributed. Similarly, Figure 4.24 compares the greedy point set with the hybrid point set of the size 500 and Figure 4.25 compares the QSlim point set with the hybrid point set. Figure 4.26 gives the three point sets derived from the three methods with size 500. In the Appendix, color pictures showing the first point set, the second point set and their interaction are given in Figure B.1. From those pictures, we can clearly see the difference between the point sets. Compared to the Greedy method, the hybrid method tends to balance the distribution of vertices. It removes some parts of greedy points in dense areas, and adjusts the position of some greedy points to give a more balanced representation of the original model. Compared to the Greedy method, the vertices of QSlim are more evenly distributed. The hybrid method alters the QSlim point set in some parts of the terrain such as areas with acute elevation changes, which possibly leads to a more accurate represented terrain representation.


(a) hybrid600-500points


(b) hybrid600-500mesh


(c) greedy600points

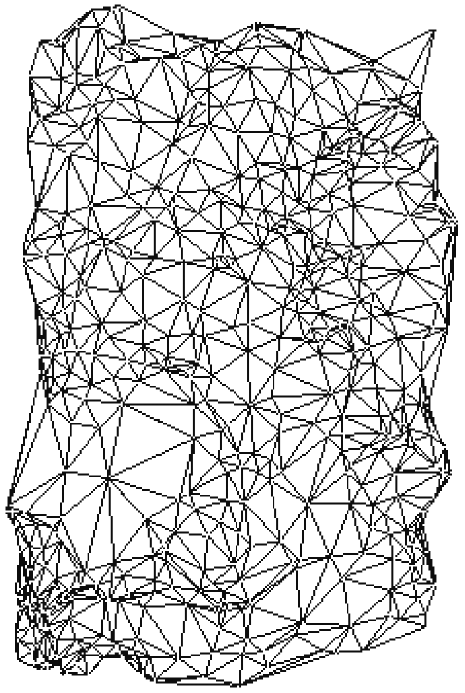
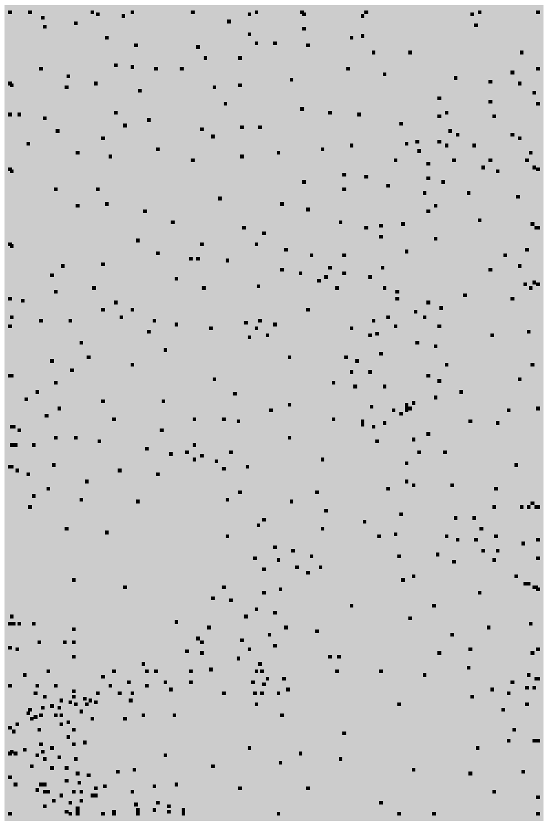

(d) greedy600mesh

Figure 4.22: Shrinking Points


(a) cbakergreedy600points

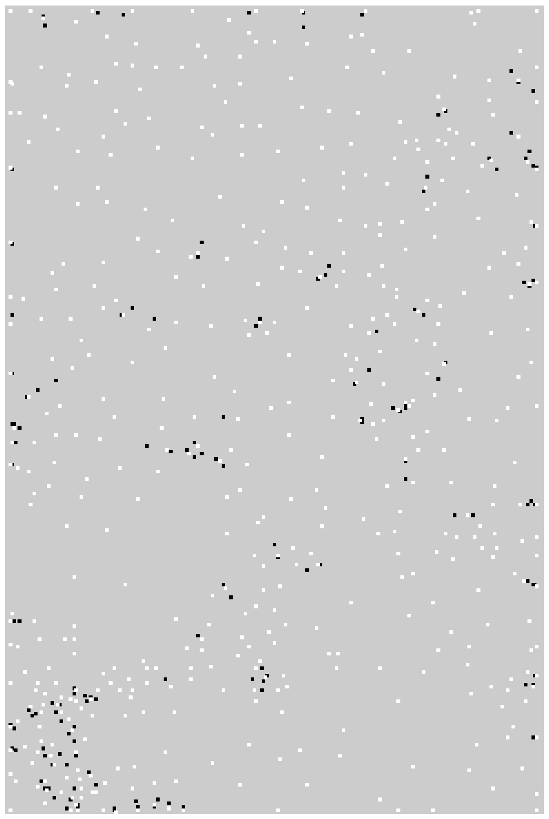
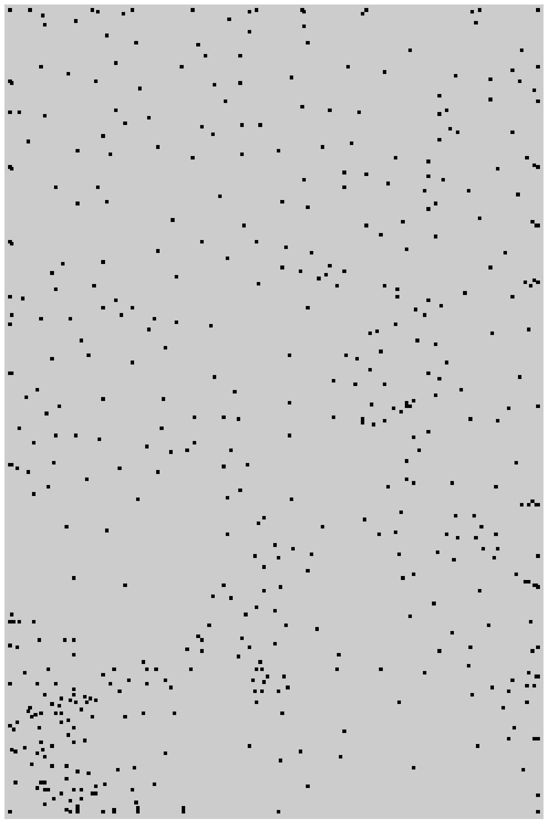

(b) cbakerhybrid600-500points

Figure 4.23: Comparing Shrinking Parts


(a) cbakergreedy500points

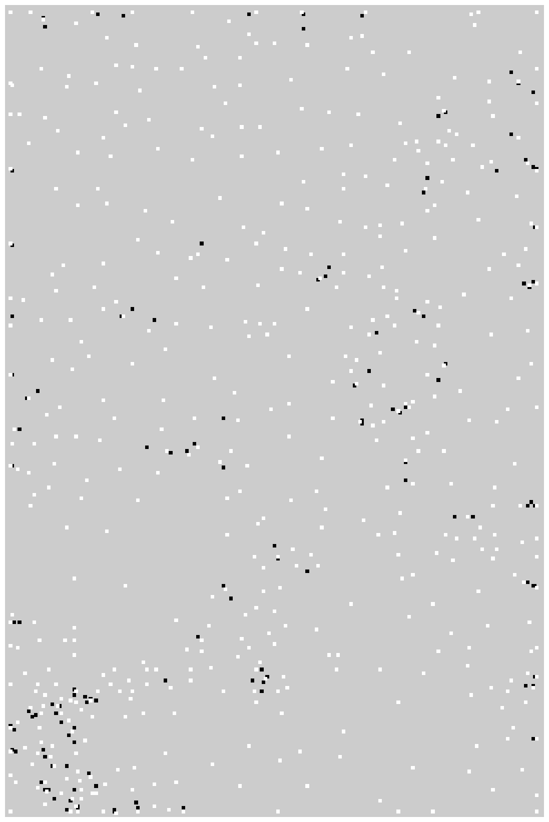
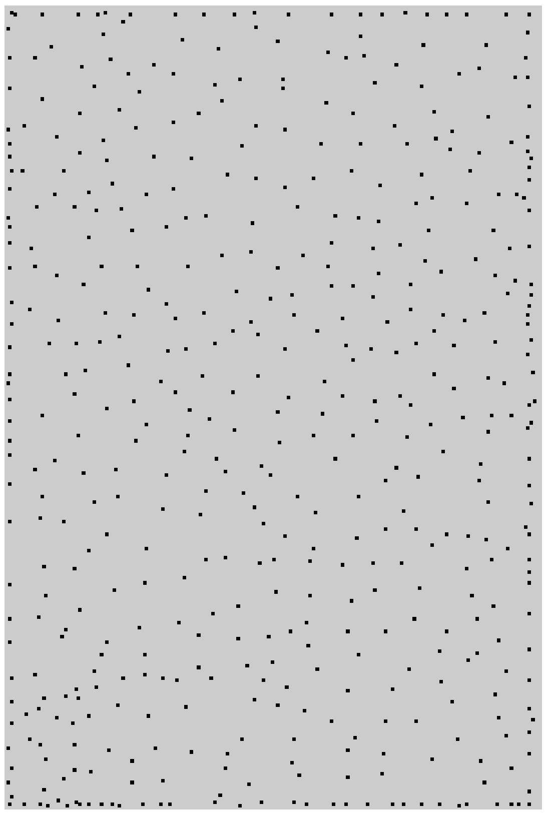

(b) cbakerhybrid600-500points

Figure 4.24: Comparing greedy points and hybrid points


(a) cbakerqslim500points

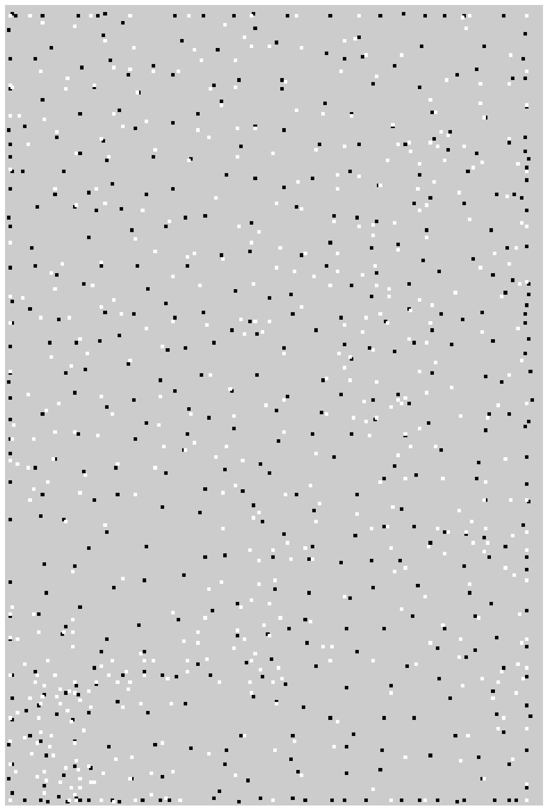
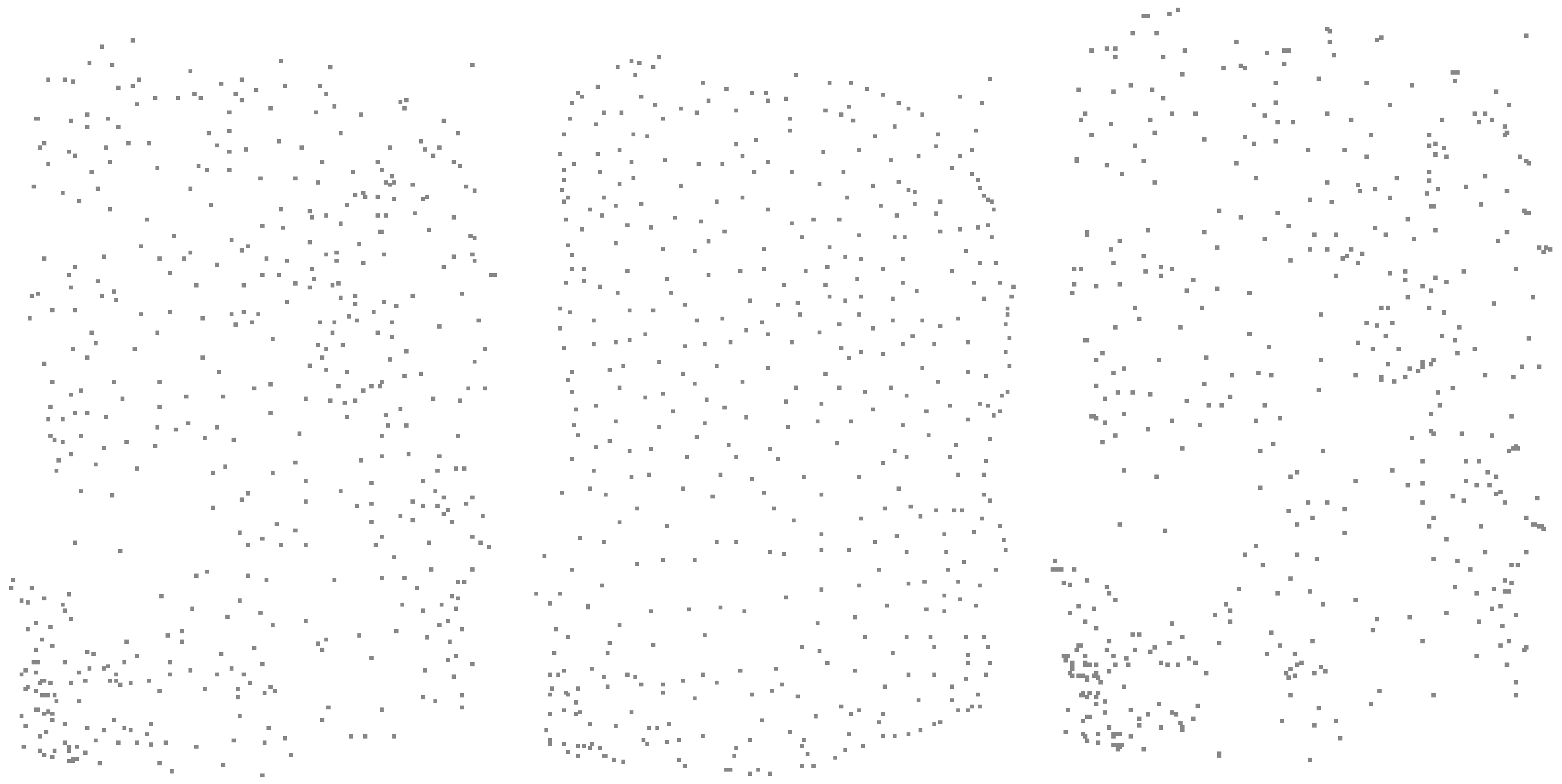

(b) cbakerhybrid600-500points

Figure 4.25: Comparing QSlim points and hybrid points



(a) cbakerhybrid600-500points (b) cbakerqslim500points (c) cbakergreedy500points

Figure 4.26: Comparing three points set of cbaker

4.3.3 Effect of the Size of Middle-Stage Model

As we mentioned before, the hybrid algorithm uses a two-stage process. Instead of constructing the target model T with t vertices directly, a middle-stage model M with m ($m > t$) vertices is first constructed. Therefore, the choice of the size m of the middle-stage model T is a natural concern.

Figure 4.27 shows the relation between the candidate size of the middle stage model and the VRMS of the target model. Analysis of other terrain models also gives similar results. The x axis shows the size of middle stage models; the number on each curve is the size of the target model; the y axis gives the VRMS of the target model. In Figure 4.27, the VRMS error becomes stable after the size of middle-stage model grows big enough; usually twice of the target size is enough. Therefore, the cost of greedy insertion is bounded. Therefore, good VRMS can be achieved without using a big size original model, which is also why the hybrid algorithm can be superior to QSlim.

4.3.4 Effect of Candidate Selection Strategy

In Section 3.1.3, different error measures for triangles have been discussed; they are $M2$, $V2$, $V3$, $V4$, $V5$, $V6$, $V7$, $V8$, $V9$ and $V10$. These error measures directly affect the way candidate points are chosen, because the triangle is the basic unit to insert new points as the model grows.

In Section 3.1.3, I conjectured that $M2$, $V4$, $V5$, $V8$ and $V9$ would have better performance than other measures. Which one is the best among $M2$, $V4$, $V5$, $V8$ and $V9$ is still unknown. From Figure 4.28, $V8$, $V9$ and $M2$ are found to be the best ones. Because VRMS is used as the error criteria to measure the accuracy of approximated models, it is not surprising that they perform the best, since $V8$ and $V9$ tend to choose insertions that favor decreasing deviation, which is directly related to VRMS. Figure 4.29 gives clearer details between the three best measures. It is hard to tell which is the best from this figure. $V9$ seems to have overall the

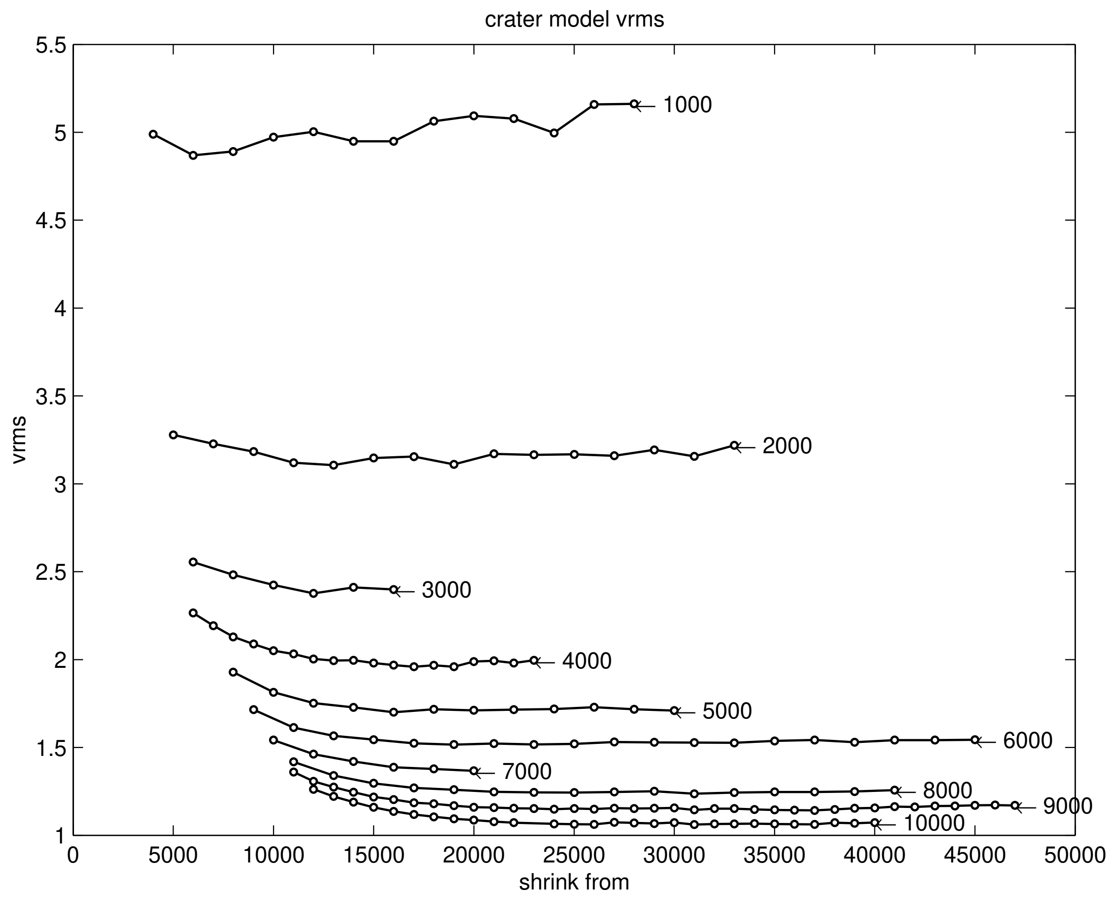


Figure 4.27: Crater: the Relation between VRMS and the size of Middle-Stage Model

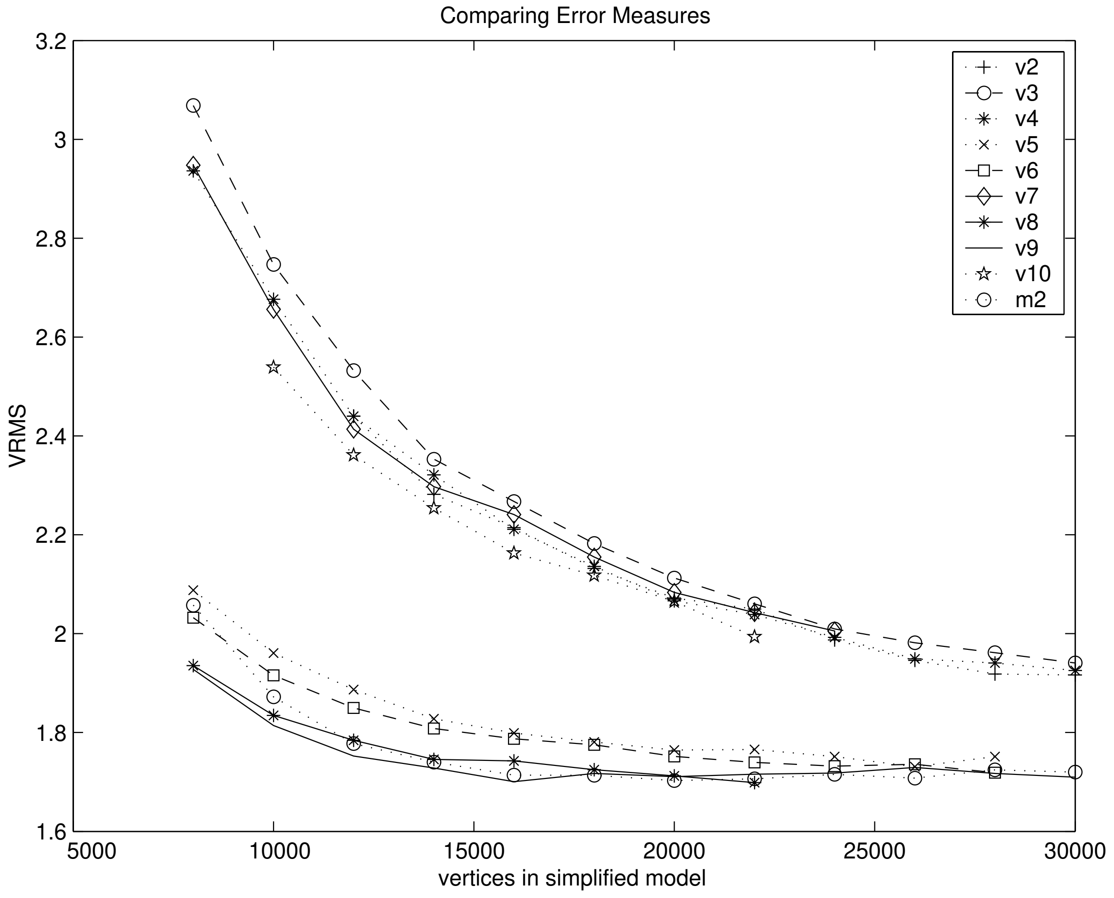


Figure 4.28: Crater: Comparing Different Error Measures

best quality. The hypothesis is that $V9$ tends to build a more balanced surface, which will help greedy insertion to chose the possible most appropriate candidate point to insert. Little and Shi have given a more detailed comparison and much more experimental results [LS01b].

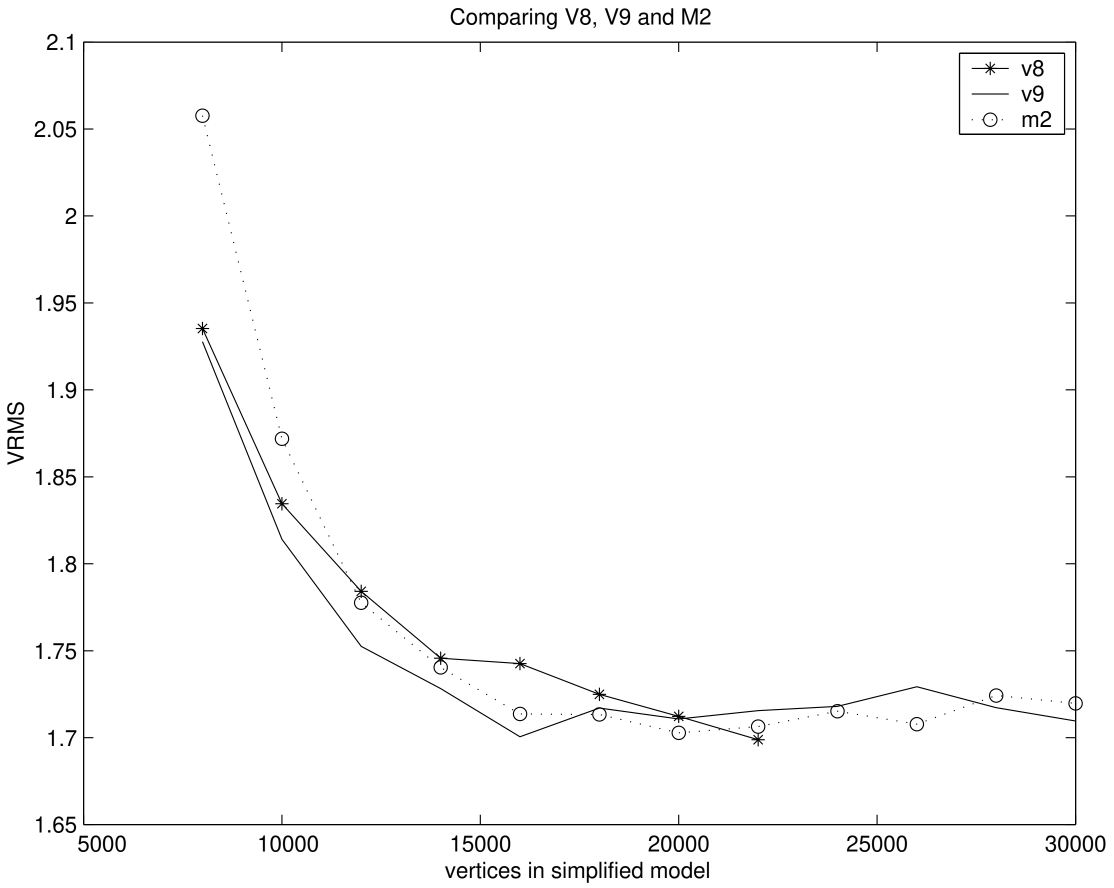


Figure 4.29: Crater: Comparing the Three Best Error Measures

Chapter 5

Conclusion and Future work

5.1 Summary of Work

I have given a hybrid algorithm which integrates greedy TIN generation with quadric simplification. Experiments prove that the hybrid algorithm improves the overall approximation performance over both previous algorithms in both geometric error and visual fidelity, and also in time compared to quadric simplification when using a small approximation.

5.2 Future Work

5.2.1 Exploring Other Combined Strategy

Intuitively, the last two combined strategies we mentioned in Section 3.3 should be effective. However, it is mysterious that they do not seem to be so. Due to our tight schedule, we did not find the reason, which could be very helpful for choosing a combined strategy.

5.2.2 Alternative Hybrid Strategy

The current hybrid algorithm is a two-step process. For a target model T with t vertices, a middle-stage model M with m ($m > t$) vertices is first constructed, then shrunk to the target model T. One alternative is to do this in the reverse direction. First, shrink the full model DEM D with n vertices to a middle stage model M with m vertices ($m < t$), then use greedy insertion to grow the model M to the target model T. The early insertions of the greedy insertion algorithm are very costly, because they need to scan large areas of terrain in order to select candidates and update errors. With this construction order, when the greedy insertion begins, the terrain region has been divided into many small regions, so the greedy insertion can avoid scanning large areas and the running time would speed up. In the first strategy, we are saving time for QSlim. With this alternative, we are saving time for greedy insertion. Because QSlim runs much faster than greedy insertion when the target model is large, we would expect a large improvement in the time performance for the hybrid algorithm. We can combine these two strategies together: if the target model is small, use $greedy - qslim$; if the target model is large, use $qslim - greedy$. Whether this strategy helps improve the approximation quality still needs to be explored.

5.2.3 Exploiting Terrain Features

Little and Shi have studied how to exploit terrain features in terrain simplification in [LS01c]. They have shown structure lines and feature points help improve the accuracy of approximated models. The hybrid algorithm inherits the ability to include structure lines and feature points into an approximation model as in their work. However, surprisingly, using structure lines and feature points does not improve performance in the hybrid algorithm, and in most cases the results are worse. We are very eager to explore how structures lines and feature points and their selection affect the hybrid algorithm. If the problem can be identified, then the performance of the hybrid algorithm can be further improved.

5.2.4 Improve Robustness in the Presence of Noise

Both greedy insertion and quadric decimation have no mechanism to deal with noise. Both methods simply treat noise as surface features. In [Gar99a], Garland has shown that approximations from quadric decimation degrade gracefully for noisy models. But greedy insertion certainly does not do well on noisy models. It simply picks the point with highest error and inserts it into current mesh, without considering whether it is noise or a real data item. Furthermore, it uses the plane decided by the three vertices of the triangle to approximate the local surface. Therefore, a noise datum can survive in the approximation and has no chance to be found. To improve the tolerance of the hybrid algorithm to noisy data, prefiltering of the surface model and better plane selection to fit a set of points located in the triangle will be needed.

5.2.5 Supporting Multi−Resolution Represention

As mentioned in Chapter 2, view dependent LOD approximations (multi−resolution represention) have been very popular in recent years because they produce faster and better approximations for a certain view point. They provide a more balanced solution to address the requirements of current computer rendering applications – high frame rate and acceptable image quality.

The greedy insertion algorithm does not naturally support a multi-resolution representation. However we can divide a big terrain into many smaller blocks, compute multiple levels of detail for each of them, and then combine them to get a multi−resolution representation for a given view point. As for QSlim, Garland[Gar99a] has proposed to use vertex hierarchies to address the problem of multi-resolution representation, similar to the methods in [Hop98, Hop97, KT96, XV96].

In order to make the hybrid algorithm support multi−resolution representation, we can still use the solution mentioned above for greedy insertion. However, this method is not as flexible as progressive mesh and vertex hierarchies methods. How to improve the hybrid algorithm to flexibly support multi−resolution representation still needs to be explored.

5.2.6 Exploiting Silhouettes to Improve the Rendering Quality

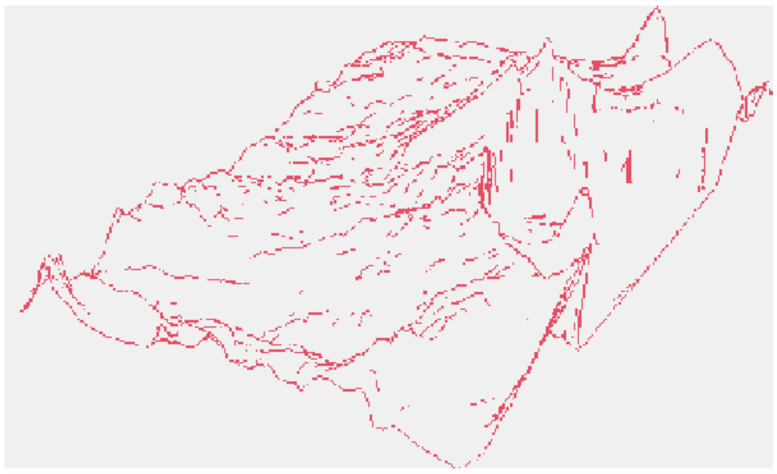
For polygonal meshes, the silhouette is defined as the set of all edges that connect back-facing polygons to front-facing (possibly visible) polygons. For a smooth surface, the silhouette can be defined as those surface points x_i with a surface normal n_i perpendicular to the view vector:

$$n_i \cdot (x_i - C) = 0$$

where C is the camera center.

Due to the properties of human perception, a detailed silhouette in a coarsely modeled surface may be sufficient for recognition and navigation purposes for certain applications such as flight simulation. Therefore, if accurate silhouettes for terrain models can be extracted in realtime, combining them with approximated models will improve the visual fidelity, or at least visual recognizability for rendering applications. Silhouettes may also help guide and enhance view-dependent simplification.

Basically there are two categories of algorithms to extract silhouettes. One category is rendering-based silhouette extraction, and a typical work is by Raskar and Cohen in [RC99]. Because this method needs to render the whole image to get the silhouette edges, it will not help up to improve the accuracy of the silhouettes of approximated models because actually the approximated models instead of the original full models will be rendered. Only the silhouette of the original full model helps improve the rendering results of a coarse model. The second category is geometry based silhouette extraction. A typical work is by Barequet et al. in [BDG+99]. Neat silhouettes can be extracted, as shown in Figure 5.1. Their algorithm is fast, but for huge terrain models, realtime extraction remains a problem.


(a) Silhouette one for crater

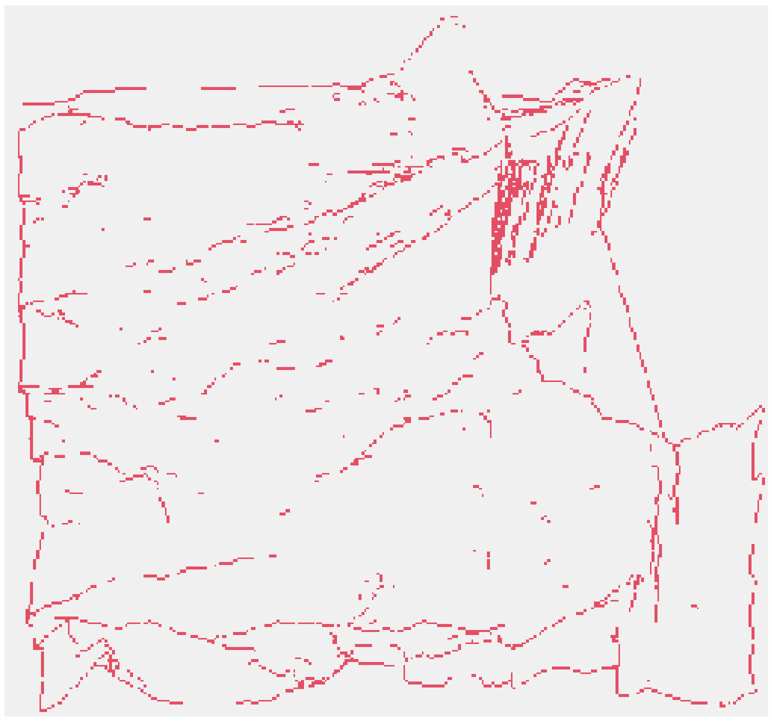

(b) Silhouette two for crater

Figure 5.1: Two Silhouettes for crater

Therefore, realtime silhouette extraction, silhouette guided simplification and rendering are open problems that deserve further research.

Bibliography

[AD97] Pankaj Kumar Agarwal and Pavan Kumar Desikan. An efficient algorithm for terrain simplification. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 139–147, Jan 1997.

[AS] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 24–33.

[BDG$^+$99] Gill Barequet, Christian A. Duncan, Michael T. Goodrich, S. Kumar, and M. Pop. Efficient perspective-accurate silhouette computation. In *Symposium on Computational Geometry*, pages 417–418, 1999.

[BFM95] Michela Bertolotto, Leila De Floriani, and Paola Marzano. Pyramidal simplicial complexes. Symposium on Solid Modeling and Applications, 1995.

[BMKN02] Boaz BenMoshe, Joseph S. B. Mitchell, Matthew J. Katz, and Yuval Nir. Visibility preserving terrain simplification an experimental study. In *ACM Symposium on Computational Geometry*, pages 303–311, Barcelona,Spain, June 2002. SoCG.

[CE01] David Cline and Parris K. Egbert. Terrain decimation through quadtree morphing. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7, pages 62–69. IEEE, 2001.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. In *CACM*, volume 19 of *10*, pages 547–554, 1976.

[Cla87] K. L. Clarkson. New application of random sampling in computational geometry. *Discrete and Comput. Geom.*, 2:195–222, 1987.

[CMS97] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. Technical report, Istituto di Elaborazione dell' Informazione-Consiglio Nazionale delle Ricerche, 1997.

[con] http://sputnik.dpi.inpe.br/teoria/mnt/mnteng.htm.

[dBD95] Mark T. de Berg and Katrin T. G. Dobrindt. On levels of detail in terrains. Technical report, Department of Computer Science, Utrecht University, April 1995.

[Del34] Boris N. Delaunay. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvenny ka Nauk 7*, pages 793–800, 1934.

[DEM] http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/u06.html/unit6.

[DLG] http://tahoe.usgs.gov/dlg.html.

[DWS$^+$97] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *8th IEEE Visualization '97 Conference*, page 81, Phoenix, AZ, October 1997.

[EKT01] William S. Evans, David G. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. *Algorithmica*, 30(2):264–286, 2001.

[FFP83] Leila De Floriani, Bianca Falcidieno, and Caterina Pienovi. A delaunay-based method for surface approximation. In *Eurographics '83*, pages 333–350, 1983.

[FL79] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. In *SIGGRAPH '79 Proceedings*, volume 13, pages 199–207, August 1979.

[Gar99a] Michael Garland. *Quadric-Based Polygonal Surface Simplification.* PhD thesis, Carnegie Mellon University, May 1999.

[Gar99b] Michael Garland. *Quadric-Based Polygonal Surface Simplification.* Ph.d dissertation, Computer Science Department, Carnegie Mellon University, May 1999.

[Ger03] Thomas Gerstner. Multiresolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, March 2003.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

[GS] M. Ghodsi and Jrg-Rdiger Sack. A coarse grained parallel solution to terrain simplification.

[GS85] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):75–123, 1985.

[GS95] Michael Garland and Paul S.Heckbert. Fast polygonal approximation of terrains and height fields. TechReport CMU-CS-95-181, School of Computer Science,Carnegie Mellon University, 1995.

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. *Computer Graphics*, 27(Annual Conference Series):19–26, 1993.

[Hel90] M. Heller. Triangulation algorithms for adaptive terrain modeling. In *Proc. 4th Internat. Sympos. Spatial Data Handling*, pages 163–174, 1990.

[HG97] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In SIGGRAPH 97 Course Notes: Multiresolution Surface Modeling, 1997.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Proc.*, pages 189–198, 1997.

[Hop98] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *In IEEE Visualization 98 Conference Proceedings*, pages 35–42, 1998.

[JS98] Bernd Jüenger and Jack Snoeyink. Selecting independent vertices for terrain simplification. In *WSCG '98*, pages 157–164, Plzen, Cz, 1998.

[KLYP00] Y. Kang, T. Lee, S. Yang, and W. Park. A fast digital terrain simplification algorithm with a partitioning method. In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, volume 2, page 613. IEEE, Beijing, China, May 2000.

[KT96] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), 1996.

[LKR$^+$96] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH'96*, pages 109–118, 1996.

[LP02] P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. Report, Lawrence Livemore National Laboratory, May 2002.

[LS01a] Peter Lindstrom and Claudio T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001*, pages 121–126, October 2001.

[LS01b] James J. Little and Ping Shi. Ordering points for incremental tin construction from dems. *Geoinformatica*, 2001.

[LS01c] James J. Little and Ping Shi. Structural lines, tins and dems. *Algorithmica, Special Issue on Algorithms for Geographical Information*, 30(2):243–263, 2001.

[Lue01] David P. Luebke. A developers survey of polygonal simplification algorithms. In *IEEE Computer Graphics and Applications*, May 2001.

[Mar78] David M. Mark. Concepts of "data structure" for digital elevation models. In *Proceedings, American Society of Photogrammetry, Digital Terrain Models Symposium*, pages 24–31, St.Louis, Missouri, May 1978.

[MP97] David M. McKeown and Michael F. Polis. Some evaluation metrics for terrain representation using triangulated irregular networks(tins). Technical report, School of Computer Science, Carnegie Mellon University, 1997.

[Paj98] Renato B. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization '98*, pages 19–26, 1998.

[PAL02] Renato Pajarola, Marc Antonijuan, and Roberto Lario. Quadtin: Quadtree based triangulated irregular networks. In *In Proceedings IEEE Visualization 2002*, page 395402. IEEE Computer Society Press, 2002.

[RC99] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *Symposium on Interactive 3D Graphics (I3DG)*, Atlanta, April 1999.

[RHS98] S. Roettger, W. Heidrich, and P. Slussallek. Real–time generation of continuous levels of detail for height fields, 1998.

[SC91] Francis Schmitt and Xin Chen. Fast segmentation of range images into planar regions. In *Conf. on Computer Vision and Pattern Recognition (CVPR '91)*, pages 710–711. IEEE Comput. Soc. Press, June 1991.

[SP03] Jose P. Surez and Angel Plaza. Refinement and hierarchical coarsening schemes for triangulated surfaces. *Journal of WSCG*, 11(1), 2003.

[Sug00] Vladislav I. Suglobov. Appearance-preserving terrain simplification. Tech report, Lomonosov Moscow State University, Moscow, Russia, 2000.

[XV96] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *In Proceedings of Visualization '96*, pages 327–334, 1996.

Appendix A

Comparing Crater Models

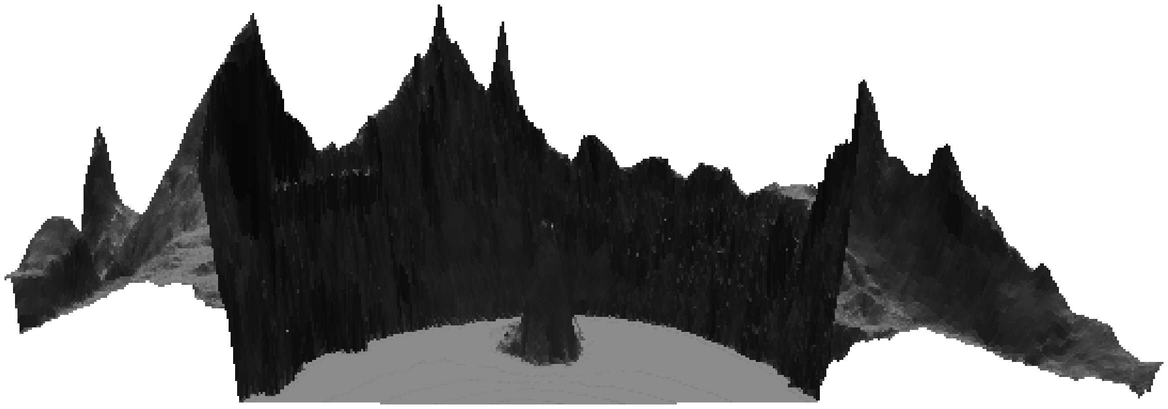
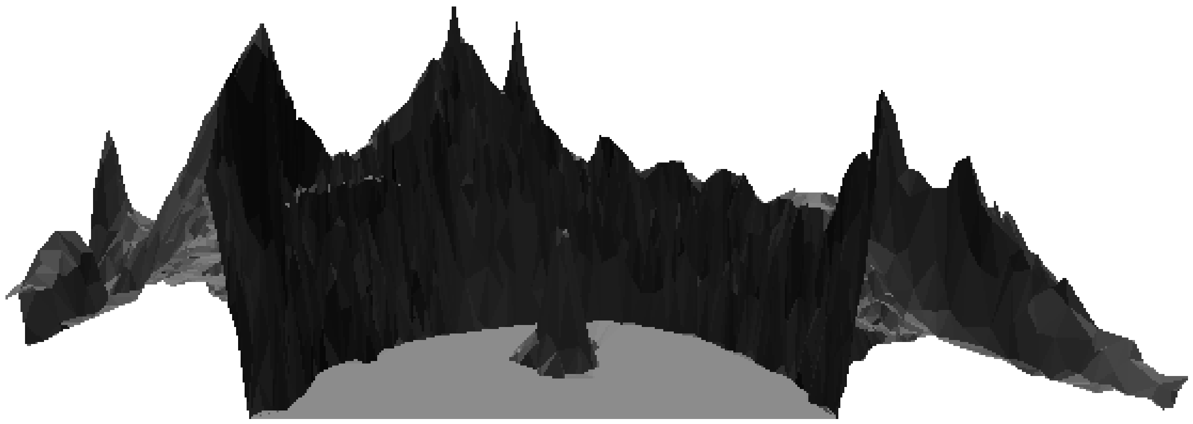
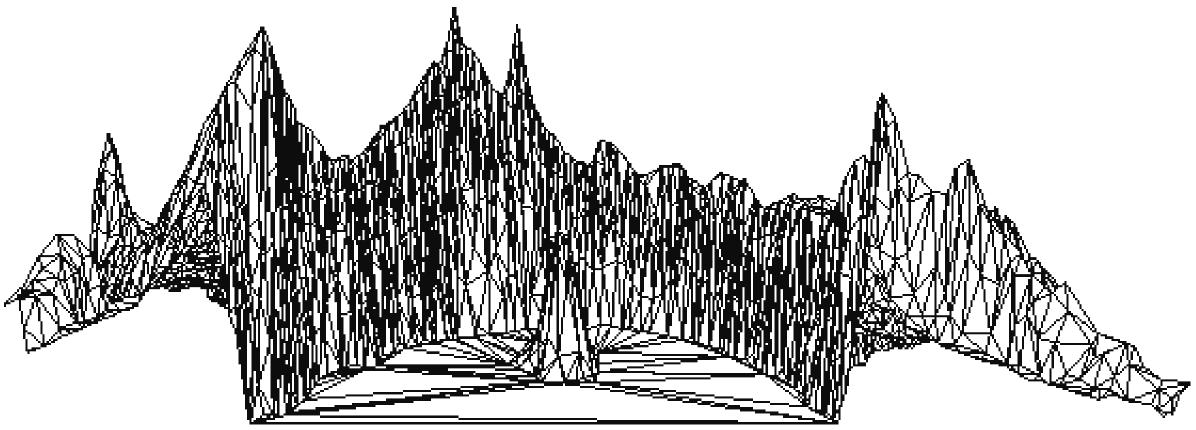


Figure A.1: Full Model of crater; 154224 vertices, 306860 faces

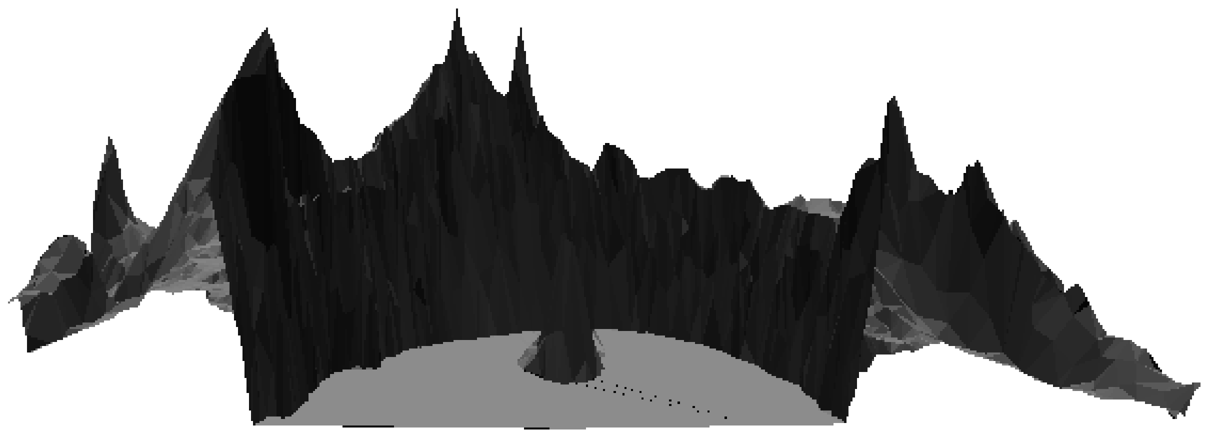


(a) Surface



(b) Mesh

Figure A.2: Hybrid Simplified crater; 5000 vertices, 9798 faces


(a) Surface

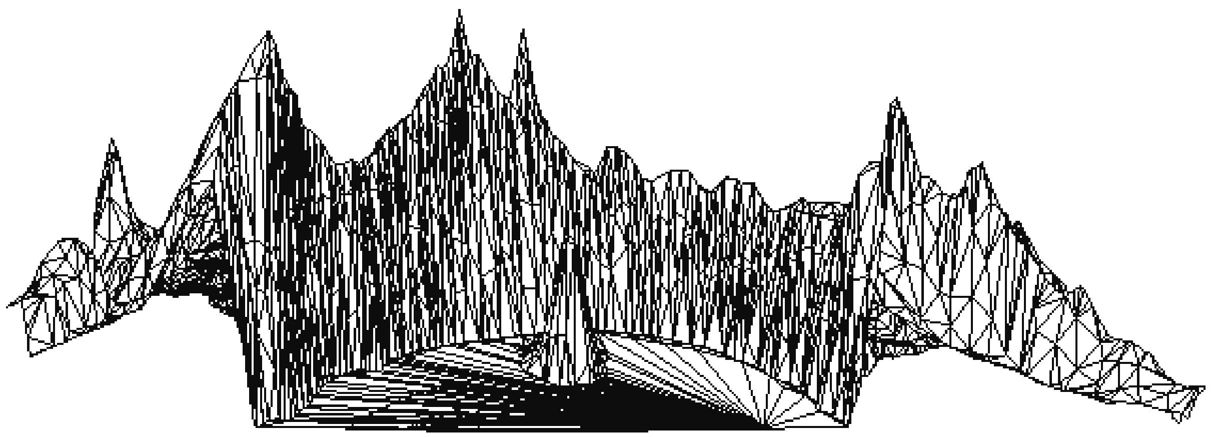
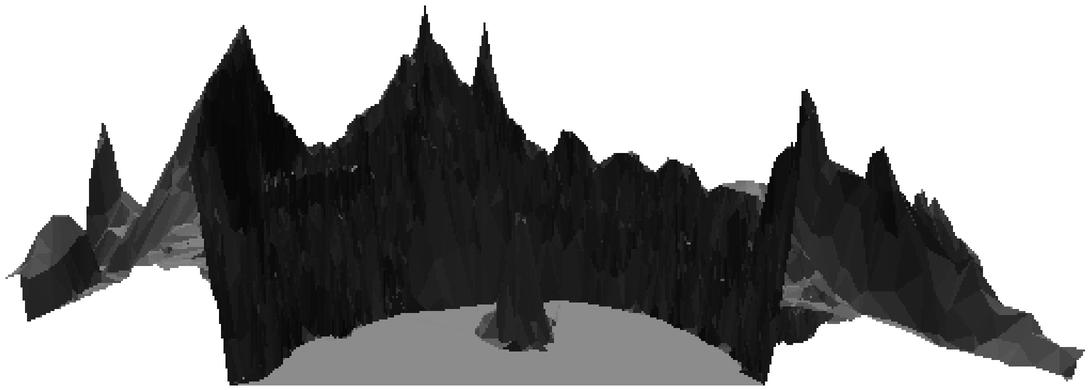

(b) Mesh

Figure A.3: QSlim Simplified crater; 5000 vertices, 9559 faces


(a) Surface

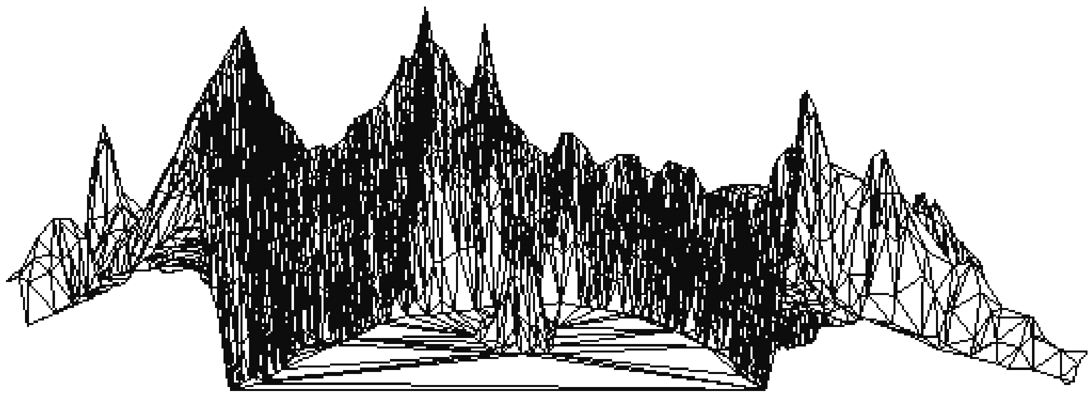
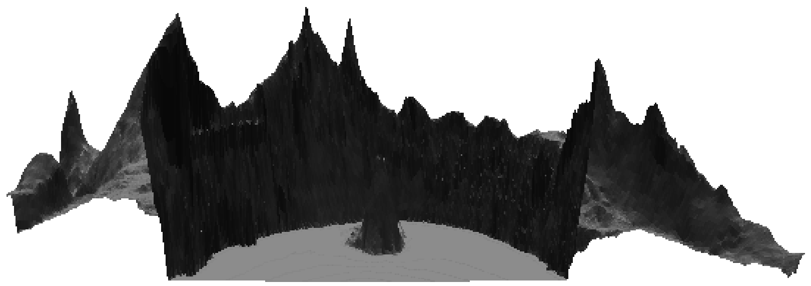
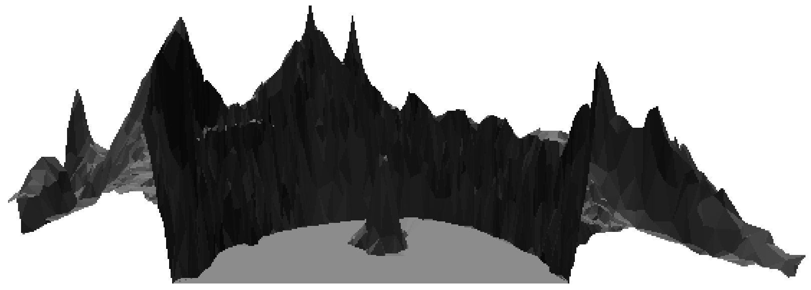
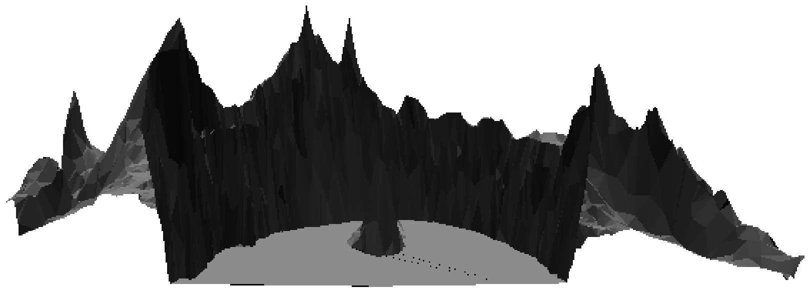

(b) Mesh

Figure A.4: greedy insertion Simplified crater; 5000 vertices, 9851 faces


(a) crater full


(b) hybrid


(c) QSlim

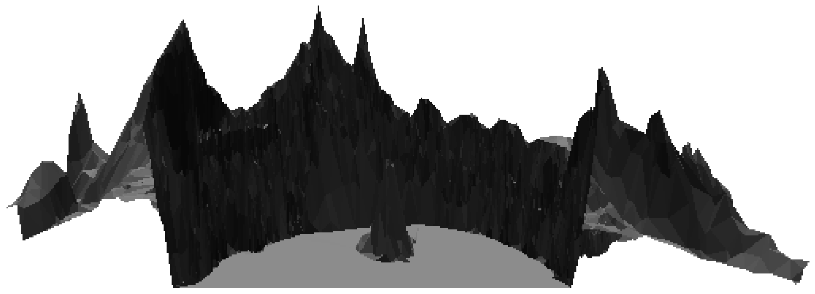
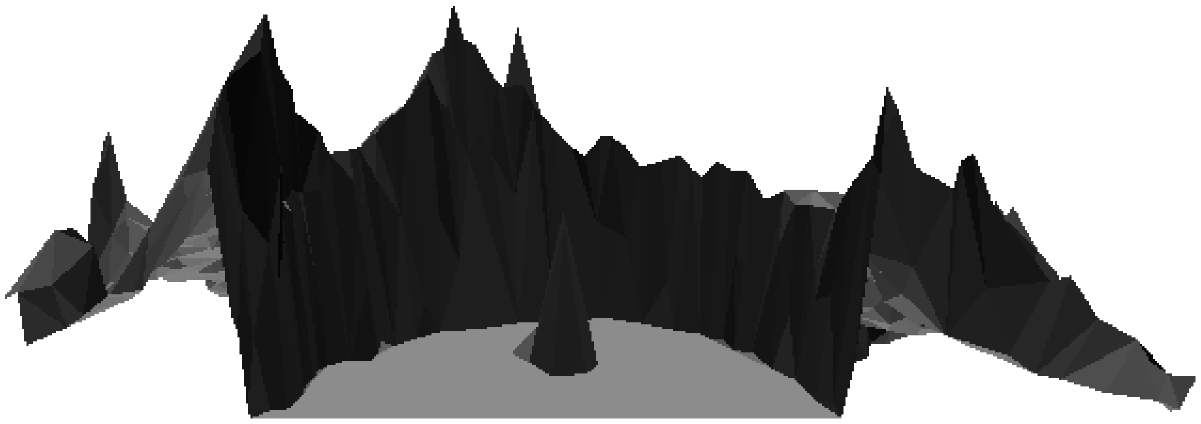
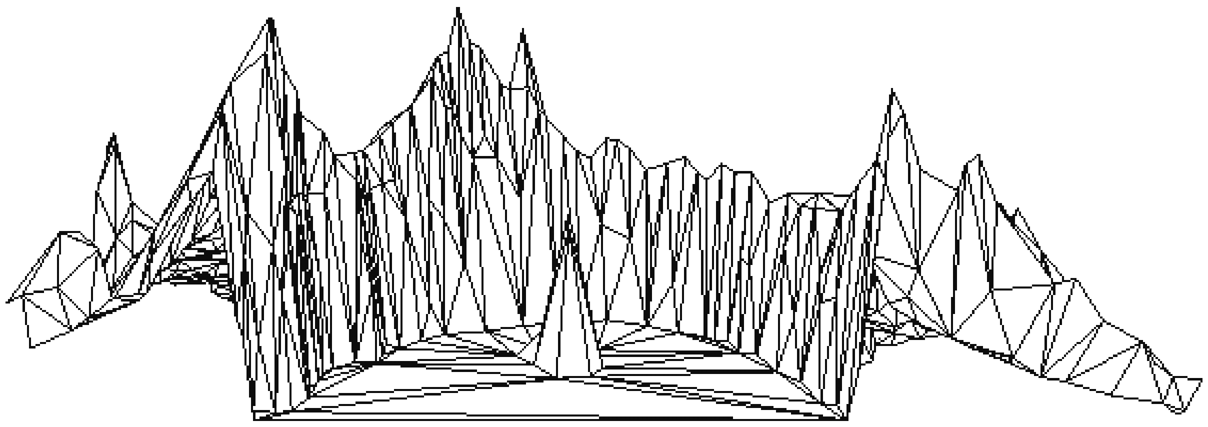

(d) GreedyInsertion

Figure A.5: crater: Comparing three methods for size 5000

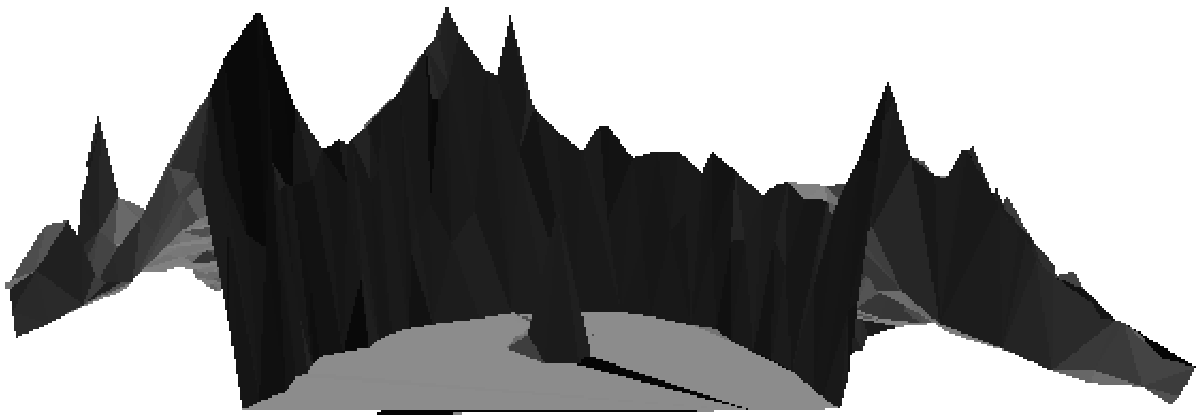


(a) Surface



(b) Mesh

Figure A.6: Hybrid Simplified cbaker; 1000 vertices, 1883 faces


(a) Surface

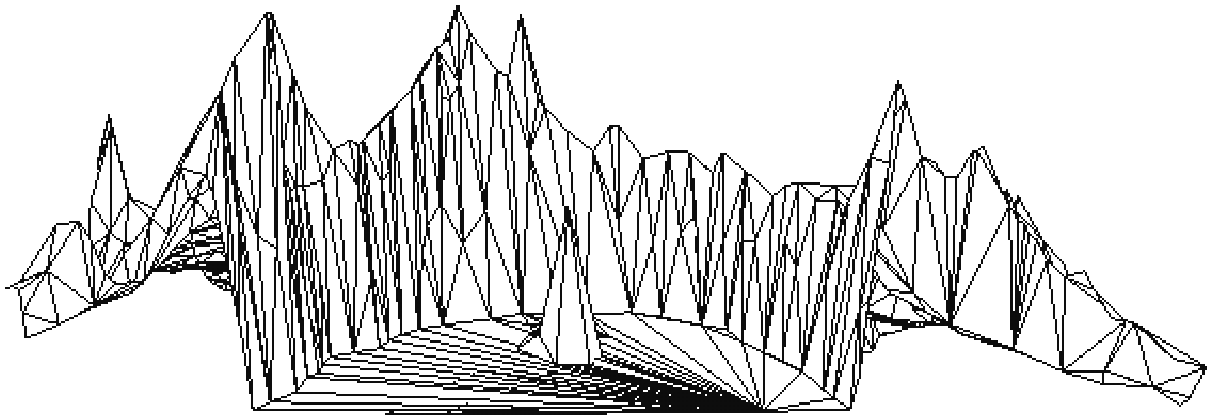
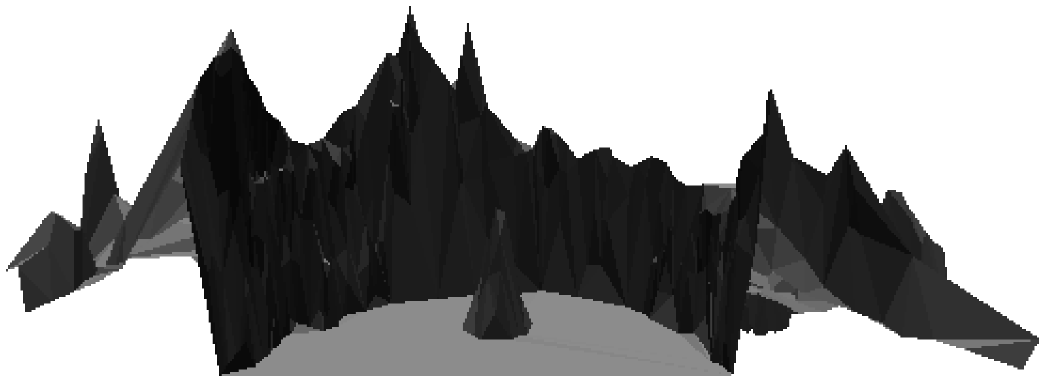
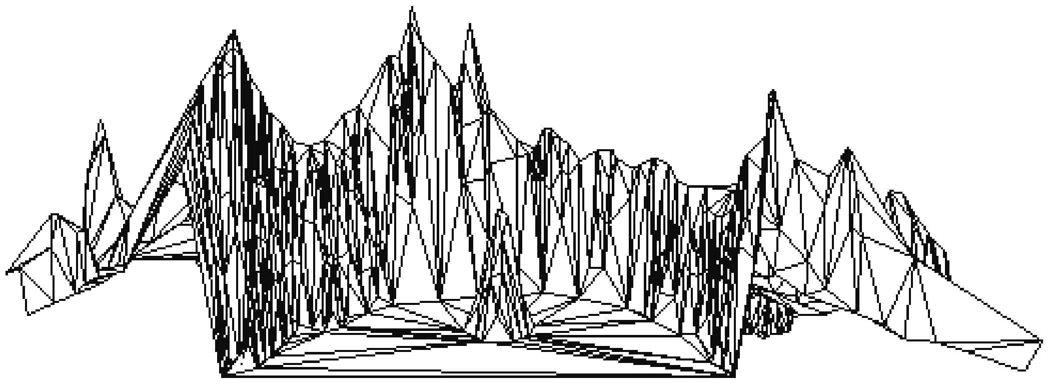

(b) Mesh

Figure A.7: QSlim Simplified crater; 1000 vertices, 1813 faces

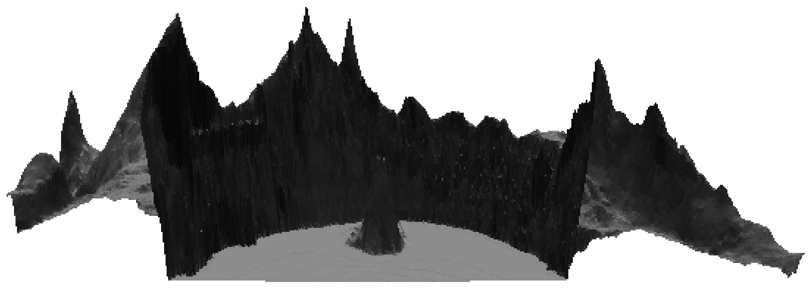


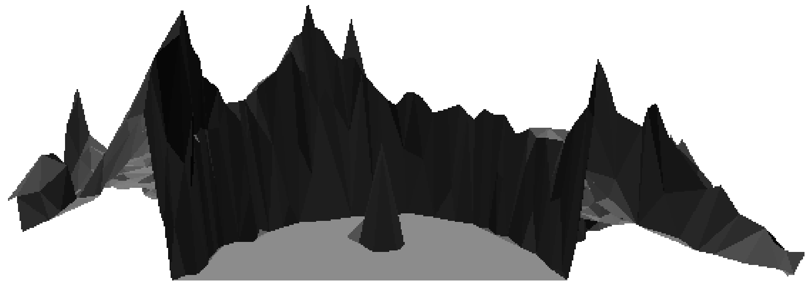
(a) Surface

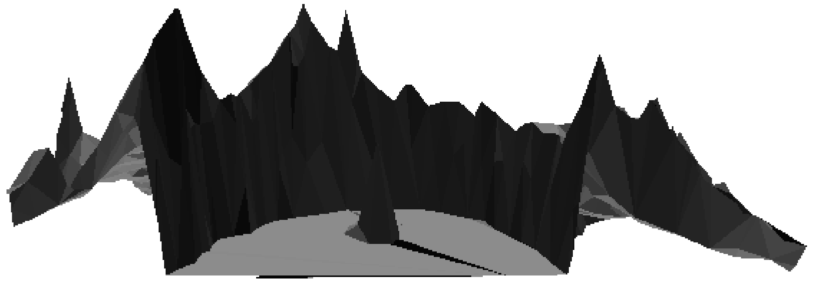


(b) Mesh

Figure A.8: greedy insertion Simplified crater; 1000 vertices, 1935 faces


(a) full


(b) hybrid


(c) QSlim

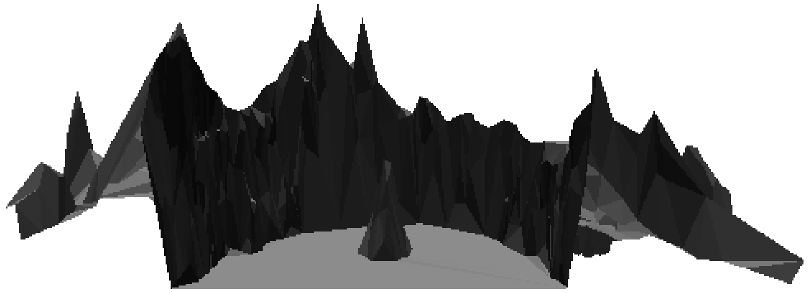
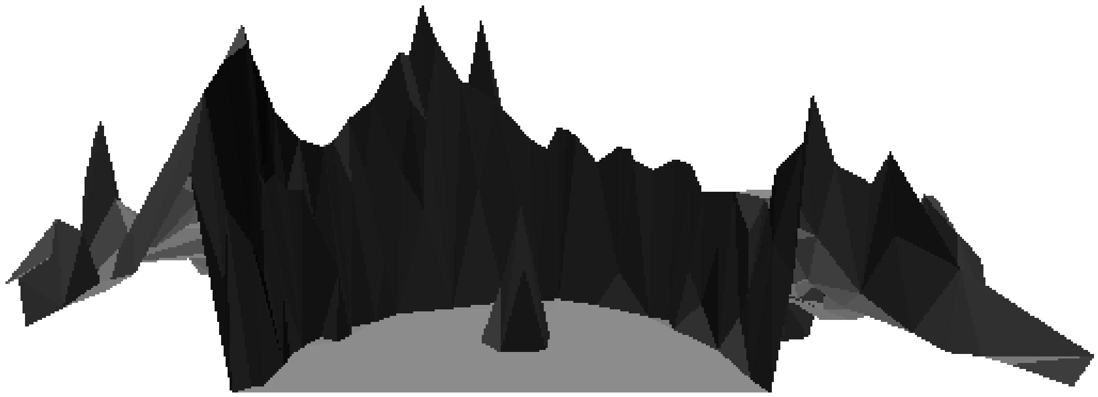
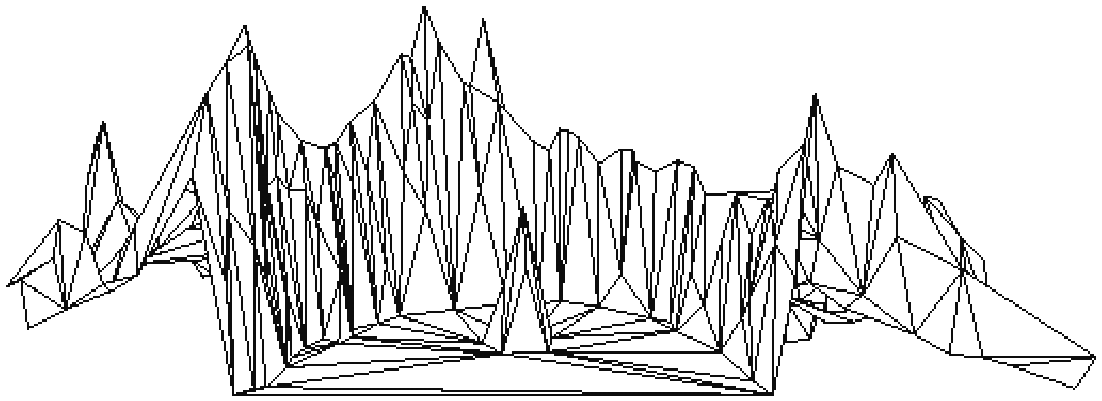

(d) GreedyInsertion

Figure A.9: crater: Comparing three methods for size 1000

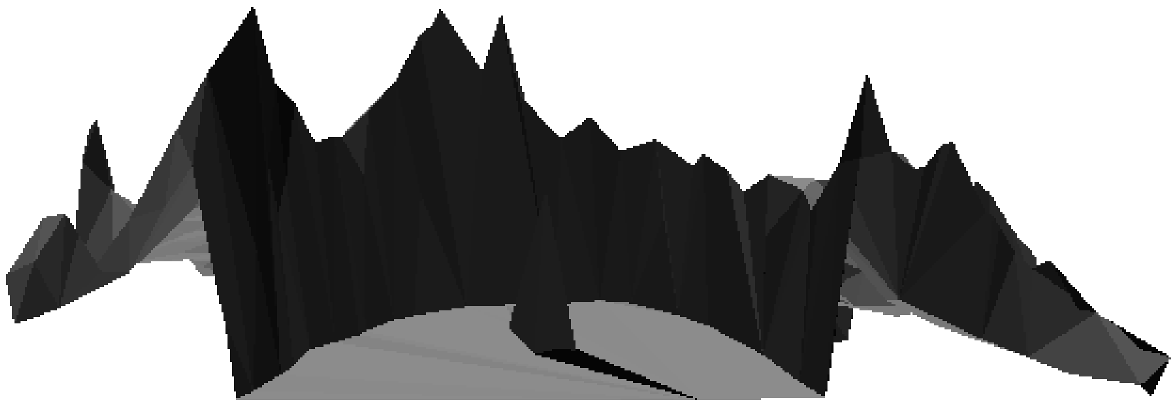


(a) Surface

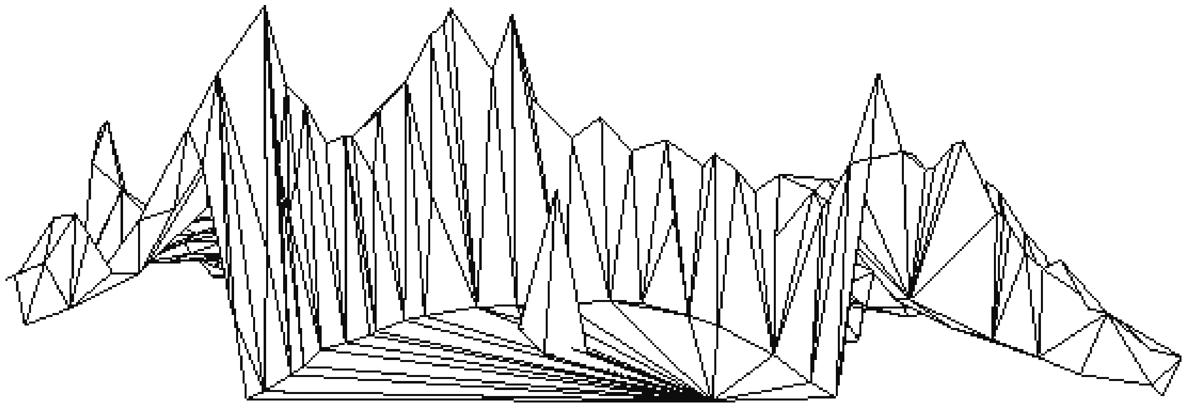


(b) Mesh

Figure A.10: Hybrid Simplified crater; 500 vertices, 936 faces

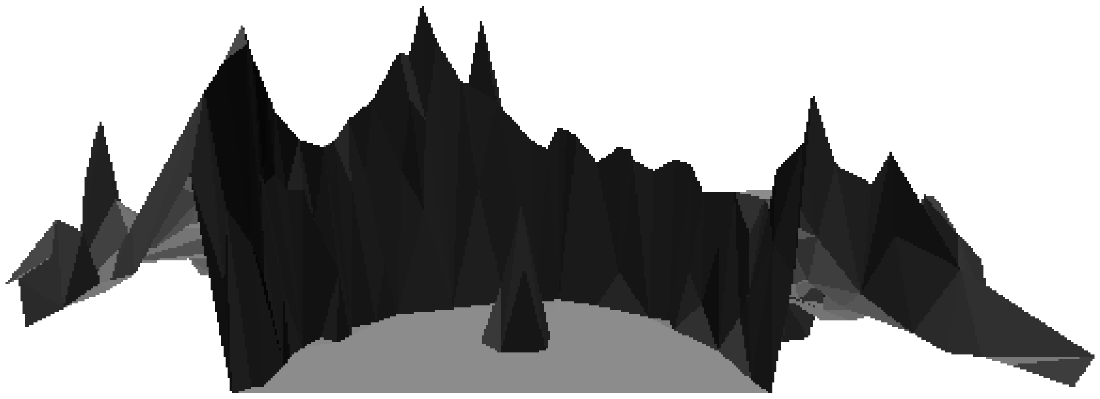


(a) Surface

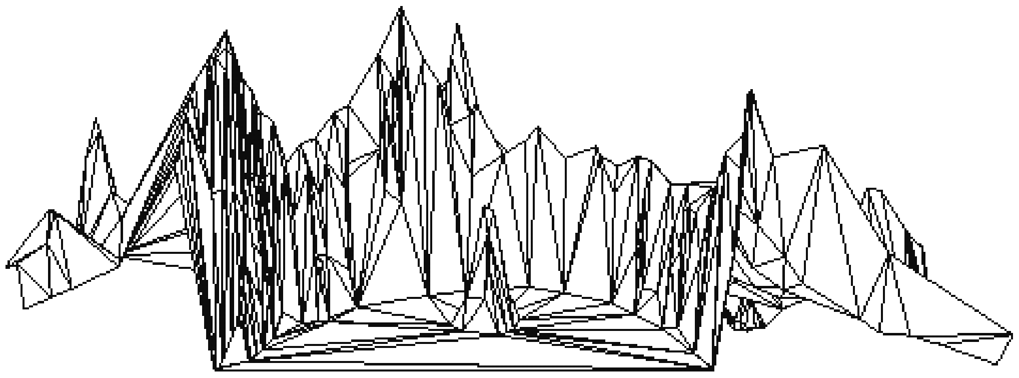


(b) Mesh

Figure A.11: QSlim Simplified crater; 500 vertices, 870 faces

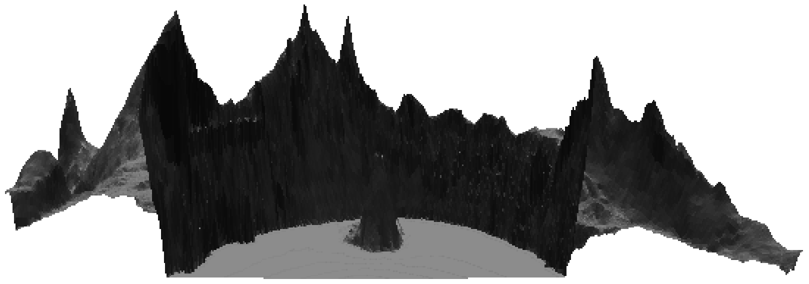


(a) Surface

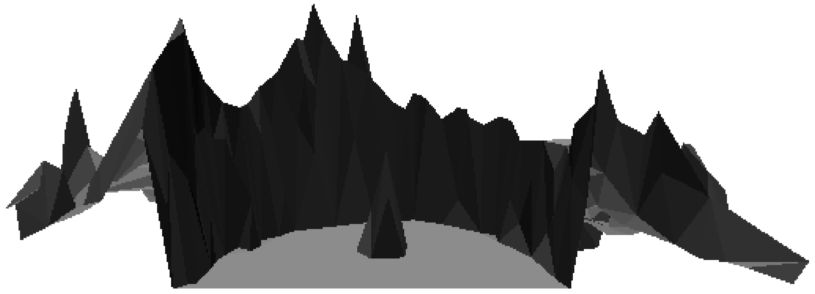


(b) Mesh

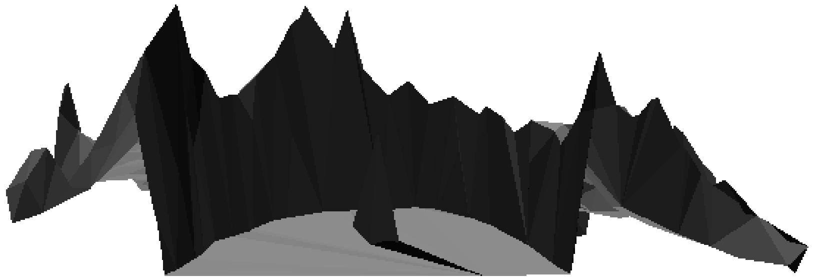
Figure A.12: greedy insertion Simplified crater; 500 vertices, 956 faces



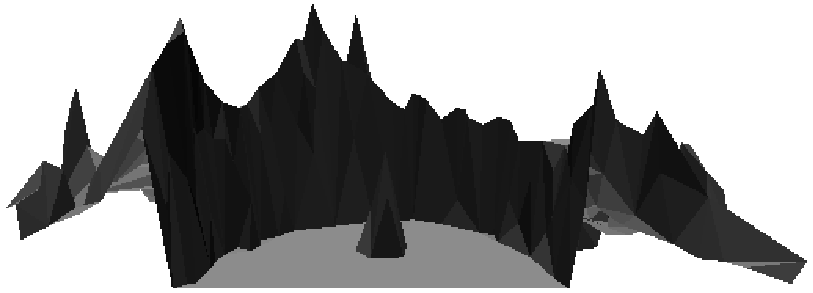
(a) crater full model



(b) hybrid 500 points



(c) QSlim 500 points



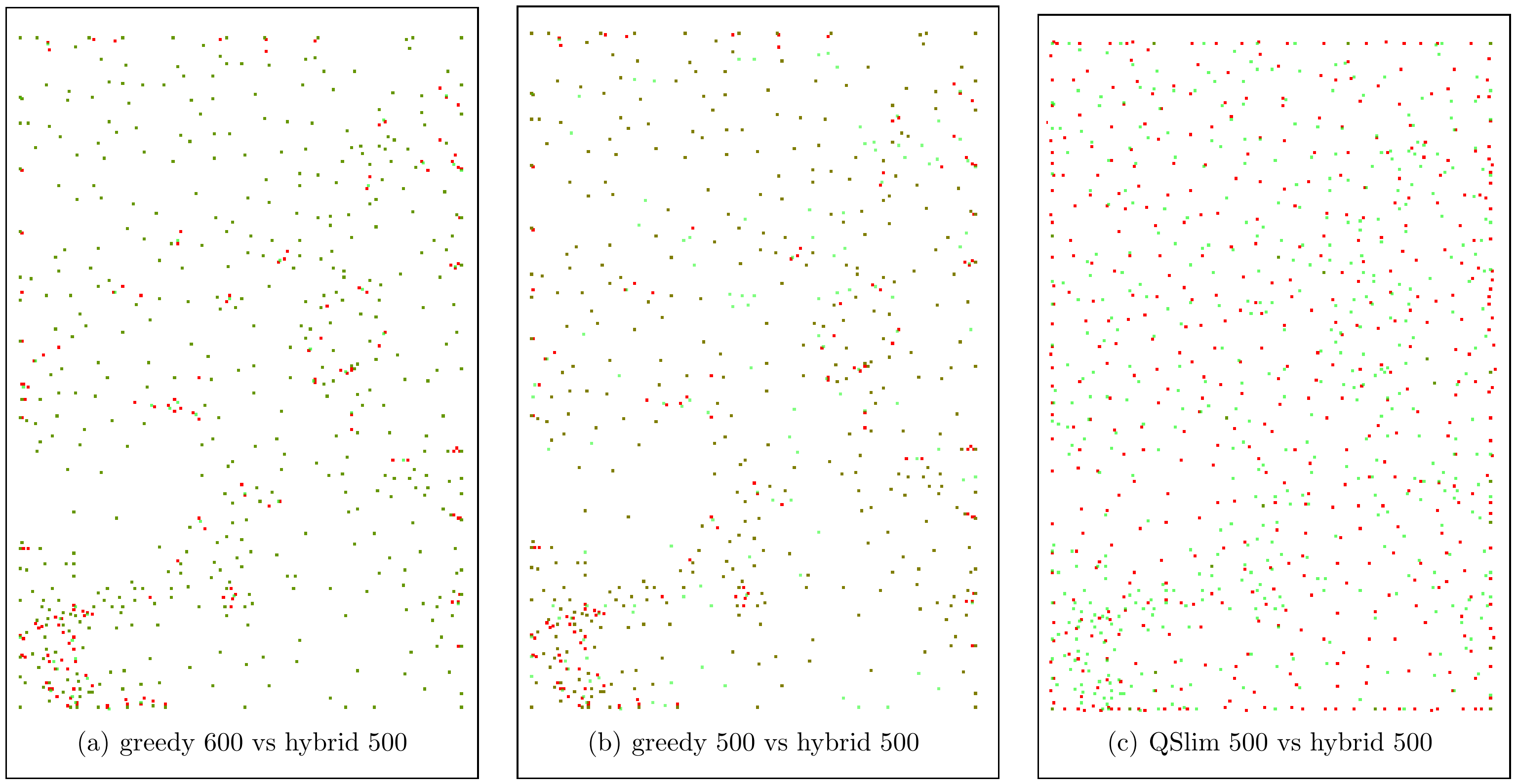
(d) Greedy Insertion

Figure A.13: crater: Comparing three methods for size 500

Appendix B

Shrinking Effects

The following pictures are converted from color images. The darkest points (originally represented by red color) represent the removed points of the first point set by the hybrid algorithm, the lightest points (originally represented by green color) represent new points added by the hybrid algorithm. The point set with grey scale in between the above two represent the unchanged points by the hybrid algorithm (or intersection of the two point sets).


(a) greedy 600 vs hybrid 500

(b) greedy 500 vs hybrid 500

(c) QSlim 500 vs hybrid 500

Figure B.1: Comparing the Shrinking Effect: Point Sets of the Three methods