FEATURE PRESERVING SIMPLIFICATION TECHNIQUES FOR TETRAHEDRAL MESHES

Chao Jin

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

September 2004
© Chao Jin, 2004

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-94742-4
Our file Notre référence
ISBN: 0-612-94742-4

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Feature Preserving Simplification Techniques for Tetrahedral Meshes

Chao Jin

Due to the wide use of increasingly larger tetrahedral meshes in volumetric visualization, simplification of tetrahedral meshes has become more and more popular in last two decades. In this thesis, we first introduce a basic tetrahedral mesh simplification algorithm based on cell collapse. Then, we present a new feature preserving simplification algorithm for tetrahedral meshes. The algorithm decimates the original dataset by iteratively removing tetrahedra without significantly altering boundary or interior field features. In a pre-processing step, we apply a level set method to find a segmentation of the volume dataset, and then label vertices on the region boundaries that potentially contribute to visually perceptible features in the rendered volume. The simplification algorithm preserves these labeled vertices as much as possible. Both incremental and greedy strategies are used to decimate tetrahedra that contain at most one labeled vertex. Field gradients, tetrahedral aspect ratio changes and variances of interior region values are further used so as to maintain features of the original dataset in regional interiors. A possible extension of combining edge collapse also presented to achieve higher decimation rates. We have implemented these algorithms and tested them using a number of standard volumetric datasets. The results have shown that the feature preserving simplification algorithm is able to preserve more features at the same decimation rates in comparison to other simplification algorithms.

Acknowledgments

I would like to gratefully acknowledge my supervisors Drs. Thomas Fevens and Sudhir P. Mudur for introducing me to this fascinating field and for providing me support during my work. Their enthusiastic guidance of my work and carefully reading of the text in this thesis are truly invaluable.

I would also like to thank Shuo Li for his help in my understanding and implementation of level set segmentation and his patience in answering my questions. I am also grateful to Dongwook Cho, Israat, Song Gao, who work in the same lab, for giving me a very happy time.

Also, I thank Ricardo Farias, Mississippi State University, Joseph S.B. Mitchell, Department of Applied Mathematics and Statistics, and Cláudio Silva, AT&T Lab Research Center, for ZSweep code and cleanSPX dataset; and www.volvis.org for Nucleon and CT-Head datasets; and NASA Ames Research Center for the Blunt Fin dataset.

Finally, I am forever indebted to my parents for their endless love, support, and encouragement whenever I need.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Volumetric visualization plays a very important role in scientific visualization, which is increasingly used in a wide number of areas such as biology, medicine, chemistry, computational fluid dynamics, etc.. Scientists and engineers employ a number of techniques such as contour detection and iso-surface generation, voxel-based tessellation, etc., to create volumetric models with different representations. Although many representations have been used in last several decades, tetrahedral mesh is still among the most popular one. This is due to the following factors:

1. The basic representational primitives, tetrahedra, are easy to deform and compute with. It is convenient to assign attributes and functions to the vertices, edges and tetrahedra. Computational steps such as interpolation, integration, and differentiation are fast and often can be done through such functions.

2. Tetrahedral mesh provides great flexibility such that other representations can

be converted into it relatively easily.

3. Finite element analysis is often conveniently performed on tetrahedral meshes.

4. The triangles that are generated by tetrahedra may be rendered using hardware acceleration [62, 42, 48, 59, 14].

However, since the aim of model generating techniques is not efficient rendering, the resulting models are usually very large in terms of the number of geometric primitives and complicated in terms of their spatial relationships.

In most cases, the original data contains vast amounts of information and with considerable noise and speckles, which result in a lot of storage space requirements and slow rendering speeds. Moreover, in some cases, it may not be necessary to render the image with its complete original information, such as in level of detail [37] rendering.

To achieve this purpose, many researchers have used simplification techniques. Since one of the main advantages of volumetric rendering is the ability to show both the interior and exterior features of the volumetric dataset, it is extremely important to ensure that the simplified dataset retains as many of the significant features as possible (both boundary and interior). This requires an effective feature detection/preservation technique combined within the simplification process.

Feature detection has been researched in the last several decades in many areas, such as computer vision, pattern recognition, and computer graphics. However, it is still a challenge problem to find good and efficient feature detection algorithms for volumetric datasets, whose geometrical and topological structures are usually very complex with redundancy and artificial noise [53, 50, 52, 17, 8, 7, 25, 6]. All existing

simplification algorithms define features based on gradient or iso-surface topological structure. The first approach may destroy some meaningful features. The second one keeps much unnecessary information as features [7, 6, 25]. In this research, we have explored the incorporation of new methods for feature detection and preservation during the simplification process.

A tetrahedron based volumetric dataset represents a region R in 3D space defined by a set of vertices and an associated tetrahedral mesh. Each vertex has one scalar (field) value as its attribute. The field values associated with any other point within R is determined by using an interpolation function. This interpolation function is based on the position and field values of the four vertices of the tetrahedron containing the point. Therefore, the volumetric dataset defines a distribution of attribute or field values in a 3D region. The objective of simplification is to build another volumetric dataset with less geometric and topological information than in the original one, and has as much as possible the same field value distribution as the original. However, an arbitrary change in the shape of tetrahedra or field values of the tetrahedral vertices may cause the attribute interpolation to result in a large error in field distribution [53, 17, 7] and also topological structure errors [50, 8].

1.2 Methodology

To achieve a rendering and storage efficient dataset, we need a tetrahedral mesh simplification tool which can reduce the complexity of the original dataset while preserve meaningful features. In this thesis, we are proposing to apply a cutting edge technique from image processing to obtain an improved tetrahedral mesh simplification algorithm. In addition to gradient based feature detection during the simplification

phase, we add a pre-precessing phase which performs volumetric segmentation to detect the interior features which should also be preserved during simplification. The two phases in our approach are:

1. ***Volumetric Segmentation***: Using an image processing method, we divide the 3D space defined by the original dataset into several sub-regions. Inside each sub-region, its field values are assumed to be following the Gaussian distribution. Our premise is that the boundaries between sub-regions have more potential to be features; therefore, we will try to preserve these boundaries during the simplification phase.

2. ***Simplification***: During the simplification phase, we simplify each sub-region individually, while preserving the boundary between neighboring sub-regions. The algorithm performs mesh decimation by a sequence of cell collapses. The order of collapse is determined by a predicted error measure, which is defined as a combination of local error and region based error.

Compared to existing simplification algorithms, our approach has some significant advantages:

1. It is very robust to noise which is a very major challenge to existing simplification algorithms.

2. It can preserve complicated topology of interior regions.

3. It can detect some features that classical gradient methods would normally miss.

This thesis is organized as follows: In chapter 2, we give a brief review of 3D surface mesh simplification technique and a survey of most popular and traditional tetrahedral mesh simplification methods. In chapter 3, we introduce a basic tetrahedral mesh

simplification algorithm with a sequence of edge collapse. A level set segmentation based feature detection method is proposed in chapter 4. With this method, a feature preserving simplification algorithm and its extension also described in this chapter. Chapter 5 is our experimental results, and the final chapter is our conclusion.

Chapter 2

Background and Related Work

2.1 History of Mesh Decimation

Most volumetric decimation algorithms come from 3D surface mesh decimation algorithms. Therefore, we first introduce a brief review of 3D surface mesh decimation algorithms. And then we present the current state of the art in tetrahedral mesh simplification.

2.1.1 3D Mesh Decimation

Surface mesh models were introduced to the scientific visualization field much earlier than the volumetric models. The idea of using the surface to present the object comes from the real world. Most objects are opaque. Our eyes can tell the object easily by detecting the object's surfaces and edges. Scientists explored a huge number of algorithms to obtain the surface models [10, 43, 51, 56, 1, 41]. Typically, these models are represented by vertices, edges, and polygons.

The complexity of these models is measured by the number of polygons [38] and

this number is currently growing faster than the ability of graphics hardware to render interactively. Therefore, many scientists and engineers have designed simplification techniques to help remove the small, distant, or otherwise unimportant portions of the model, seeking to reduce the rendering cost without a significant loss in the scene's visual content [22]. Also, they explored the idea of multiresolution models [44, 26, 49], a computer graphics technique which is very old but still current, for visualization of very large datasets in real time speeds. As early as 1976, James Clark described the benefits of representing objects within a scene at several resolutions [9],and flight simulators have long used hand-crafted multi-resolution airplane models to guarantee a constant frame rate. Recently, a flurry of research has targeted generation of such models automatically. The main methods for simplification, as classified in [22], includes decimation methods [21, 16], refinement methods [15, 27, 12, 13], and optimal methods [39, 11, 3]. In our thesis, we are mainly concerned with decimation methods.

Decimation methods form a significant class of surface simplification algorithms. This class starts with a polygonized representation and successively simplifies it until the desired level of approximation is achieved. Most decimation algorithms fall into one of the following categories:

1. ***vertex decimation algorithms***: This class of algorithms performs the decimation operation by first deleting a vertex, then all edges, and triangles connected to the vertex; and then retriangulating the hole generated by the operation [32].

2. ***edge decimation algorithms***: This class of algorithms performs the decimation operation by deleting one edge, then the two triangles adjacent to that edge, and then, merging the two vertices of the deleted edge [16].

3. ***triangle decimation algorithms***: This class of algorithms performs the decimation operation by deleting one triangle and all vertices and edges immediately adjacent to the deleted triangle; and then, retriangulating the hole generated by this deletion operation.

4. ***patch decimation algorithms***: This class of algorithms performs the decimation operation by deleting several adjacent triangles and then retriangulating the polygonal hole. [28].

2.1.2 Volume Mesh Decimation

In this section, we review some of the existing algorithms described in the literature for simplification of irregular mesh volume datasets, which is related to our research. Based on different atomic decimation operations, we can classify the simplification algorithms for volumetric datasets into three types: vertex decimation [17, 25], edge collapse [6, 8, 50, 52] or tetrahedral cell collapse [7, 53].

In 1998, Trotts et al. [53] presented a method for the construction of multiple levels of detail of tetrahedral meshes approximating a trivariate function at the different levels by extending a polygonal geometry reduction technique. Their algorithm starts with a full-detail, high-resolution tetrahedralization of a three-dimensional region, and constructs increasingly coarser representational levels by a sequence of three edge collapse operations. The order of tetrahedral collapse is based on error prediction defined on the triangulation based linear spline function, where the functional values associated with the vertices are the spline coefficients. They use an incremental approach with an update of the error bound for each individual tetrahedral cell after each decimation step. They also discuss the preservation of the boundary. In 1999,

Trotts et al. [52] extended this work by binding the degeneration error to the gradient of simplified field values determined from the original field values. Also, they changed from tetrahedron collapse to edge collapse as the primary decimation operation. They state that their technique achieves 83.9% decimation for the Blunt Fin dataset with 419,297 cells in almost 2 days.

Staadt and Gross [50] extended the work of Hoppe [26] on progressive triangular meshes to tetrahedral meshes to generate an incrementally refined progressive tetrahedralization based on edge collapse. The decimation sequence is based on a predicted error that includes gradient, volume preservation and uses the edge collapse and vertex split primitive operations. They also do static sharp boundary checks and dynamic edge intersection checks to avoid inconsistent topological changes. The flipping problem, where a tetrahedral vertex shifts to the other side of its facing triangle thus introducing a negative volume tetrahedron, is prevented by checking the sign of the parallelepipedial product of three edges of the tetrahedron. From the results of their algorithm, it is evident that the approach does not always preserve boundary and/or interior features. Also, because their simplification algorithm includes a very expensive dynamic mesh-consistency test, its computation times are very high, especially for very large datasets; nearly 5 hours were needed to deal with a dataset of 576,576 cells.

In 2000, Cignoni et al. [8] give a framework for incremental 3D mesh simplification also based on edge collapse. In their paper, they propose an approach for the integrated evaluation of the error introduced by both the modification of the domain and the approximation of the field of the original volume dataset. They also include a computation time comparison of various techniques that can be used to compute

the error after simplification.

In 2002, Chopra and Meyer [7] introduced a fast tetrahedral decimation algorithm called TetFusion. During simplification, the algorithm avoids such problems as negative volume tetrahedra, volume boundary self-intersections, and tetrahedral boundary self-intersections in regions where the boundary is concave. They are able to achieve reduction of up to 98% for meshes consisting of 827,904 tetrahedra in less than two minutes. However, this algorithm does not incorporate any checks to preserve the original distribution of field values or interior features, except in forbidding any change in volume boundary tetrahedra. Hence for smaller datasets, this algorithm cannot achieve high decimation rates, as larger percentage of the tetrahedra are on the boundary and are untouchable during simplification.

Recently, Hong and Kaufman [25] described an algorithm that performs simplification using vertex decimation. In their algorithm, they classify all interior vertices as either topologically critical points or regular points. The topological structure is preserved by not touching these critical points and also by preventing regular vertices from becoming critical during simplification. To order the vertices during simplification, they use a gradient magnitude based error metric. They also pre-compute a solid angle which is used to detect boundary features that are to be preserved during the simplification.

Chiang and Lu [6] use Morse theory to preserve the topological structure among all isosurfaces during the progressive simplification at multiple levels of a tetrahedral volume dataset. In the first stage they determine a subdivision to separate the dataset into topology-equivalent regions with ranges of isovalues by identifying critical points where the topology of the field values changes. They construct a contour tree that

maps out neighbouring topologically equivalent regions with respect to these critical points. In the second stage they simplify each topologically-equivalent region separately using edge collapse. During simplification, any change to the isosurface-based topological structure is forbidden. The free decimation can only be done though on a pure-cell whose vertices are not critical points and that are in the same topologically equivalent region. This results in a strict topology-preserving simplification that is highly performance limited, as the algorithm will retain even tiny insignificant features, say, caused by noise.

Base on those researchers' work, in next chapter, we propose a basic tetrahedral mesh simplification algorithm with cell collapse. The sequence of collapse operation is ordered by a predict error defined by a combination of gradient changes and shape changes.

Chapter 3

Basic Volumetric Dataset Decimation Algorithm

In this chapter, we propose a basic simplification algorithm based on cell collapse, and discuss such issues as new vertex generation, error metrics, flipping problem and boundary preservation.

3.1 Goal and Difficulty of Decimation

As defined in [20], mesh decimation describes a class of algorithms which transfer a given mesh into another one with fewer primitives, such as vertices, edges, faces or cells. The decimation procedure is usually controlled by user defined quality criteria which prefer meshes that preserve specific properties of original data as much as possible. Typically, these criteria are related to the appearance of the final image, such as color difference, shape of object and topological structure [62, 58].

However, the appearance information maybe very unclear before we see the rendered result. In other words, it is very hard to define which information is more important than others and must remain in the simplified meshes before we obtain the final image. In medical imaging, for instance, topology may provide critical information; depending on the application, preserving a hole in the heart wall may be much more important than preserving the exact shape of the surface. However, in most rendering applications, we can safely simplify the topology of models. Actually, it is often desirable to do so. Consider a model of a sponge. When we view very closely, the intricate structure of holes in the sponge is a visually important feature. But when viewed from a distance, these holes are imperceptible. The entire sponge can be adequately approximated by a simple block. In this chapter, we have to opted to compromise in some respect to a more general algorithm which mainly focuses on retaining geometry and field value distribution information.

Another big challenge is that for volume data simplification, like most problems where optimal algorithms are infeasible, a tradeoff exists between quality and efficiency. A fast algorithm capable of producing high-quality approximations has been the primary goal of simplification work. However, there is always a gap between the very fastest algorithms, which can produce poor approximations, and those algorithms which produce very high quality approximations, but which can be rather slow. Therefore, we also have opted to compromise quality in some respects in order to create a more efficient algorithm.

3.2 Definitions

Before we begin the discussion on simplification, we give some definitions which we use later.

- **Tetrahedral Mesh** Let $\mathbb{D} : (\Sigma, V, \Gamma, S)$ be our dataset where Σ is a closed sub-region in 3D space R, V is a set of sample points within the region Σ, $\Gamma = \{\tau_1, \tau_2, ..., \tau_m\}$ is a tetrahedralization of m cells defined on V, and S is the scalar attribution within region Σ. An assumption widely used in volume rendering and simplification is that the scalar value of an attributed point $p \notin V$ within a tetrahedron can be obtained by linear interpolation of the four sample vertices of the tetrahedron.

- **Neighbor set of a tetrahedral cell** For $\forall \tau \in \Gamma$, $A(\tau)$ defines a subset of Γ , which shares one and only one vertex with τ; $D(\tau)$ defines a subset of Γ, which shares at least two vertices with τ. We denote the collection of $A(\tau)$ and $D(\tau)$ as $Neighbor(\tau)$. More formally:

$$A(\tau) = \{\tau_i \in \Gamma | \tau_i \text{ share exact one vertex with } \tau\} \tag{1}$$

$$D(\tau) = \{\tau_i \in \Gamma | \tau_i \text{ share at least two vertices with } \tau\} \tag{2}$$

$$Neighbor(\tau) = A(\tau) \cup D(\tau) \tag{3}$$

- **Boundary Cell** For all $\tau \in \Gamma$, if any of the vertices of τ are on the boundary, which means the cell contains at least 1 vertex and at most 3 vertices on the boundary, we denote τ as a boundary cell.

- **Boundary Neighbor Cell** For all $\tau \in \Gamma$ that are not boundary cells, if any $\tau_j \in Neighbor(\tau)$ is a *boundary cell*, we denote τ as a boundary neighbor cell.

3.3 Decimation Method Based on Cell Collapse

As we discussed before, given an irregular dataset $\mathbb{D}$, the term simplification refers to the problem of building an approximate representation $\mathbb{D}'$ with less information than $\mathbb{D}$, built by choosing a set of vertices V' smaller than V, and a new triangulation Γ of V' that defines a field value distribution S' which keeps almost the same scalar attribution as of S. We have chosen cell collapse as our decimation operation. As discussed in the previous section, there are mainly three kinds of decimation methods used in tetrahedral mesh simplification: vertex decimation, edge collapse, and cell collapse. However, among these three methods, we find that cell collapse has a higher decimation rate per step in general. The collapse of one cell τ, excluding the boundary cell results in the removal of exactly three vertices, six edges, and at least 10 additional tetrahedral cells, which share a edge or a face with τ. Based on a test run on the data set SPX with 12,936 cells, during each cell collapse, we have found that we decimate an average of 20 cells. Moreover, it is not required to maintain the edge information during the decimation process. The simpler data structure of an array of vertices and an array for tetrahedral cells is sufficient. All these result in less computation time to implement decimation. In other words, cell collapse may lead us to more efficient decimation algorithms.

In each step, we choose one τ_i as the collapsing candidate, and then find the tetrahedra sets $A(\tau_i)$ and $D(\tau_i)$. After the collapse operation, the candidate cell become a vertex; the tetrahedra inside $A(\tau_i)$ moves the vertex shared with the collapsed

candidate to the newly generated one; the tetrahedra inside $D(\tau_i)$ which share three vertices with the candidates become edges, and the tetrahedra inside $D(\tau_i)$ which share two vertices with the candidates become triangle faces. The hole space generated by the collapse operation is distributed amongst the candidate's local neighbors. (See Figure 3.1)

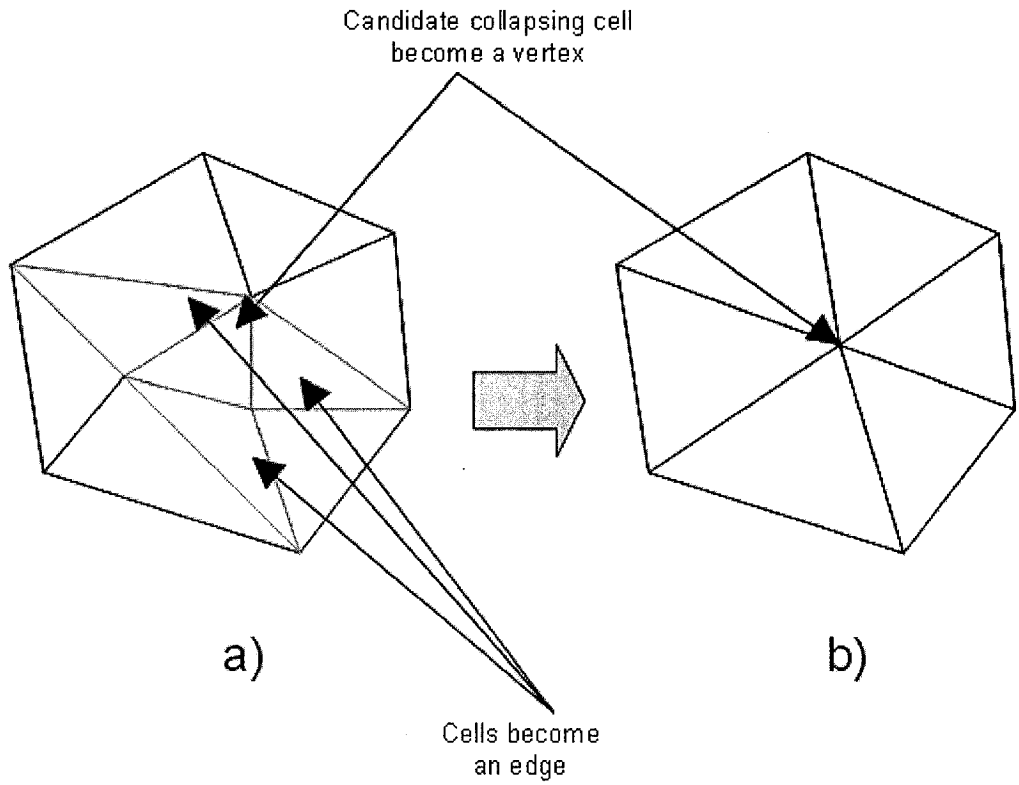

Figure 3.1: Demonstration of collapsing operation in 2D *a*) The candidate collapsing cell τ and it's $Neigbhor(\tau)$ collection. *b*) Mesh after the collapsing operation. The three vertices of the candidate cell become a new vertex. All cells in $A(\tau)$ set move one vertex to a new position. All cells in $D(\tau)$ become an edge.

3.3.1 New Vertex Placement

Because we use cell collapse, it is unavoidable to generate new vertices in original tetrahedral mesh. Obviously these new vertices must be placed inside the volume of

collapsed cells. The easiest implementation of the collapse would be to replace three vertices with the fourth one. However, it is not a good choice, most of the time, because we cannot be sure that all vertices of a cell have similar attribute values and have the same classified feature. Also, the hole resulting from a cell collapse operation is distributed unevenly to its local neighbors in the $A(\tau)$ set.

Another simple way to replace the new vertex is to use the bary center of the collapsed cell, and determine its attribute value by a linear interpolation of attribute values of the original four vertices. The new vertex carries a blended feature of original vertices. Also, it is a symmetric choice because the hole resulting from the cell collapse operation is distributed symmetrically amongst the local neighborhood.

For boundary cells, it is more reasonable to have the new vertex appear on the original boundary complex. Therefore,

1. If the boundary cell contains only one boundary vertex, we choose the boundary vertex as the new vertex;

2. If the boundary cell contains two boundary vertices, we choose the center of edge defined by the two boundary vertices as the new vertex;

3. If the boundary cell contains three boundary vertices, we choose the bary center of the face composed by the three boundary vertices as the new vertex.

3.3.2 Error Metrics

Considering that the goal of simplification is that the final mesh has an attribute value distribution that is as close as possible to the original one, not all tetrahedra are suitable to become collapsing candidates. As discussed in [53, 50, 52, 17, 25, 6],

error prediction is very important for a good simplification algorithm. We always want to choose a tetrahedral cell which causes the least modification to the attribute value distribution.

We define the change in attribute value distribution S and S' , before and after cell collapse, on a 3D region R as our error, denoted by $\Delta\varepsilon$

$$\Delta\varepsilon = \sum_{p\in\Sigma} |S_p - S'_p| \tag{4}$$

However, it is computationally very expensive to calculate the $\Delta\varepsilon$ after each collapse operation. Therefore, instead of computing error after each cell collapse, we want to use some measures for ordering the cells based on the possibility of large $\Delta\varepsilon$ [26, 50, 53, 52, 7, 25, 6].

The first criterion we can use and should use in error metrics is gradients. A large gradient change means there is a possible feature [31, 34]. Therefore, the cell which contains big gradient changes may carry more information than the one that contains small gradient changes. As candidates for decimation, the one with a smaller gradient is better than the one with a larger gradient. We define the gradient error as follows:

$$\varepsilon_{gradient} = \omega_{gradient}\frac{1}{4}\sum_{i=0}^{3} |S_i - S'_n| \tag{5}$$

where S_i are the attribute values of original vertices of the collapsed cell.

S'_n is the attribute value of new vertex which is created after collapsing the cell.

The second criterion we consider is the geometric shape of the tetrahedral cell. After the collapse of one cell, attribute value distribution inside the area R' which is defined by $Neighbor(\tau)$ will change. If R' is a big area, it may cause more modification

in distribution. If the candidate cell has big volume, the collapse operation may cause a large changes in S. The collapse of a sharp cell may cause more $\Delta\varepsilon$ than the collapse of a round cell [36, 33]. Therefore, we define:

$$
\begin{aligned}
\varepsilon_{shape} &= \omega_{neighbor}\varepsilon_{neighbor} + \omega_{sharp}\varepsilon_{sharp} + \omega_{volume}\varepsilon_{volume} \\
\varepsilon_{neighbor} &= \frac{\sum_{\tau_i \in Neighbor(\tau)} Volume(\tau_i)}{Volume(\tau)} \\
\varepsilon_{sharp} &= \frac{1}{4}\sum_{i=0}^{3} |v_i - v_n| \\
& \quad v_i \text{ is the original sample vertex,} \\
& \quad v_n \text{ is the new genertated one} \\
\varepsilon_{volume} &= \frac{1}{3!}|a \cdot (b \times c)| \\
& \quad a,\ b,\ c \text{ represent the three edges sharing one common vertex}
\end{aligned}
\tag{6}
$$

We use the summed fraction of volume for all cells inside neighbor set of candidate with the one of candidate, and stretch length of vertices of tetrahedron to check for a sharp cell. It is also possible to use the solid angle to check the latter.

Combining the two types of error, we define a simple error metric as follows:

$$\Delta\varepsilon = \varepsilon_{gradient} + \varepsilon_{shape} \tag{7}$$

3.3.3 Boundary Decimation

In TetFusion [7], in order to preserve the geometry boundary, the collapse operation is forbidden on the following types cells:i) boundary cells, which has at least one

vertex on the boundary, and ii) the neighbor boundary cells, which have at least one tetrahedral cell in its $Neighbor(\tau)$ that is a boundary cell. This requirement preserves the geometry boundary very well. However, it also prevents reaching higher decimation rates, especially for volume datasets which have a complicated structure.

Let B be the original boundary defined by the tetrahedral mesh, and B' be the simplified boundary after one or more decimation operations. We wish to choose a suitable criterion measuring the changes between B and B' as $|B - B'|$. If this change is within a reasonable tolerance, we can allow the cell on the boundary to be a candidate for the collapsing operation.

Hausdorff Distance

Usually when considering distances between two polygons, we mean the shortest one: for instance, if we have two polygon P_1 and P_2, the distance from a point $p_i \in P_1$ to the polygon P_2, we generally assume that it is the distance from p_i to the nearest point $p_j \in P_2$. The same logic applies for distance between two polygons: if two polygons P_1 and P_2 are at some distance from each other, we commonly understand that distance as the shortest one between any point of P_1 and any point of P_2. It is denoted as:

$$Dis(P_1, P_2) = \min_{p_i \in P_1} \{ \min_{p_j \in P_2} d(p_i, p_j) \} \tag{8}$$

We usually refer to equation (8) as the *minimal* distance between two polygons.

However, the *minimal* distance can become quite unsatisfactory for some applications because the *minimal* distance is totally independent of the shape of each polygon. For example in figure(3.2) , we can say that the two polygons are very close to each other because the minimal distance is very small. In fact, sometimes when

we say two polygons are very close, it really means that no point of one polygon is far from the other one. In this sense, the two polygons in Figure(3.2) are not so close, as their farthest points, shown in gray, could actually be very far away from each other. Another disadvantage of *minimal* distance is that it carries very low information content. From Figure(3.3), it is quite obvious that the minimal distance value of these two polygons is exactly the same as that in figure(3.2), while *something* did change substantially with the configuration of these objects.

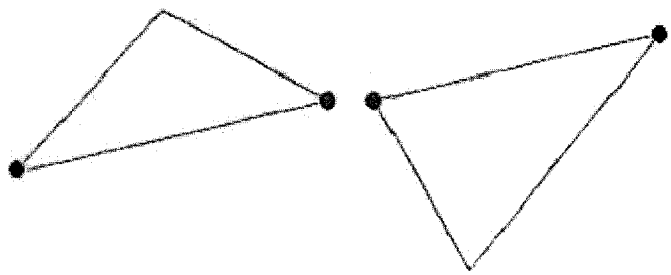

Figure 3.2: The *minimal* distance doesn't consider the whole shape.

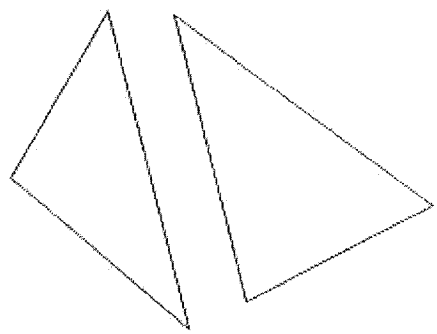

Figure 3.3: The *minimal* distance doesn't account for the position of the objects.

Under this situation, we choose the Hausdorff distance measure which does capture the subtleties, ignored by the *minimal* distance. Named after Felix Hausdorff (1868-1942), Hausdorff distance is the maximum distance of a set to the nearest point in the other set [45]. More formally, Hausdorff distance from polygon P_1 to polygon P_2 is a max-min function, defined as follows:

$$hausdorff(P_1, P_2) = \max_{p_i \in P_1} \{ \min_{p_j \in P_2} \{d(p_i, p_j)\}\} \tag{9}$$

where, p_i and p_j apply to all points inside the polygons, and not only to their vertices.

Another feature of *Hausdorff Distance* is that it is asymmetric as well; in other words, it is oriented. Most of the times, $hausdorff(P_1, P_2)$ is not equal to $hausdorff(P_2, P_1)$. We call equation (9) as *directed Hausdorff Distance*. A more general definition of *Hausdorff Distance* is:

$$Hausdorff\ Distance(P_1, P_2) = \max\{hausdorff(P_1, P_2), hausdorff(P_2, P_1)\} \tag{10}$$

Boundary Simplification based on error predicted with Hausdorff Distance

We use Hausdorff Distance to measure the difference between the original boundary and a simplified one. For all boundary cells, their cost functions includes the boundary changes, defined by Hausdorff distance of original boundary and modified one.

$$\varepsilon_{boundary} = \omega_{boundary} Hausdorff(B, B') \tag{11}$$

where,

B and B' present a subset of boundary, defined by boundary cell and its neighbor set.

As a result, the predicted error of boundary cell and boundary neighbor cell is defined by the following formula:

$$\Delta\varepsilon = \varepsilon_{gradient} + \varepsilon_{shape} + \varepsilon_{boundary} \tag{12}$$

3.3.4 Flipping Problem

Flipping, also referred as fold-over [6], is a very common problem in volume data simplification. It means after one decimation operation, for a cell which is affected by the operation, a new vertex may end up at the opposite side of the plane defined by the other three unchanged vertices. See Figure (3.4).

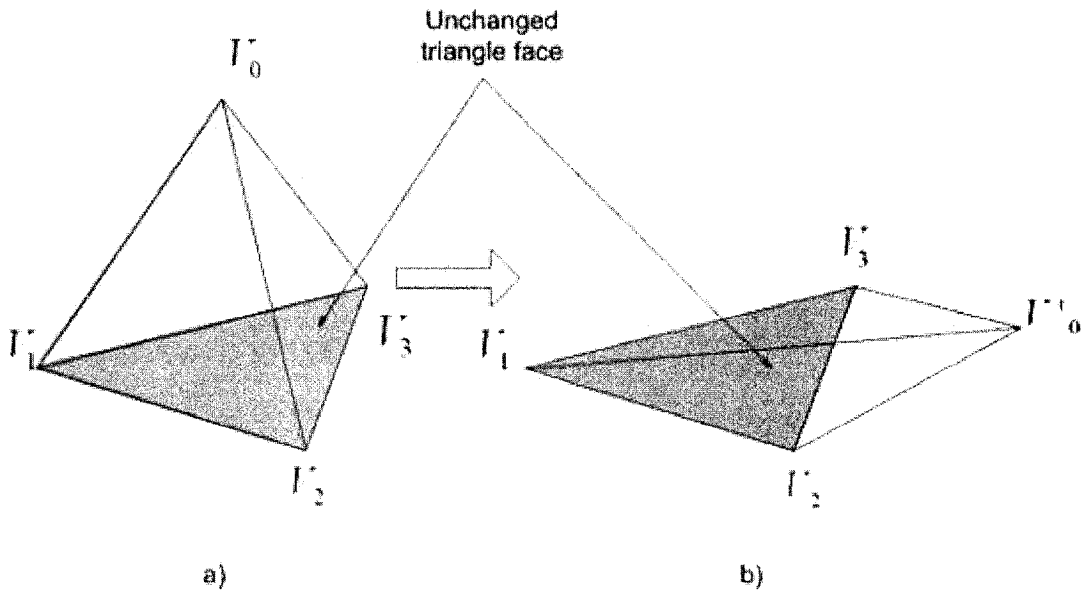

Figure 3.4: Flipping Problem. a)Cell τ before collapsing operation. b)Cell τ' after collapsing operation. Vertex v_0 moved v_0', which is on the other side of triangle face $v_1v_2v_3$

We define the signed volume of a tetrahedron as the parallelepipedial product of its 3 edges $e_1 = \overrightarrow{v_0v_1}$, $e_2 = \overrightarrow{v_0v_2}$ and $e_3 = \overrightarrow{v_0v_3}$:

$$\frac{1}{6}[e_1, e_2, e_3] = \frac{1}{6}e_1 \cdot (e_2 \times e_3) \tag{13}$$

In other words, to avoid the flipping problem, after a decimation operation, we must not produce a tetrahedral cell of negative volume. During the simplification process, we keep a binary flag of volume sign for each tetrahedron. After one collapse

operation, we test, for each $\tau_i \in A(\tau_c)$, whether its signed volume has the same sign (positive or negative) before and after collapsing τ_c. If this is true for each $\tau_i \in A(\tau_c)$, then we allow the collapse of τ_c, otherwise we disallow the collapse operation, and recover the vertex and cell array.

3.3.5 Basic Decimation Algorithm

There are two kinds of decimation strategies: incremental and greedy. For the incremental model, starting with the original tetrahedral mesh $\mathbb{D}_0 : (\Sigma, V, \Gamma, S)$, iterative simplification algorithms generate a sequence of approximations, $\mathbb{D}_1, \mathbb{D}_2 \cdots$, arriving at the final approximation $\mathbb{D}_k$. Here, our definition is different from the progressive mesh notation used by Hoppe [26] where $\mathbb{D}_0$ is the base mesh which, by a sequence of refinements, is transformed into the original mesh $\mathbb{D}_k$. An incremental representation is one which encodes the original model $\mathbb{D}_0$, the final model $\mathbb{D}_k$, and all the intermediate approximations $\mathbb{D}_1, \cdots, \mathbb{D}_{k-1}$. However, the available approximations are restricted to exactly those which were generated during simplification.

The following are the steps for incremental algorithm, which we refer as *Basic Incremental*:

1. Compute the predicted error based on the error metrics discussed in section (3.3.2) for all tetrahedral cells, and put them into a priority queue (we use a heap), ordered by predicted error.

2. While there is still at least one cell remaining in the priority queue with a predicted error which is less than the user specified tolerance, pick the first cell τ, and do the following:

(a) Delete τ from the priority queue.

(b) Determine $A(\tau)$ and $D(\tau)$ for τ.

(c) Determine the position of the new vertex, and update the Vertex Array.

(d) Carry out a flipping check among the cells in $A(\tau)$. If the collapsing operation causes any cell flipping, then recover the original vertices of τ, double the cost of τ, reinsert it into the queue, and skip steps 2e. & 2f.

(e) Delete all cells in $D(\tau)$ from the priority queue and Cell Array.

(f) Remove all cells in $A(\tau)$ from the priority queue and recalculate their predicted errors; then insert them back into the priority queue.

3. Save the simplified tetrahedral mesh.

For the greedy model, starting with the original tetrahedral mesh $\mathbb{D}_0$, we make a pass through the complete tetrahedral structure and attempt collapse operations for all cells whose predicted error is less than an error threshold. The following are the steps for greedy algorithm, which we refer as *Basic Greedy*:

1. Compute the predicted error based on the error metrics discussed in section (3.3.2) for all tetrahedral cells, and put them into a priority queue (we use a heap), ordered by predicted error.

2. While there is still at least one cell remaining in the priority queue with predicted error less than the user specified tolerance, pick the first cell τ, and do the following:

(a) Delete τ from the priority queue.

(b) Determine $A(\tau)$ and $D(\tau)$ for τ.

(c) Determine the position of the new vertex, and update the Vertex Array.

(d) Carry out a flipping check among the cells in $A(\tau)$. If the collapsing operation causes any cell flipping, then recover the original vertices of τ,double the cost of τ, reinsert it into the queue, and skip step 2e.

(e) Delete all cells in $D(\tau)$ from the priority queue and Cell Array.

3. Save the simplified tetrahedral mesh.

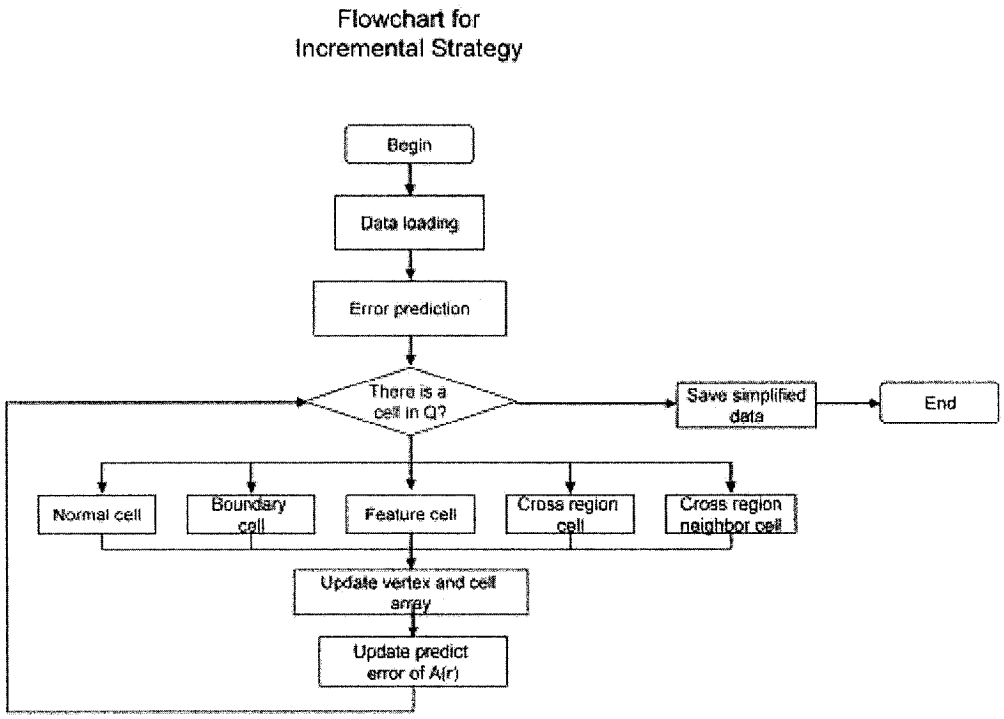

Figure 3.5: Incremental Model

Figure (3.5) and (3.6) demonstrated the processes of both incremental and greedy strategies. Compare with incremental model, greedy one is much more faster. However, it cannot reflect the error changes during simplification processing, and it cannot reach high decimation rates as incremental model.

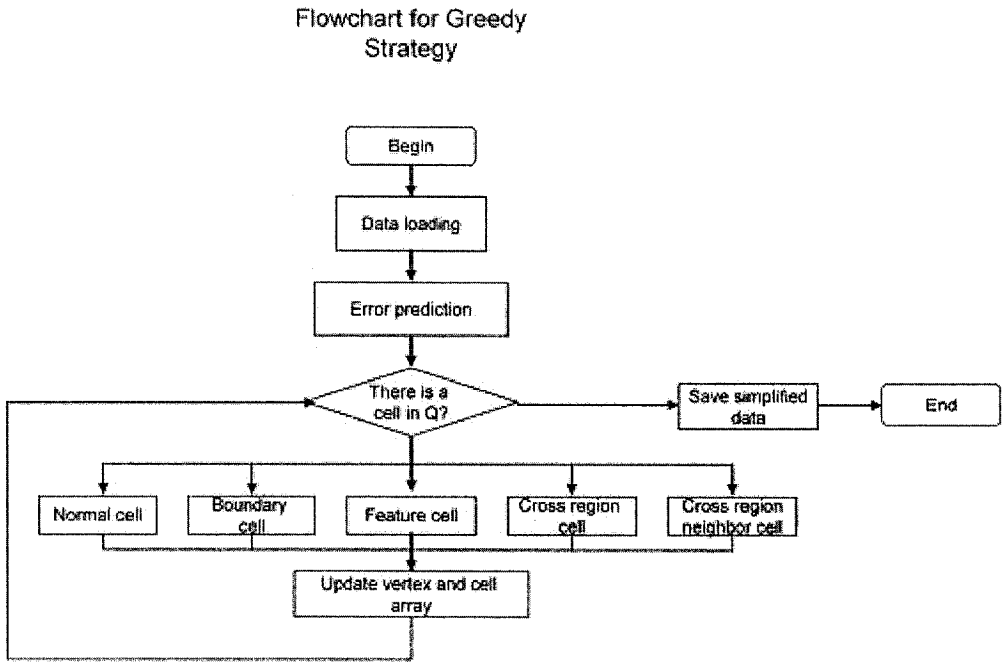

Figure 3.6: Greedy Model

3.4 Multi Level Resolution of Tetrahedral Meshes

A multi level resolution model is a model which captures a wide range of levels of detail of an object and which can be used to reconstruct any one of those levels on demand. The significant advantage of multi-resolution representation is that it allows us to extract a fairly wide range of levels of detail. In our work, we use three methods to generate a set of meshes $\mathbb{D}_0$, $\mathbb{D}_1$, $\mathbb{D}_2$, ... ,$\mathbb{D}_n$ approximating the original mesh at multiple levels of resolution:

1. We begin from the original mesh, and choose a sequence of error bounds $\varepsilon_0 < \varepsilon_1 < \varepsilon_2 < ... < \varepsilon_n$. Then, we select tetrahedra from the priority queue, collapse them, and reinsert the stretched tetrahedra into the queue until all tetrahedra in the queue have $\Delta\varepsilon > \varepsilon_0$. We save the simplified mesh into $\mathbb{D}_1$.

   After obtaining $\mathbb{D}_1$, we start with it and repeat the decimation operation until

we get $\mathbb{D}_2$. We continue these steps until we obtain $\mathbb{D}_n$.

2. Specify a number (or a percentage) of tetrahedra to be collapsed in each step. Then, we perform the incremental decimation algorithm. We keep a variable for remaining number of tetrahedra; when it reaches our limit, we stop decimation. The intermediate meshes are defined by those tetrahedra remaining in the queue after each step. The error for each mesh is stored.

3. As in the work of Gieng *et al.* [19], we remove a set of tetrahedra from the queue and collapse them in parallel. There is a restriction that the neighbor set of the tetrahedra to be collapsed in parallel must not intersect.

The above is for the incremental algorithm. Similarly, we have the following steps for the greedy algorithm.

1. We begin from the original mesh, and choose a sequence of error bounds $\varepsilon_0 < \varepsilon_1 < \varepsilon_2 < ... < \varepsilon_n$. Then, we collapse all tetrahedra from the priority queue whose predicted error is less than ε_0, Then, we save the simplified mesh into $\mathbb{D}_1$.

   After obtaining $\mathbb{D}_1$, we start with it and repeat the decimation operation until we get $\mathbb{D}_2$. We continue these steps until we obtain $\mathbb{D}_n$.

2. Specify a number (or a percentage) of tetrahedra to be collapsed in each step. Then, perform the decimation with greedy algorithm. There is a variable for remaining number of tetrahedra; when it reaches this limit, stop decimation. The intermediate meshes are defined by those tetrahedra remaining in the queue after each step. The error for each mesh is stored.

3. Remove a set of tetrahedra from the queue and collapse them in parallel. There is a restriction that the neighbor set of the tetrahedra to be collapsed in parallel must not intersect.

3.5 Implementation Results

We have implemented both incremental and greedy algorithms on on a Windows Platform with a 2.39GHz Intel Pentium 4 CPU and NVIDIA Quadro4 900 XGL adapter with 128M RAM. Also, we use ZSweep [14] as our rendering algorithm to get the final images.

Table 3.1 shows the parameters used, for both incremental and greedy algoriths, during simplification of SPX dataset with 12,936 cells and 2,896 vertices. The multiple levels of simplified results based on incremental algorithm are shown in Figures (3.7) to (3.11).

$\omega_{gradient}$	ω_{sharp}	$\omega_{neighbor}$	$\omega_{boundary}$
2	1	1	3

Table 3.1: Parameters used during simplification of SPX dataset

	figure 3.7	figure 3.8	figure 3.9	figure 3.10	figure 3.11	figure 3.12
$\Delta\varepsilon$	0	1.492	6.963	14.444	4,338.19	2,239.34
times in second	N/A	0.189	0.335	0.782	1.007	0.125

Table 3.2: The $\Delta\varepsilon$ of multi-resolution of SPX dataset

Figure (3.12) shows the simplified result of SPX dataset with greedy algorithm. The comparison between incremental algorithm and greedy algorithm is shown in Table (3.2). It shows that incremental algorithm can reach higher decimation rates

than greedy one. Also, its result is better than greedy one. The drawback of incremental algorithm is slow speed. It has much more computational time than greedy algorithm. This substantiates our earlier statement that there is always a gap between the very fastest algorithms, which can produce poor approximations, and those algorithms which produce very high quality approximations, but which can be rather slow.

In general, volumetric datasets are full of noise and artifacts. Simplification algorithms should robust to those noise. However, either algorithm *Basic Incremental* or *Basic Greedy* can deal with this problems. Therefore, in next chapter, we introduce new region based feature detection which is obtained by level set segmentation algorithm. Using that, a feature preserving simplification algorithm is proposed after that.

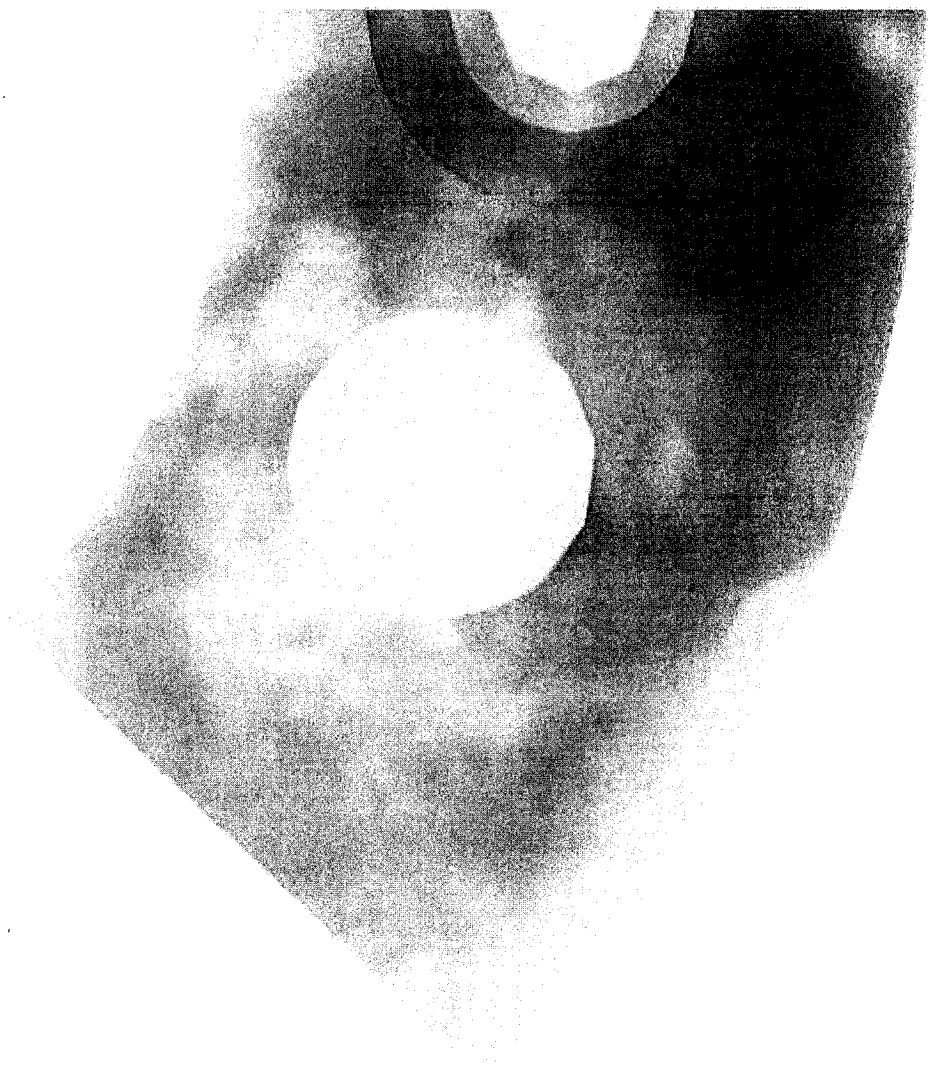

Figure 3.7: The rendering result of SPX dataset with 12,936 cells

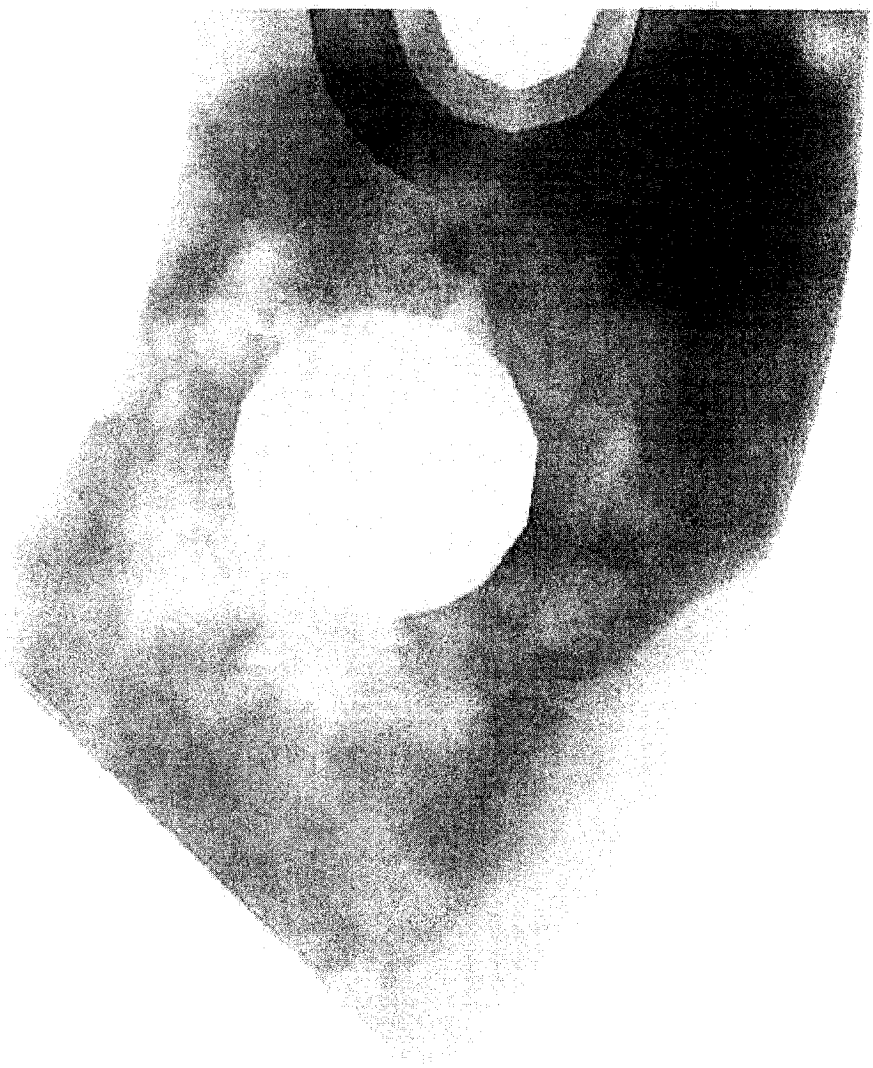

Figure 3.8: The rendering result of simplified SPX dataset with 10,464 cells

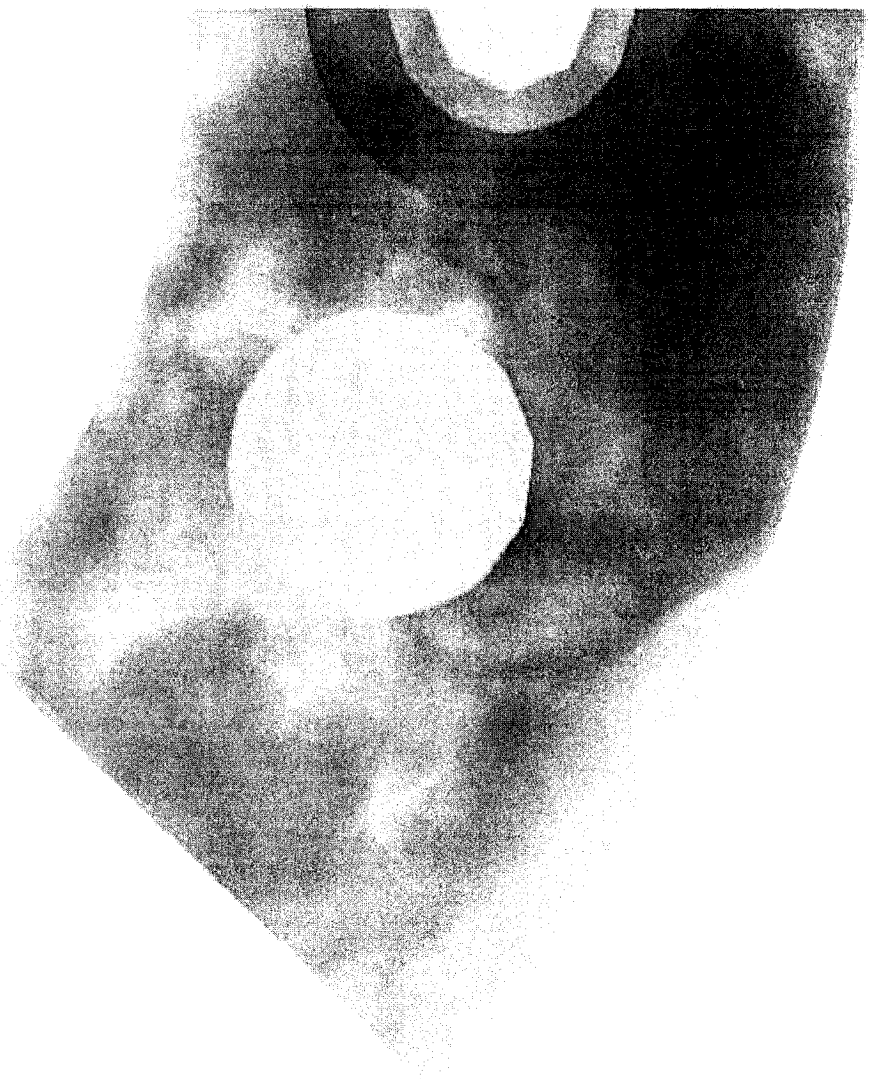

Figure 3.9: The rendering result of simplified SPX dataset with 9,026 cells

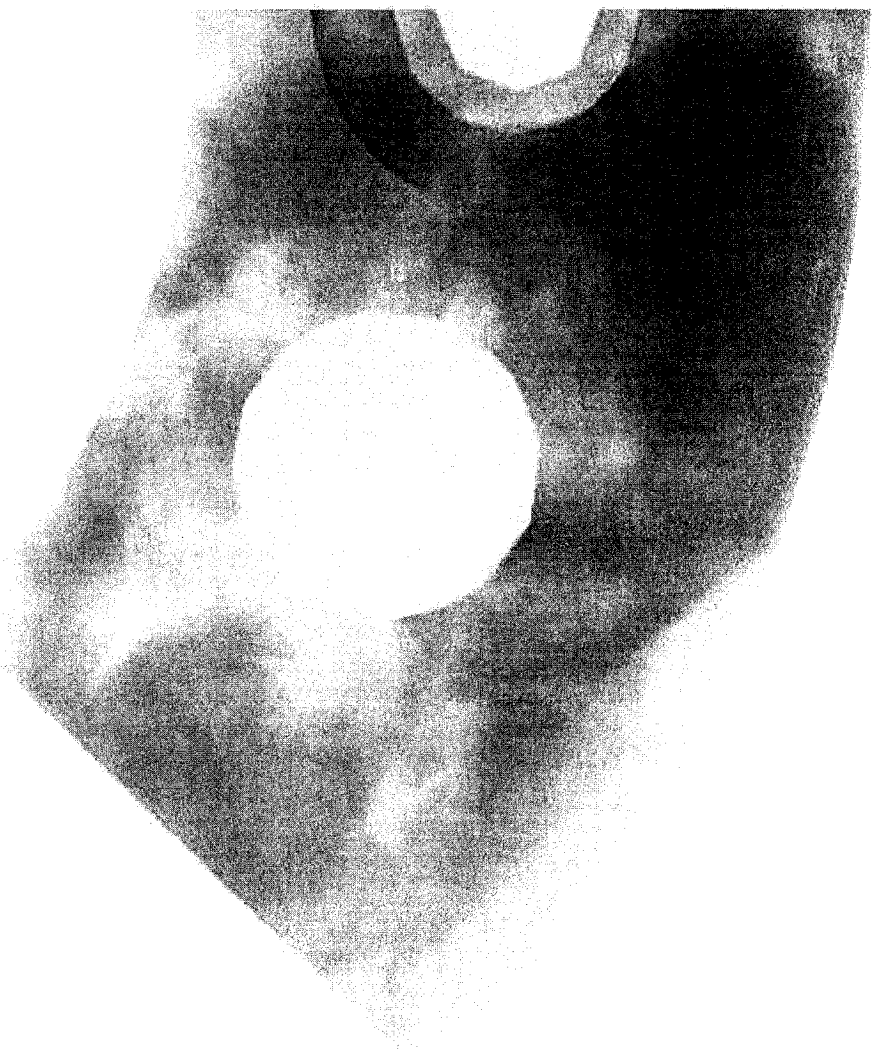

Figure 3.10: The rendering result of simplified SPX dataset with 7,934 cells

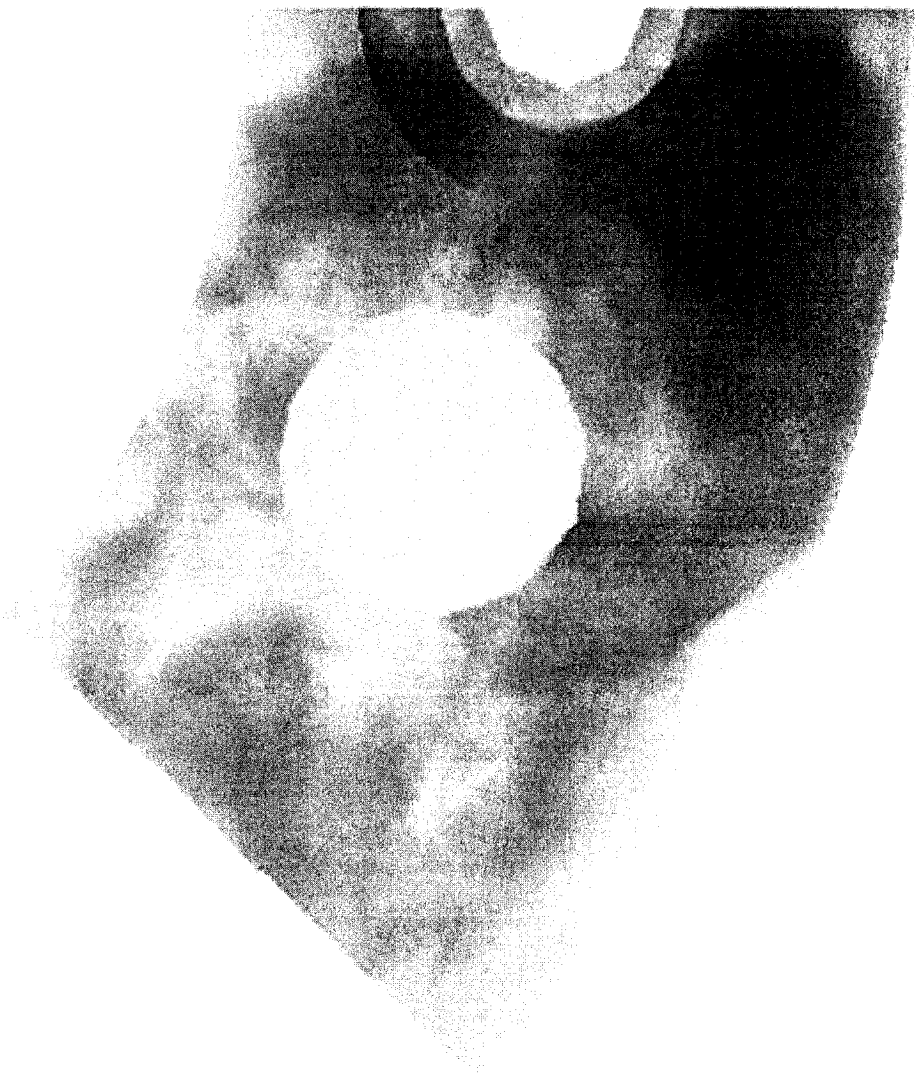

Figure 3.11: The rendering result of simplified SPX dataset with 7,045 cells

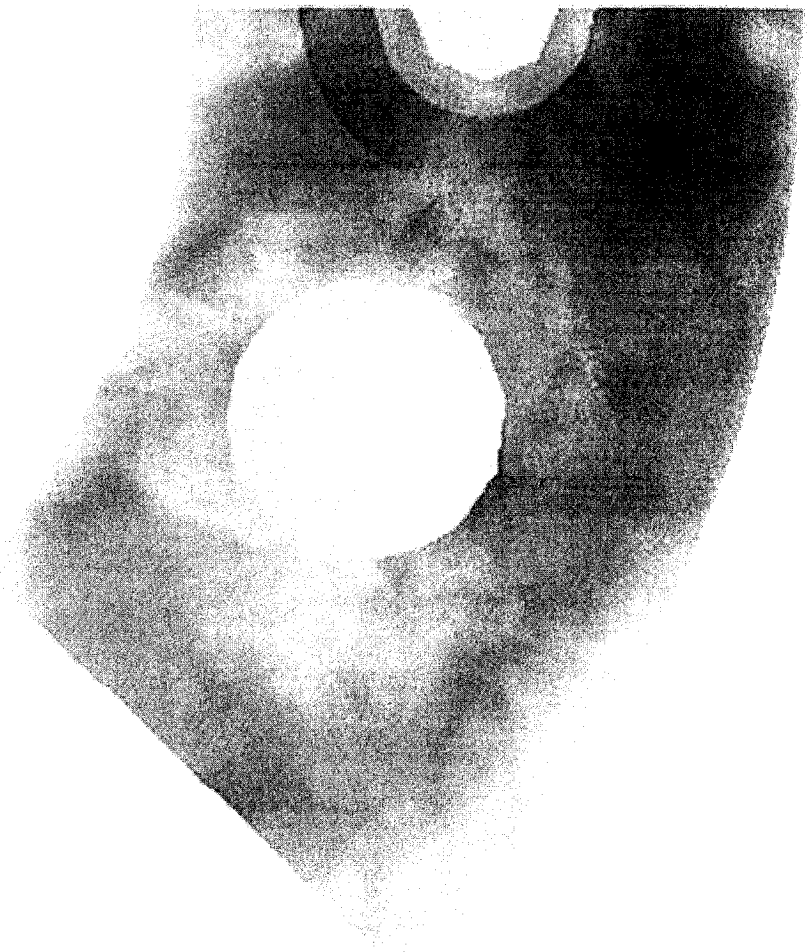

Figure 3.12: The rendering result of simplified SPX dataset with 7,932 cells by greedy algorithm.

Chapter 4

Tetrahedral Mesh Simplification with Feature Preservation

In this chapter, we propose a new method of detecting features with segmentation and use this method to introduce a feature preserving simplification algorithm. We work with the assumption that the interior and exterior boundaries obtained by segmentation contribute more to the final appearance than the rest of the volume. Our method includes two parts: volume data set segmentation (feature detection) and simplification. The main purpose of the first step is to detect the features during a preprocessing phase which should be preserved during the simplification phase.

4.1 Motivation

As discussed earlier in sections (1.1) and (3.1), the task of decimation algorithms is to reduce unnecessary or unimportant information in tetrahedral meshes, while preserving important features. The challenging thing is to decide which part includes

features and which part does not.

Current simplification algorithms tend to keep the boundary information and decimate only interior cells [7, 17, 50]. When they perform the interior decimation, they work with the assumption that large gradient has more potential to contain features. However, this is not enough. Gradient based features tend to be inaccurate in some case such as Figure (4.1). It shows that the classical gradient based edge detection method provides a misleading feature information. Figure (4.1-a) shows the dataset with histogram shown in Figure (4.1-c). A visible edge separates the dataset into two parts. However, by classical gradient edge detection method, the features are represent as several disconnected pieces of horizonal edges.

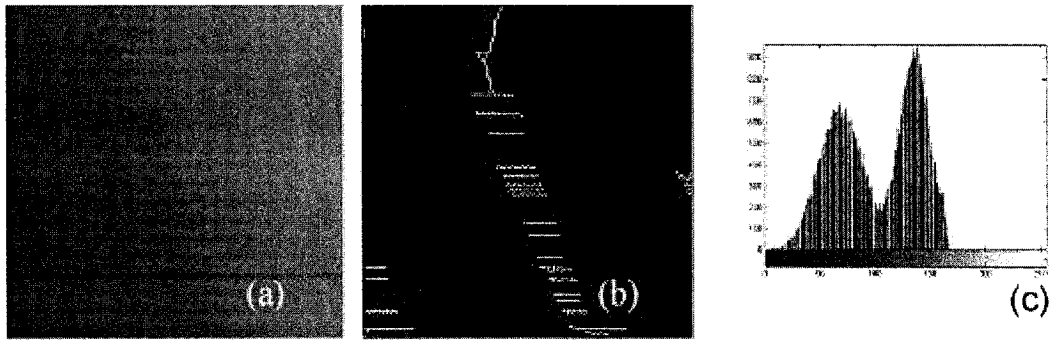

Figure 4.1: Boundaries and features. (a) One slice of Original dataset. (b) Features detected by Gradient based method. (c) Histogram of field values of slice (a).

In addition to gradient based features, in 2003, Chiang and Lu [6] introduced a feature preservation method that preserves the topological structure of iso-surfaces. They restrict decimation operation so that no operations can destroy the topological structure of iso-surfaces. Free decimation operation can only be done on pure cells, which means all vertices of the cells are in the same detected topologically equivalent region and are not critical vertices of Morse iso-surfaces. The downside of their work is that iso-surfaces often contain many topological errors in the form of tiny handles

that are not critical for rendering [24, 60, 2]. It keeps too much noise information and their technique is unable to reach high decimation rates. As shown in their work [6], in most cases, the percentage of the pure cells is too low. For example, they state in their paper, the cleanSPX dataset with 12,936 tetrahedral cells has only 97 pure cells [6].

In this thesis, we present a feature detection method which is based on volumetric segmentation. The segmentation process divides the dataset into several homogenus regions (See Figure(4.2). The vertices defining the boundary between different regions contribute more to the rendered visible features than those vertices in the region interiors. In our simplification phase, we decimate the dataset while preserving these features. By this approach, we potentially preserve more of the significant features than would be possible with the gradient method. Also by simplifying within regions without worrying about topology, we have greater flexibility during the simplification phase while preserving the features. As in all other algorithms, we also prevent flipping of a tetrahedron which is a very common problem during mesh simplification.

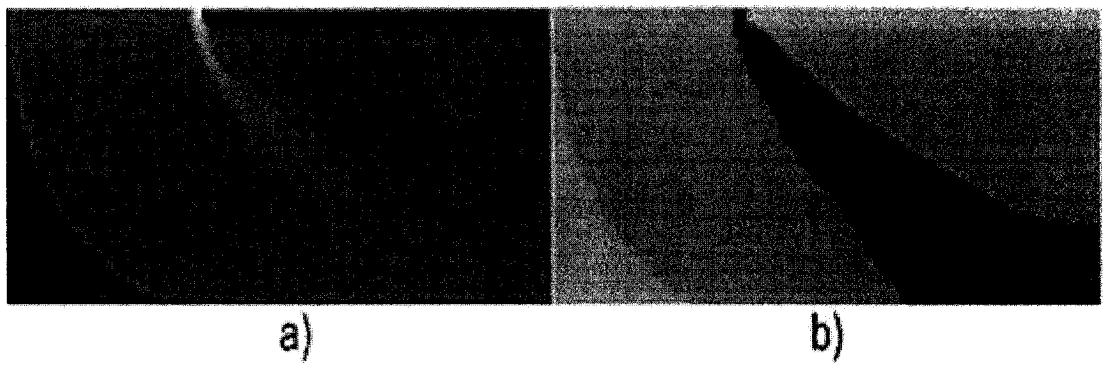

Figure 4.2: a) A slice of bluntfin dataset. b) We can roughly separate the slice into three regions. In each subregion, we have a Gaussian distribution of the attribute values. Collapsing the cells within each subregion will not cause large changes in the original data set.

The method proposed in this thesis has the following significant advantages:

1. In addition to all features detected by classical methods, our method is able to detect more reliable and noise robust boundary features. The decimation is separately performed on each sub-region which has a small range of field values. Thus, collapsing the cells there will not cause huge modification of structure present in the original data.

2. Unlike gradient based method which is not based on the idea of regions, our method can define meaningful regions, which must be simplified separately. For instance, in medical imaging, the segmentation step can give us meaningful regions and boundary, such as bone and tissue, which we must simplify separately.

4.2 Segmentation of Volumetric Dataset

Our feature detection is based on volumetric segmentation. Segmentation is a well studied area, and there exist approaches based on different needs. In general, a segmentation method can be classified into two categories. These are region based and boundary based methods [29]. The former class uses properties of areas of the dataset to choose among possible segmentations, while the latter looks at the properties of the dataset only on the boundary of the proposed segmented regions.

Both methods have their advantages and disadvantages. Region based methods tend to be global, optimizing a functional of the segmentation. On the other hand, they often ignore important boundary properties such as smoothness. Boundary based approaches can treat such properties very naturally, but suffer from their own difficulties. First, most algorithms find only local minima, and thus have no measure of the significance of the extracted boundary for the dataset as a whole. Second,

although there do exist algorithms guaranteed to find global minima, using graph techniques such as dynamic programming and Dijkstra's algorithm, these do not adapt easily to closed contours. Unfortunately, open contours do not segment regions in the image, so that further processing is needed to group the contours into proto-surfaces. Third, boundary based methods cannot incorporate region information such as texture easily. In addition, many of the existing algorithms require initialization by the user in some way, by specifying the end-points of the contour, or by defining an initial contour that then evolves to a solution.

Therefore, in 1988, Osher and Sethian [40] proposed a new segmentation approach based on the class of deformable models, referred as "level set" or "geodesic active contours or surfaces." The application became extremely popular because of its ability to capture the topology of shapes in 3D datasets.

In our experiments, we have consider two very popular and basic methods – thresholding and gradient based edge detection, along with the new approach: level set segmentation; and compare the results.

4.2.1 Thresholding Method

Thresholding approaches segmentation of a dataset by creating a binary partitioning of the attributes values. It is the simplest segmentation algorithm. The key issue in this technique is a single value called threshold. When using one threshold, all points with attribute values greater than the threshold are grouped together into one class and those with attribute values below the threshold are grouped together into another class. Use of a single threshold thus results in a binary segmented volume. Figure (4.2.1) shows a slice of volume dataset and its histogram image.

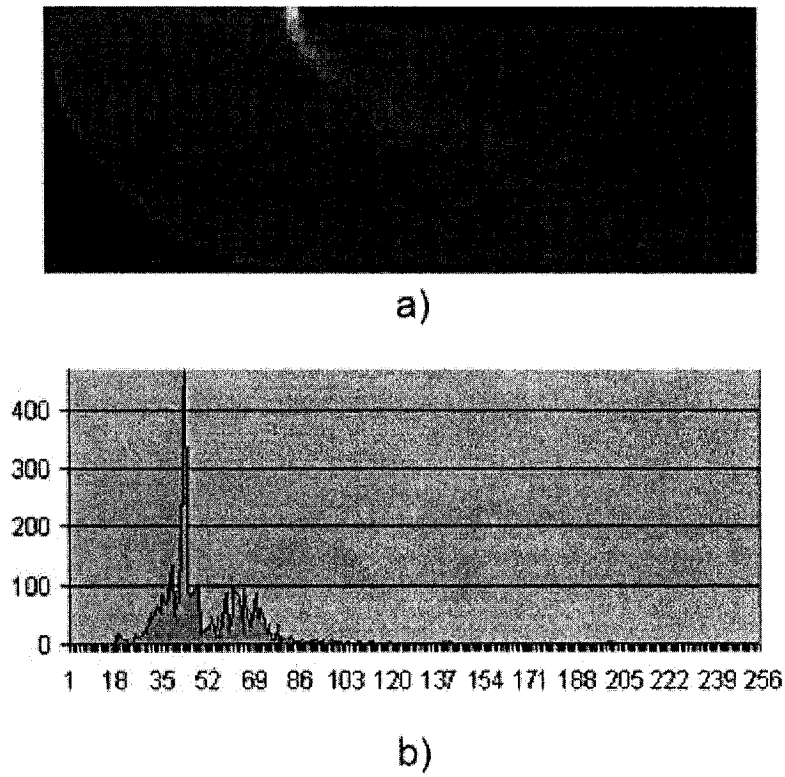

Figure 4.3: Example of thresholding. a) one slice for dataset bluntfin (187,395 cells) b) the histogram of its attributes value distribution.

This technique can be extended to using multiple thresholds, where a region is defined by two thresholds, a lower threshold and an upper threshold. Each voxel of the input volume then belongs to one of the regions based on its attribute value. This technique is known as multi-thresholding [46, 55, 5]. In Figure (4.4) we show the histogram (Figure (4.4-(b)) for the volume (Figure (4.4-(a)). To apply thresholding, we take two thresholds T1 and T2 as shown. We then get three distinct regions as seen from the histogram. Although simple, this technique is very effective in getting segmentation done in volumes with a very good contrast between regions. The main drawback of this technique is that the results are too tightly coupled to the thresholds used. Any change in the threshold values can give a different segmented region. The thresholds are usually generated interactively by using visual feedback. Some

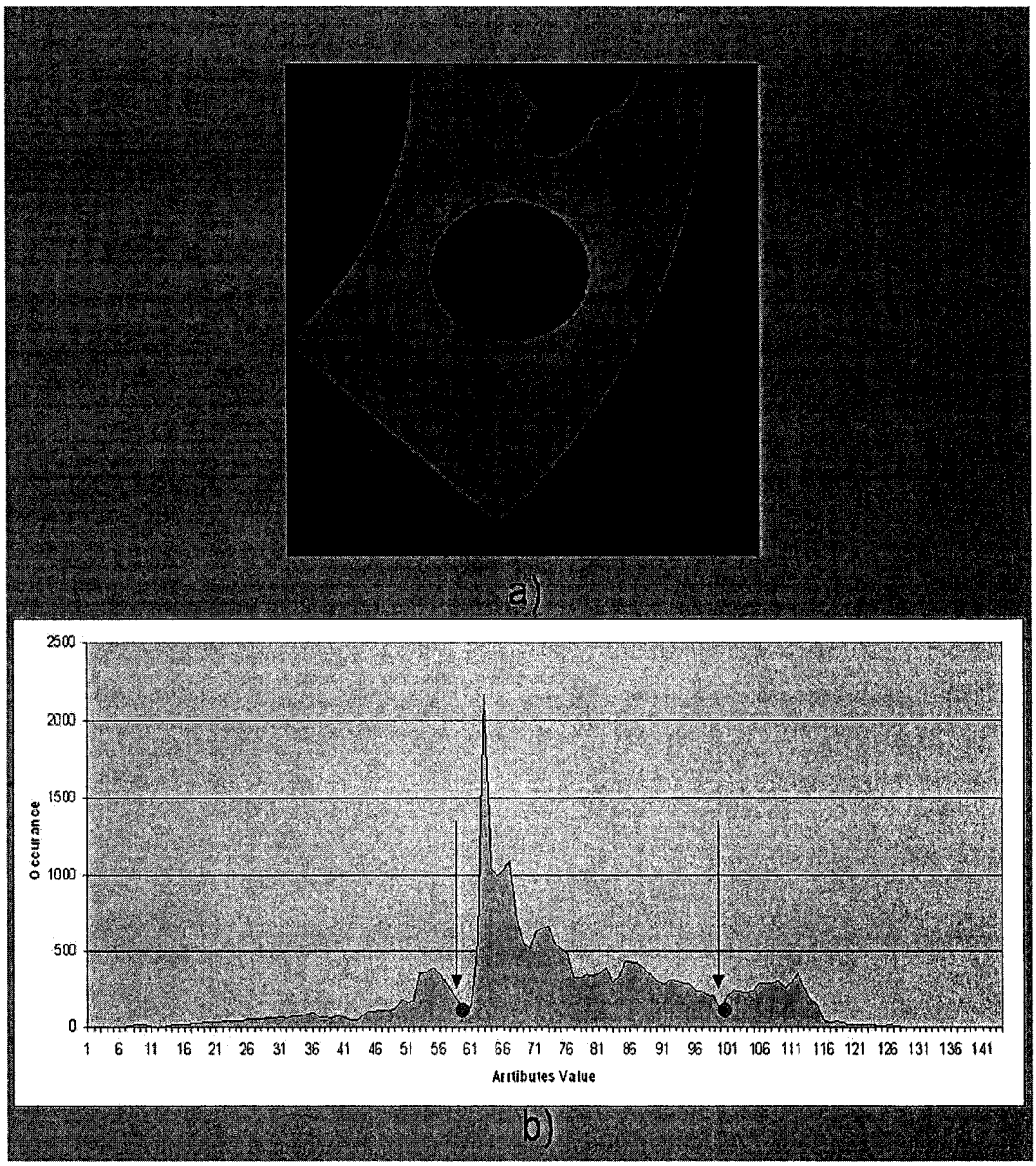

Figure 4.4: Example of multi thresholding. a) The slice image of SPX dataset. b)Histogram of a slice of SPX dataset with two thresholds and dividing the histogram in three regions.

automatic methods do exist with varying degree of success to automate the process of doing correct thresholds [30]. Another drawback which is a direct consequence of the previous one is that the technique is very sensitive to noise and attribute value inhomogeneities. Thus in some cases, the segmentation results are not reliable. This is generally used as the first step towards segmentation of a volume.

4.2.2 Gradient based edge Detection

As a well-developed technique in image processing, gradient based edge detection plays an important role in 3D and 4D segmentation. One advantage of edge detection techniques is that they work very well on datasets with good contrast between different regions. The edges are detected perfectly and can be verified visually. On the down side, these algorithms detect all the edges. It is very difficult to find the correlation between the edges and the regions of interest. In addition, these algorithms do not perform well on datasets with low contrast between regions. These algorithms are also susceptible to noise. In most of the cases, these algorithms are not used on their own for segmentation, but are coupled with other segmentation algorithms to solve a particular segmentation problem.

A number of gradient based edge detecting operators have been proposed until now. Liu [35] proposed a 3D surface detection algorithm that extends the classical Robert's operator into 3D space. Herman and Liu [23] later extended this algorithm to 4D. Zucker and Hummel [64, 65] developed an optimal three-dimensional edge detection operator, which was essentially a Sobel Operator. Figure (4.2.2) is the segmentation of one slice of SPX with 12,936 cells with the Sobel edge detector.

From the results, we come to the conclusion that this method can only give the

Figure 4.5: Gradient-based edge detection of one slice of SPX with 12,936 cells with Sobel edge detector. a) One slice of data SPX; b) Segmentation result by Sobel edge detector.

boundary in local view. Also, it is too sensitive to noise. It is not really suitable for our purpose.

4.2.3 Level set method

Proposed by Osher and J. Sethian [40], level set methods have attracted more and more attention of researchers from different areas. Ω is defined as a bounded open subset of $\mathbb{R}^2$, and $\partial\Omega$ denote its boundary. Let $u_0 : \overline{\Omega} \Rightarrow \mathbb{R}$ be a given slice of volumetric dataset, and $C(s) : [0,1] \Rightarrow \mathbb{R}$ be a parameterized curve.

The curve C is represented implicitly via *Lipschitz Function*, which is also referred as level set function, ϕ, by $C = \{(x,y)|\phi(x,y) = 0\}$. The evolution of the curve is given by the zero-level curve at time t. And evolving of the curve C is controlled by the function $\phi(t,x,y)$ in normal direction with speed F.

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| F \tag{14}$$

$$\phi(0, x, y) = \phi(x, y) \tag{15}$$

where:

the set $C = \{(x, y) | \phi_0(x, y) = 0\}$ defines the initial contour.

A particular case is the motion by mean curvature. This is given by equation (16) as the curvature of ϕ passing $through(x, y)$

$$F = div\left(\frac{\nabla \phi(x, y)}{|\nabla \phi(x, y)|}\right) \tag{16}$$

As a consequence, the equations become the following:

$$\begin{cases} \frac{\partial \phi}{\partial t} = |\nabla \phi| div\left(\frac{\nabla \phi}{|\nabla \phi|}\right), & t \in (0, \infty), \quad x \in \mathbb{R}^2 \\ \phi(0, x, y) = \phi_0(x, y), & x \in \mathbb{R}^2 \end{cases} \tag{17}$$

where ϕ_0 is initial level set function.

Level Set Segmentation

Level set segmentation has a significant advantage that it is robust to noise and sampling artifacts which is a typical problem of 3D datasets. These noise and artifacts cause considerable difficulties when applying classical segmentation techniques such as edge detection and thresholding. These techniques either fail completely or require some kind of postprocessing step to remove invalid object boundaries in the segmentation results. As described in [47], let Ω be an open domain subset of $\mathbb{R}^3$ with a

smooth boundary. Let$\phi_0 : \Omega \Rightarrow \mathbb{R}^3$ represent the observed data function. And let Ω_i be the region defined as follows:

$$\Omega_i = \Big\{ x \in \Omega | \quad x \text{ belongs to the } i_{th} \text{ region} \Big\} \tag{18}$$

1. A partitioning of Ω consists of finding a set $\Big\{\Omega\Big\}_{i=1...k}$ such that:

$$\Omega = \bigcup_{i=1}^{k} \Omega_i \quad \text{and} \quad \Omega_i \bigcap_{i \neq j} \Omega_j = \phi.$$

2. The partition $\Big\{\Omega_i\Big\}$ is a classification of the observed data φ_n and takes into account the **Gaussian Distribution** property of the classes:

$$\text{minimize} \sum_i \int_{\Omega_i} \left(\frac{\varphi_0 - \mu_i}{\delta_i} \right)^2 \quad \text{with respect to } \Omega_i.$$

As we defined previously $\partial\Omega_i$ is the boundary of Ω_i. Here, we further define $\Gamma_i = \partial\Omega_i \cap \Omega$ as the boundary of Ω_i within domain Ω, and the interface between Ω_i and Ω_j is given by:

$$\Gamma_{ij} = \Gamma_{ji} = \Gamma_i \cap \Gamma_j \cap \Omega, \qquad \forall i \neq j$$

Therefore, we have:

$$\Gamma_i = \bigcup_{j \neq i} \Gamma_{ij}$$

See Figure(4.6).

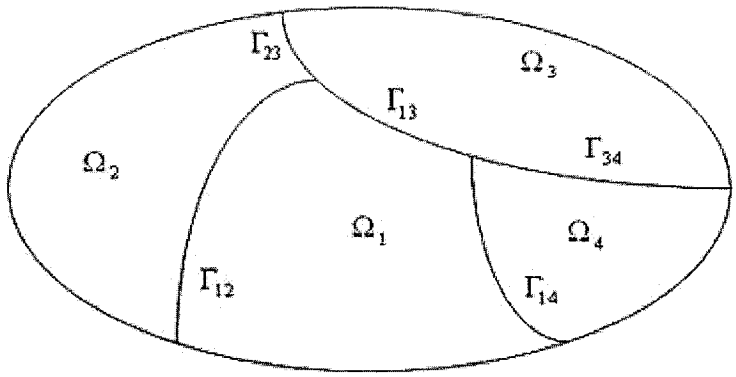

Figure 4.6: An example partition of region Ω.

The level set function is modified from Samson et al. [47]. It achieves segmentation by variational level set method [63, 4, 57]. Thus, it can detect some features that a local gradient method cannot. As described in Tsai's work [54], the robustness to noise makes the boundary more accurate.

In 2000, Samson *et al.* proposed a supervised classification model to find a partition composed of homogeneous regions, assuming that the number of classes and their attribute value properties are known. Therefore, for K region segmentation, the proposed method used K level set functions $\phi_i : (i \in [1, K])$ to represent each of them as shown in equation (19), which shows the energy function we used. The energy function consists of three terms: minimal variance energy E_{minv}, minimal length energy E_{minl}, and non-overlap energy $E_{nonover}$. Equation (20) shows the level set function we use.

$$E(\phi_i, ...\phi_k) = E_{minv}(\phi_i, ...\phi_k) + E_{minl}(\phi_i, ...\phi_k) + E_{nonover}(\phi_i, ...\phi_k) \qquad (19)$$

where:

$$
\begin{aligned}
E_{minv}(\phi_i, ...\phi_k) &= \sum_{i=1}^{k} e_i \int_{\Omega} H_\alpha(\phi_i)(1-c_i)^2 dxdy \quad \forall i, e_i \in \mathbb{R} \\
E_{minl}(\phi_i, ...\phi_k) &= \sum_{i=1}^{k} \gamma_i \int_{\Omega} \delta_\alpha(\phi_i)|\Delta\phi_i| dxdy \\
E_{nonover}(\phi_i, ...\phi_k) &= \frac{\lambda}{2} \int_{\Omega} \Big(\sum_{i=1}^{k} H_\alpha(\phi_i) - 1 \Big)^2 dxdy
\end{aligned}
$$

$$
\phi_i^{t+1} - \phi_i^t = \Delta t \cdot \delta_\alpha(\phi_i^t) \cdot \Big[v_i div\Big(\frac{\Delta\phi_i}{|\Delta\phi_i|}\Big) - e_i(I-c_i)^2 - \beta\Big(\textstyle\sum_{i=1} k H_\alpha(\phi_i) - 1 \Big)^2 \Big] \tag{20}
$$

In equation (19), $H_\alpha(\cdot)$ is the ***Heaviside Function*** and c_j are the mean of positive areas in level set function ϕ_k. In equation(20), $\delta_\alpha(\cdot)$ is the direct delta function and v_i and β are constants.

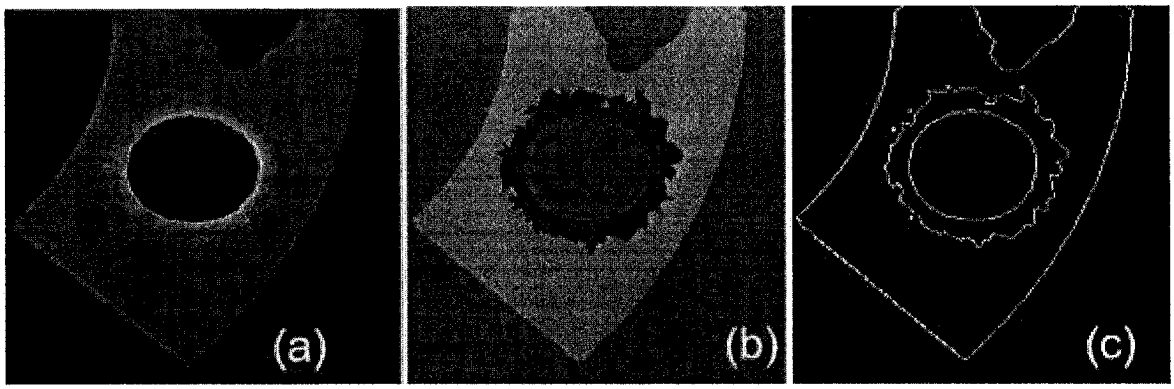

Figure 4.7: Segmentation result by coupled level set method. a) Detected interior boundary for SPX dataset. b) Sub-regions based on the detected interior boundary for SPX dataset (shown in slice).

After the segmentation, we obtain several sub-regions each with distinct Gaussian distribution of attributes. Our premise is that vertices on the boundary between different regions are likely to demark visible features. Therefore, during the tetrahedra decimation process, we seek to preserve as many of these vertices as possible. In Figure

(4.7), we see a segmentation result by coupled level set function of SPX dataset.

4.3 Tetrahedral Mesh Simplification with Region Based Feature Preservation

4.3.1 Definition

Before we begin the discussion of feature preserving simplification, we introduce some definitions which we use later.

- Feature Vertex

  If $\forall v_i \in V$ is on the boundary detected by segmentation phase, we label it as a ***Feature Vertex.***

- Feature Cell

  If $\forall \tau_i \in \Gamma$ has one or more than one feature vertex v_i, we label it as a ***Feature Cell.***

- Cross Region Cell

  If $\forall \tau_i \in \Gamma$ has vertices in more than one sub-region, we label it as a ***Cross Region Cell.***

- Cross Region Neighbor Cell

  If $\forall \tau_i \in \Gamma$ has a neighbor cell $\tau_j \in Neighbor(\tau)$, which is a cross region cell, we label it as a ***Cross Region Neighbor Cell.***

4.3.2 Feature Detection

In our approach, we divide the 3D space defined by the original dataset into several sub-regions defined by a level set segmentation of the volume. Inside each sub-region, its field values follow a Gaussian distribution in 3D. During the simplification phase, we simplify each sub-region individually, and preserve the boundary between neighboring sub-regions. Because the field value distribution of each sub-region has a specific range, any change in geometric information within each sub-region cannot affect the field value distribution outside of this range (neighboring sub-regions have their own distributions of field values). By maintaining the boundaries between these simplified sub-regions, the features of the original dataset are also preserved. We use an image processing method based on volume segmentation to detect region boundaries inside the 3D volume. These boundaries are more meaningful, and we believe are features to be preserved.

Mapping feature point from regular mesh to irregular mesh

Currently, our algorithm performs the level set segmentation on a regular mesh. In other words, we classify the feature points on regular mesh which we generate from irregular mesh. We define two ways to transfer the feature points back to the original mesh.

1. Mapping from regular $\mathbb{C}$ to irregular mesh $\mathbb{D}$: For each point in regular mesh, we define a mapping function f, which defines only one vertex in irregular mesh. When the resolution of the regular mesh $\mathbb{C}$ is much bigger than the irregular one $\mathbb{D}$, the mapping function may map several points in regular mesh onto only one vertex in the irregular mesh. Therefore, for each vertex v_i in irregular mesh,

we get a set $s = \{p_j = f^{-1}(v_i) | p_j \in \mathbb{C}\}$. We classify that, for each vertex in irregular mesh $\mathbb{D}$, if any point inside the set s is a feature point, then the vertex is a feature vertex. The nearest neighbor algorithm turns out to be a good choice for the mapping function.

2. Mapping from irregular to regular mesh: We first cut the regular mesh into small cubes, which use the sample points as the vertices. If any cube contains at least one feature point, we define it as a feature cube. Then we map the vertices in irregular mesh into the regular mesh. If a vertex of irregular mesh is inside a feature cube, we classify the vertex as a feature point in irregular mesh. If the vertex is mapped to the corner or on the edge, we define a cube set that the vertex belongs to. If any cube in the set is a feature cube; we classify the vertex as a feature point in irregular mesh. Here, we may meet a problem when the resolution of regular mesh is much higher than the irregular one, for example 100 times. To solve this problem, we reasonably increase the size of the cube, such as every two vertices or more vertices define a corner of one cube.

The results of feature vertex detection is shown in results section 5.1.

Very recently Xu *et al.* [61] have described a technique for level set segmentation directly on irregular meshes. This would help avoid the additional conversion steps involved in our preprocessing phase.

4.3.3 Feature Preservation

Both exterior and interior features are important for volumetric datasets. We have already defined a way to decimate the exterior boundary while preserving features

in section (3.3.3). After feature detection, a number of rules are defined to preserve interior features.

To preserve the interior features, we perform the decimation for each sub-region $(\mathbb{D} : (\Sigma, V, \Gamma, S))$ disjoint from the other regions. We also perform the collapse operation on the cell whose predicted error is currently the smallest. We update the predicted error of each affected cell after each tetrahedral collapse, and we continue the decimation process until the user specified error tolerance τ is met. To ensure preservation of the interior and exterior features of the original dataset, we include a set of additional tests to forbid any collapse operation that will cause a change in the boundary of sub-regions. These tests classify three types of interior cells that need special attention.

1. Feature Cells: If any τ has more than one feature point, we do not perform the collapse operation on it. If τ has exactly one feature point v_f, we define the collapse operation as collapsing τ to v_f. We denote this operation as $\tau \rightarrow v_f$. Figure (4.8) demonstrates this operation.

2. Cross-region Cells : If any τ has vertices in more than one sub-region, we do not perform the collapse operation on it.

3. Cross-region Neighbor Cells: For any τ, if its $A(\tau)$ or $D(\tau)$ contains only one cross region cell, doing a collapse operation to the centroid will cause the change of the structure of the sub-region. To avoid this problem, we define the collapse operation for such a cell as the collapse of τ to the vertex v_n which is the common vertex with the cross-region cell. We denote the operation as $\tau \longrightarrow v_n$. If its $A(\tau)$ or $D(\tau)$ contains more than one feature cell and more than one feature

vertex, we do not collapse the cell. Cells in $D(\tau)$ may contain any number of feature points.

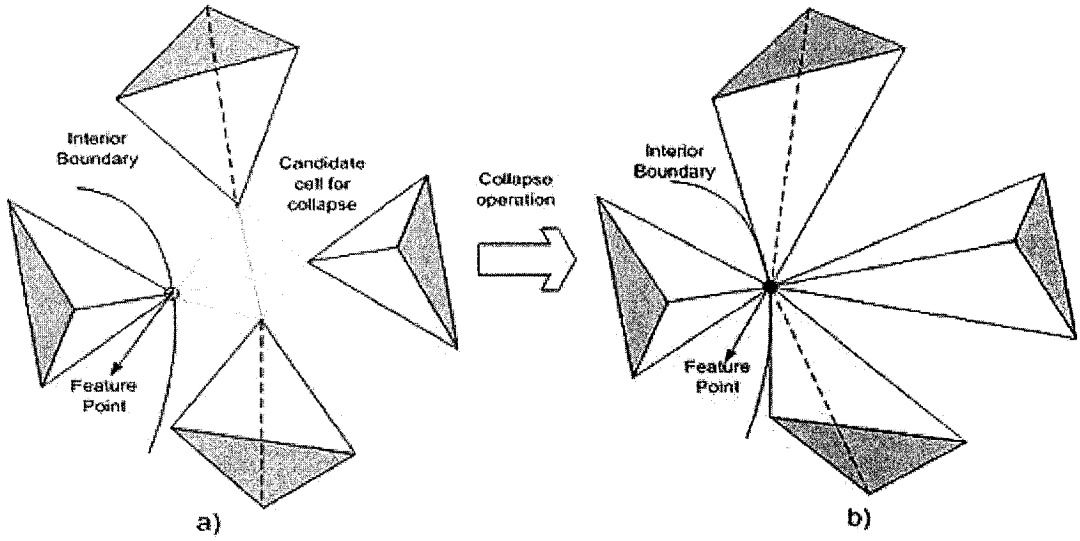

Figure 4.8: Demonstration of decimation operation with feature preservation. a) The gray cell in the center is the candidate of collapsing operation. It contains one feature vertex. The surrounding cells are tetrahedra inside $A(\tau)$). b) After collapse operation, the candidate collapses to the feature vertex. The surrounding cells stretch themselves to the feature vertex.

The above tests are defined so as to ensure that there is no large change in interior boundaries. To ensure the preservation of the geometry of exterior boundary, we define a boundary cell check as follows:

4. Boundary Cells: If τ has exactly one vertex on the boundary, we define the decimation operation as a collapse of τ to v_b (the vertex on the boundary). We denote the operation as $\tau \longrightarrow v_b$. If any τ has more than one vertex on the boundary, we do not perform the collapse operation on it.

We denote all cells that are not feature cells, cross region cells, cross region neighbor cells or boundary cells as normal cells. The normal cells may be collapsed freely.

Here we define the collapse operation for normal cells as $\tau \longrightarrow v_c$, where v_c is the centroid of the cell. By choosing the centroid, the volume of the deleted cell is distributed evenly amongst the remaining local neighboring cells.

We also define a flipping check to ensure no flipping occurs during the simplification process.

5. Flipping Check: For any cell inside the $A(\tau)$, where $\tau \longrightarrow v_i$, v_i should stay on the same side of its facing triangle. We test all the cells in $A(\tau)$ and check whether its signed volume has the same sign (positive or negative) before and after the collapsing operation. If any cell changes its volume sign, we do not allow the collapsing operation and return the dataset to the previous state before the collapse.

4.3.4 Error Prediction

In section (3.3.2), we have already stated the cost function of error predict in equation (7). However, after we segment the volume, decimation operation may also affect its regions. The error function defined in equation (7) is not adequate to reflect region based error. As a result, we define *region Level Error Measurement* ε_{region}:

$$\varepsilon_{region} = \omega_{region} \frac{\delta_c}{\delta_N} \tag{21}$$

where: δ_N is the variance of all vertices in region N

$$\delta_N = \frac{1}{N} \sum_{v_i \in N} (s_i - \overline{s})^2$$

δ_c is the variance for four vertices of collapse candidate c

$$\delta_c = \frac{1}{4}\sum_{i=0}^{3}(s_i - \overline{s})^2$$

$$\Delta\varepsilon = \varepsilon_{region} + \varepsilon_{gradient} + \varepsilon_{shape} + \varepsilon_{boundary} \tag{22}$$

We compute the predicted error using a combination of region level and local error functions (shown in Equation (22)) and use it to prioritize cells in order of decimation during the simplification. It may be argued that if the cell is inside an attribute region that is composed of several disconnected parts, then each vertex of the cell may be inside different parts. In that situation, based on our test, the cell would be a normal cell; however, it is a cross boundary cell. Definitely, it is possible that one region is composed of several disconnected parts, but the situation that is mentioned earlier is impossible. That is because, attribute value distribution within a cell is defined by the vertices of the cell by linear interpolation. Therefore, the extreme points could only appear on the sample vertices [18]. In other words, if all vertices of a cell are within a region X, the whole cell is also in that region. When we perform the collapsing, the new vertex will also be in the same region.

4.3.5 Simplification Algorithm

Similar to the algorithms described in section (3.3.5), the core of the tetrahedron collapse algorithm takes a tetrahedral mesh ($\mathbb{D}_0 : (\Sigma, V, \Gamma, S)$) as its input, and outputs a new simplified mesh ($\mathbb{D}_1 : (\Sigma, V, \Gamma, S)$) with reduced geometric and topological information. Below we give an overview of our incremental simplification algorithm,

which we refer as *FeatureIncremental*:

1. Compute the predicted error based on the error metrics discussed in section (4.3.4) for all tetrahedral cells, and put them into a priority queue, ordered by predicted error.

2. While there is still at least one cell remaining in the priority queue, with predicted error less than the user specified tolerance, pick the first cell τ, and do the following:

   (a) Delete τ from the priority queue.

   (b) Determine $A(\tau)$ and $D(\tau)$ for τ.

   (c) Check the type of the current cell τ : Feature Cell, Cross-region cell, Cross-region Neighbor Cell, Boundary Cell or Normal Cell, as described in the previous section.

   (d) If the cell is cross-region cell, or has two or more feature vertices, skip steps 2e. 2g. 2f. and 2h. Otherwise, based on the type of the cell, determine the position of the vertex to replace τ, and replace its original vertices with the replacement vertex.

   (e) Carry out a flipping check among the cells in $A(\tau)$. If the collapsing operation causes any cell flipping, then recover the original vertices of τ and skip steps 2g., 2f. and 2h.

   (f) Detect whether the new generated vertex is a feature vertex or not.

   (g) Delete all cells in $D(\tau)$.

   (h) Remove all cells in $A(\tau)$ from the heap and recalculate their predicted errors; then insert them back into the priority queue.

3. Save the simplified tetrahedral mesh.

4.4 Decimation Algorithm Combined with Edge Collapse

In the *FeatureIncremental* algorithm, we successfully preserve both interior and exterior boundary by not touching feature vertices and cells. However, these steps prevent us from achieving high decimation rate. For example, for SPX dataset with 12,936 cells, the highest decimation rate is 60%. And we expect much higher rates for decimation algorithms. The main reason is we label all boundary cells as unchangeable. In order to avoid this, we combine cell collapse with edge contraction, which gives us more freedom when we perform decimation.

4.4.1 Decimation Algorithm

When we encounter a cross region cell, as we have defined before, we do not allow a collapse operation. However, there maybe some edges on with we can perform a edge collapse operation.

We calculate the cost for edges within $Neighbor(\tau)$.

$$\varepsilon_{edge} = |s_{v_i} - s_{v_j}| \cdot |v_i - v_j| \tag{23}$$

Where s_{v_i} is the scalar value of vertex v_i. Term $|s_{v_i} - s_{v_j}|$ refers to the difference between the scalar value of vertices v_i and v_j. Term $|v_i - v_j|$ refers to the distance between vertices v_i and v_j.

Therefore, we change check (2) in section (4.3.2) to the following:

2. Cross-region Cells: If any τ has vertices in more than one sub-region, we first detect the neighbor set of the cell. And then, we use the equation (23) to sort all edges which have their two vertices in the same region. And pick the one with the least ε_{edge} to perform edge collapse operation.

Note that, the edge collapse operation can also be used when we encounter the flipping problem.

4.4.2 Topologically Critical Points of Isosurface

To obtain high decimation rate, our labelling of all feature vertices as unchangeable is not reasonable. We need some measure to sort the importance of feature vertices also, and allow some of them to be decimated. Here we consider the topological structure of the volumetric dataset.

Morse Theory

The topology of the scalar field is characterized by its *critical point.* A point x is a *critical points* on the C^2 scalar function $f(x)$ if all first order partial derivatives of f evaluated at x are zero, i.e., $\Delta f = 0$. Gerstner and Pajarola [18] considered piecewise linear interpolation applied to tetrahedral grids, which leads to C^0 continuous functions, and showed that *critical points* can only occur at mesh vertices.

Connected components in an edge graph of a surrounding polyhedron correspond to connected components in a neighborhood of a vertex. From this observation, Weber et. al. [58] have given a definition for *regular and critical points* based on the number of connected components in their local neighborhood. Assume the number

of "positive" connected components surrounding point x is n_p, and the number of "negative" connected components surrounding point v_i is n_n. If $n_p = n_n = 1$, the vertex v_i is a regular point. If $n_p = 1$ and $n_n = 0$, vertex v_i is a minimum. If $n_n = 1$ and $n_p = 0$, vertex v_i is a maximum. If $n_n + n_p > 2$, vertex v_i is a saddle.

To classify an *internal* vertex v, we need to carry out the following steps:

1. We take all the tetrahedral cells sharing v as one of their cell vertices. For each such cell, there is one triangle without v as a vertex. We take all such triangles, whose vertices are exactly the neighbors of v and whose edges connect these neighbors together. These triangles then form a graph G with nodes and edges being the vertices and the edges of the triangles respectively.

2. For each node p in G, we classify p as " + " if its scalar value is larger than the scalar value of v. Similar, if its scalar value is smaller than v, we classify it as " - ".

3. Remove all edges connecting two nodes of opposite signs, and obtain a new graph G'.

4. Using the depth-first search algorithm, we compute the number of connected components with the same sign in G'. Then check:

   (a) If there is exactly one positive component and one negative component, v is a normal point.

   (b) If there is only one positive component, v is a minimum point.

   (c) If there is only one negative component, v is a maximum point.

   (d) If there are more than two components, v is a saddle point.

	Number of topologically critical vertices	Number of Feature Vertices
CleanSPX	852	1,694
Blunt Fin	1,799	5,698

Table 4.1: Number of topologically critical vertices of our testing datasets

Table (4.1) shows the number of vertices which are both region based feature vertices and topologically critical vertices.

Rank Feature Vertices

We sort all feature vertices into four levels (0-3) based on following criteria:

1. If the feature vertex is a topologically critical point, we give it the highest rank 3.

2. If the feature vertex shares at least one edge with a topological critical point, we rank it is as 2.

3. If the feature vertex shares at least one edge with other feature vertices, we rank it as 1.

4. If a feature vertex does not share any edges with other feature vertices, we rank it as 0.

As a consequence, the check (1) in section (refsc:featureD) changes to following:

1. Feature Cells: If τ has exactly one feature point v_f, we define the collapse operation as collapsing τ to v_f. We denote this operation as $\tau \rightarrow v_f$. If any τ has more than one feature vertex, we check the rank of each feature vertex, and collapse the cell to the one that has the highest rank. If more than one

feature vertex has the highest rank, we choose the vertex with larger number of shared edges with other feature vertices. If the vertices still have the same number of edge sharing feature vertices, we randomly pick one of these vertices as the vertex after collapsing.

In next chapter, we show the implementation result of our feature preserving simplification algorithm.

Chapter 5

Implementation Results

We present in this chapter the results from an implementation of our simplification algorithm applied to a number of bench mark tetrahedral meshes. We have implemented our algorithm on a Windows Platform with a 2.39GHz Intel Pentium 4 CPU and NVIDIA Quadro4 900 XGL adapter with 128M RAM. Also, we use ZSweep [14] as our rendering algorithm to get the final images. The bench mark test data sets are SPX with 12,936 elements, Blunt Fin with 187,395 elements, nucleon with 320,000 elements and CT-head with 3,951,605 elements.

5.1 Experimental Result of Feature Preserving Simplification Algorithm

Table (5.1)gives the details of the 3 volume data sets on which we have applied our simplification algorithm. Table (5.2) shows the number of feature points detected by the level set segmentation process. It shows that we get approximately 20% feature vertices for cleanSPX dataset, 13.6% for bluntfin dataset, and 21.6% for Nucleon

	Number of Cells	Number of Vertices
CleanSPX	12,936	2,896
Blunt Fin	187,395	41,984
Nucleon	320,000	68,921
CT-head	3,951,605	819,200

Table 5.1: Details of experimental datasets

	Number of Feature Vertices Used	Total Vertices
CleanSPX	1,694	2,896
Blunt Fin	5,698	41,984
Nucleon	14,855	68,921
CT-head	140,233	819,200

Table 5.2: Number of feature vertices marked by segmentation

dataset. Table (5.3) shows the decimation rates that we can achieve for the given tolerance values and the corresponding computation times (not including the pre-processing time). It may be noted that reducing the tolerance requirements can give us higher decimation rates.

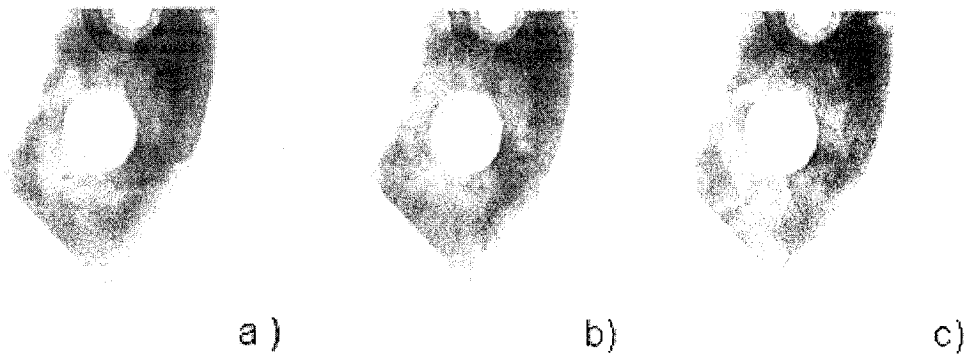

Figure 5.1: Result of Simplification. a) The original dataset. b) 45% simplification without interior feature detection. c) 45% simplification with interior feature detection

From the images shown in Figure (5.1) we can see that even with 45% decimation, our method preserves features much better, without any excessive increase in

	Decimation Rate	Tims in Secs under incremental model	Time in Secs under greedy model
CleanSPX	60%	19.609	8.587
Blunt Fin	53%	5,423.52	695.35
Nucleon	83%	8,653.69	1,056.93
CT-head	50%	N/A	14,027.5

Table 5.3: Decimation rates and computation times (without the preprocessing phase).

computation times during the rendering phase.

We compare the simplified result with the original one by calculating the value difference of two images in square root of sum of R, G and B per pixel. The degree of error between simplified and the original datasets is displayed with this color difference in Figure (5.2), with the greater amount of red in Figure (5.2-b) indicating more feature losing in the rendering of the dataset.

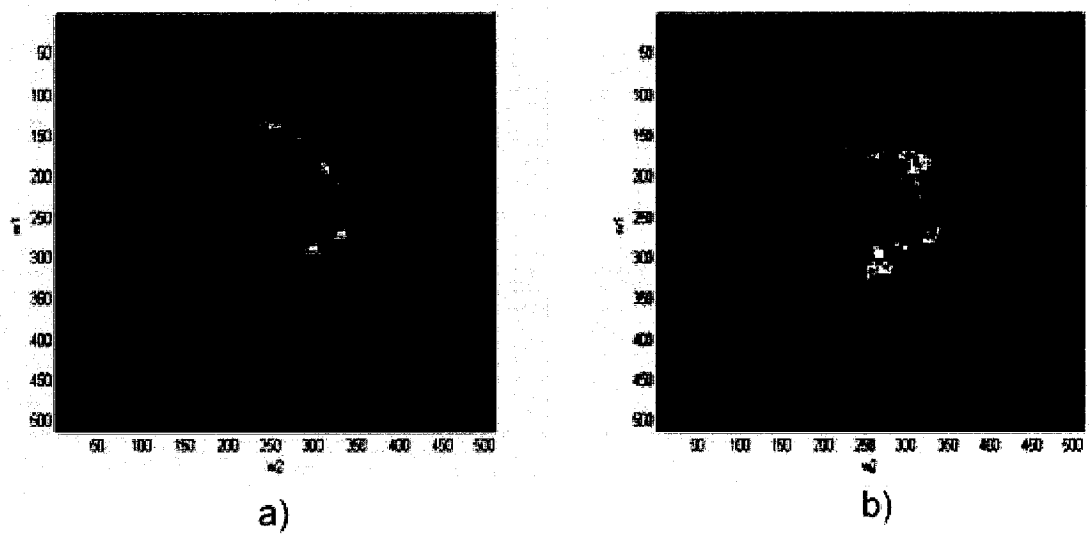

Figure 5.2: Comparison of simplification result. a) The difference for simplification result with detection of interior features, which is the image shown in Figure (5.1-b). b) The difference for simplification result without detecting interior features, which is the image showed in Figure (5.1-c). The two simplified images have the same decimation rate. w1, from 0 to 512, is the resolution in y direction, and w2, from 0 to 512, is the resolution in x direction.

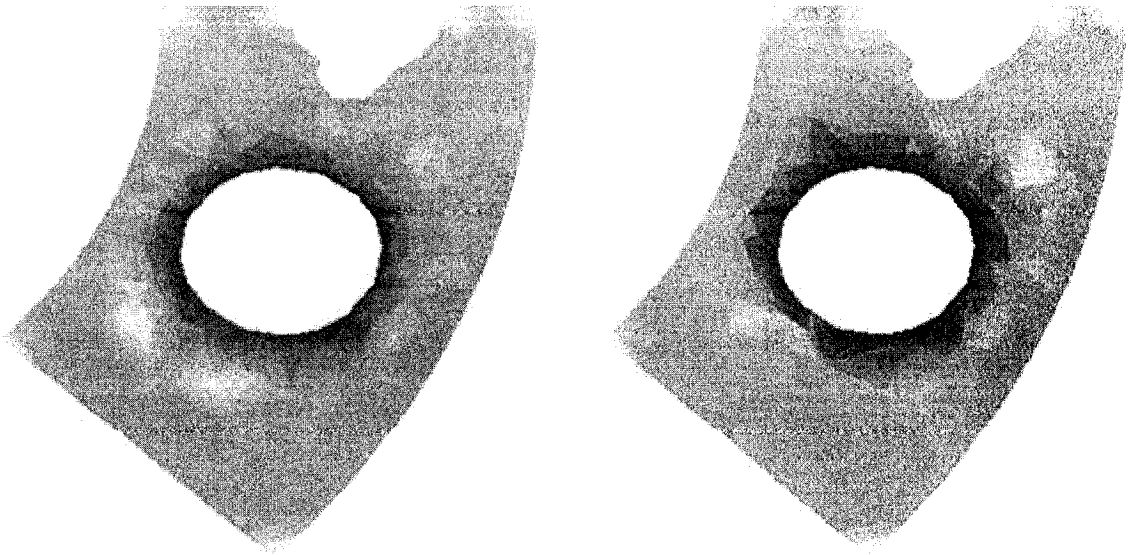

Figure 5.3: Comparison of simplification result in slice. a) One slice of original dataset; b) One slice of 45% simplified dataset

Figure(5.3) compares the original rendered slice with a rendering of a simplified (45%) slice. This clearly demonstrates that interior features are also well preserved after simplification.

The result of our segmentation process can be seen in the slice shown in Figures (5.4-(a)) and (5.4-(b)).

Figure (5.5) shows our algorithm applied to the widely used benchmark standard volume dataset, Blunt Fin. Once again, the results show that our algorithm preserves features at high decimation rates.

More implementation results showed in Figures (5.6, 5.7, 5.8, 5.9, 5.12, and 5.13)

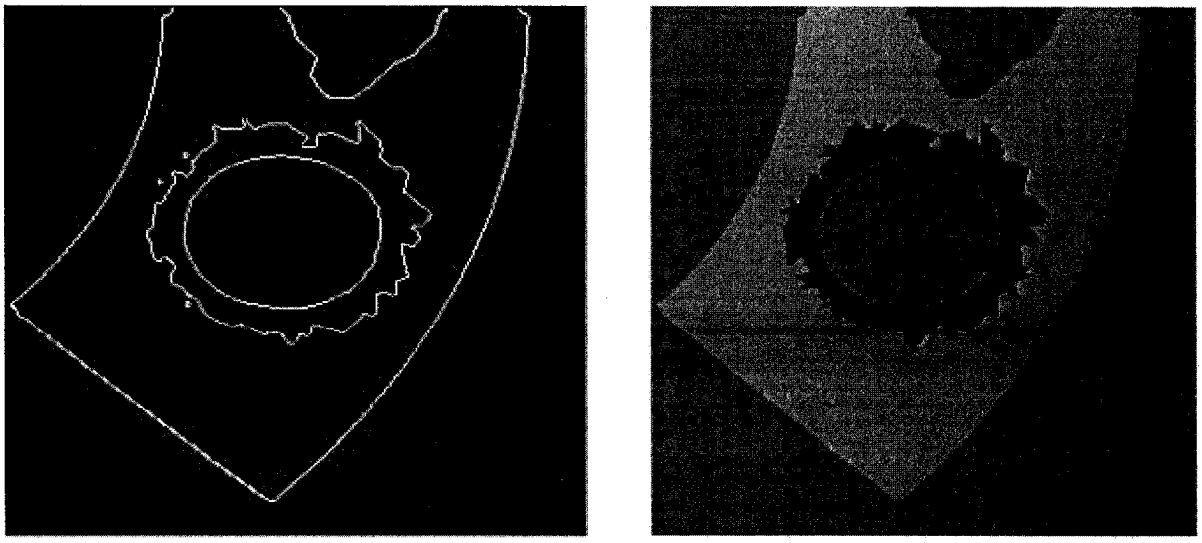

Figure 5.4: a) Detected interior boundary for SPX dataset. b) Sub-regions based on the detected interior boundary for SPX dataset (shown in slice)

5.2 Experimental Results for Higher Decimation Rates

In this section, we present our experimental results when attempting higher decimation rates using our algorithm. This is achieved by compromising on feature preservation criteria.

From figure (5.10) and (5.11), we can tell that the boundary curve has changed. More results show in Figure (5.12 and 5.13).

All results demonstrate that our algorithm can preserve enough features during simplification.

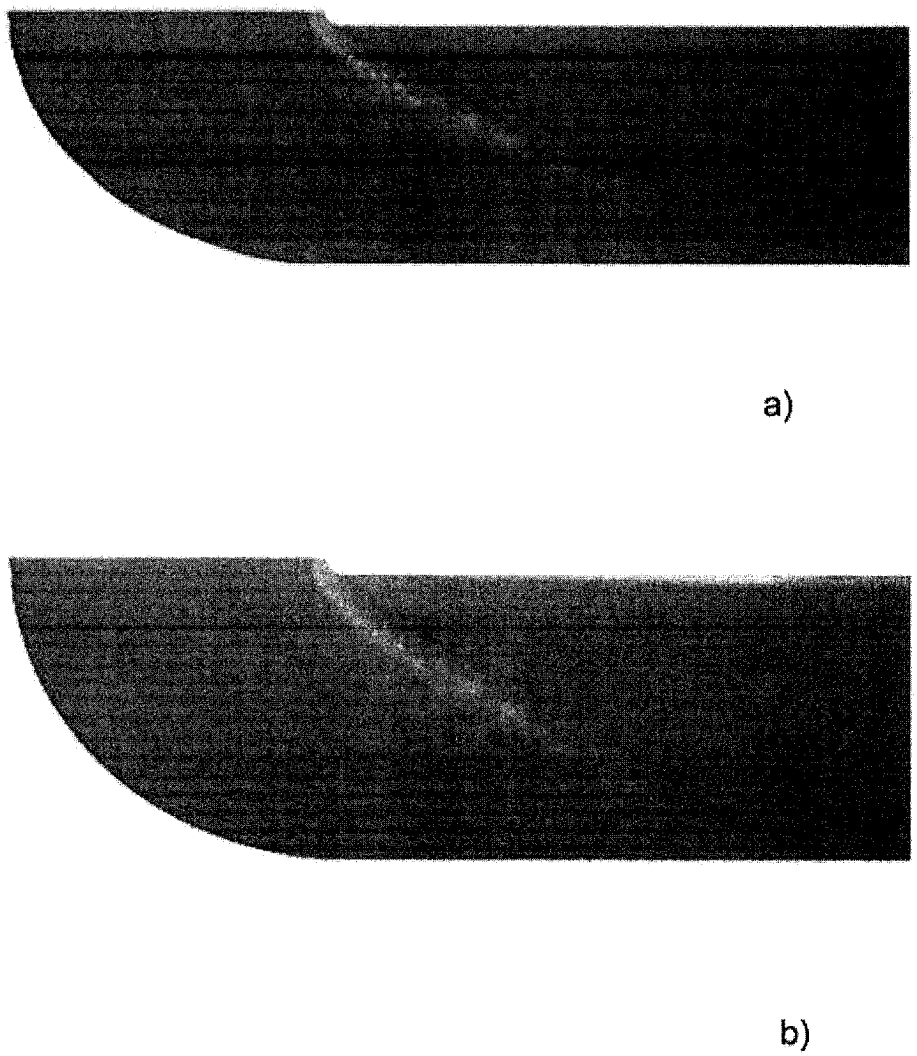

Figure 5.5: Result of simplification. a) The original dataset. b) 52% simplified dataset

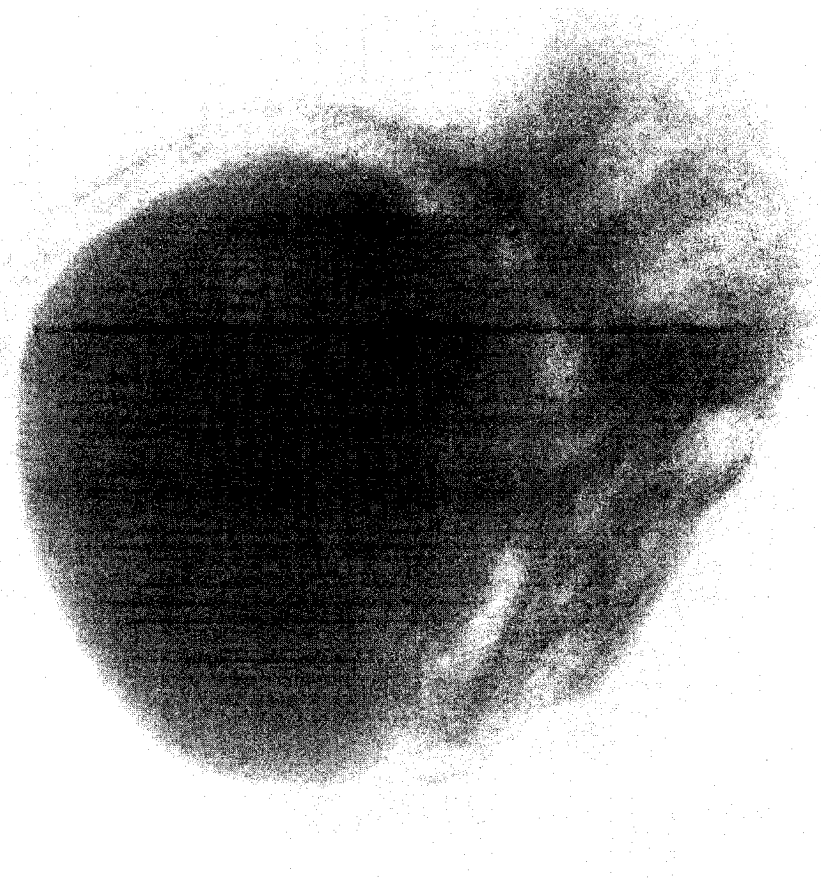

Figure 5.6: Result of simplification of CT Head, a resolution of 128x128x50, with 3,951,605 cells and 819,200 vertices. The $\Delta\varepsilon$ is 0.047937 and the decimation rate is 30%

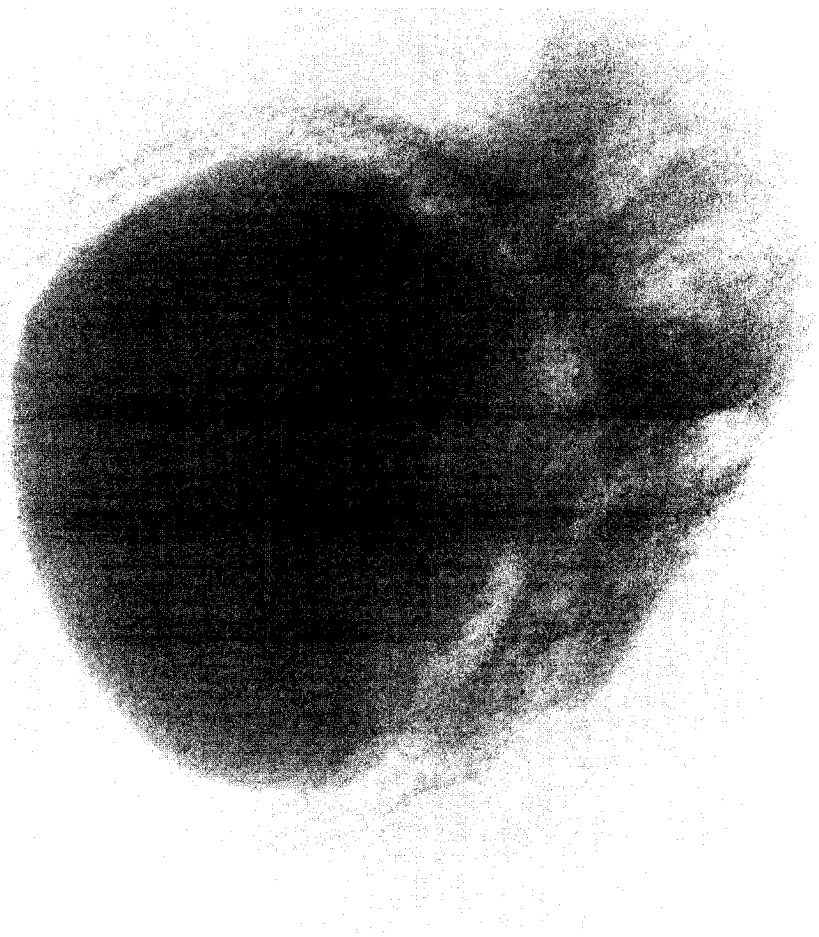

Figure 5.7: Result of simplification of CT Head, a resolution of 128x128x50, with 3,951,605 cells and 819,200 vertices. The $\Delta\varepsilon$ is 0.048926 and the decimation rate is 33%

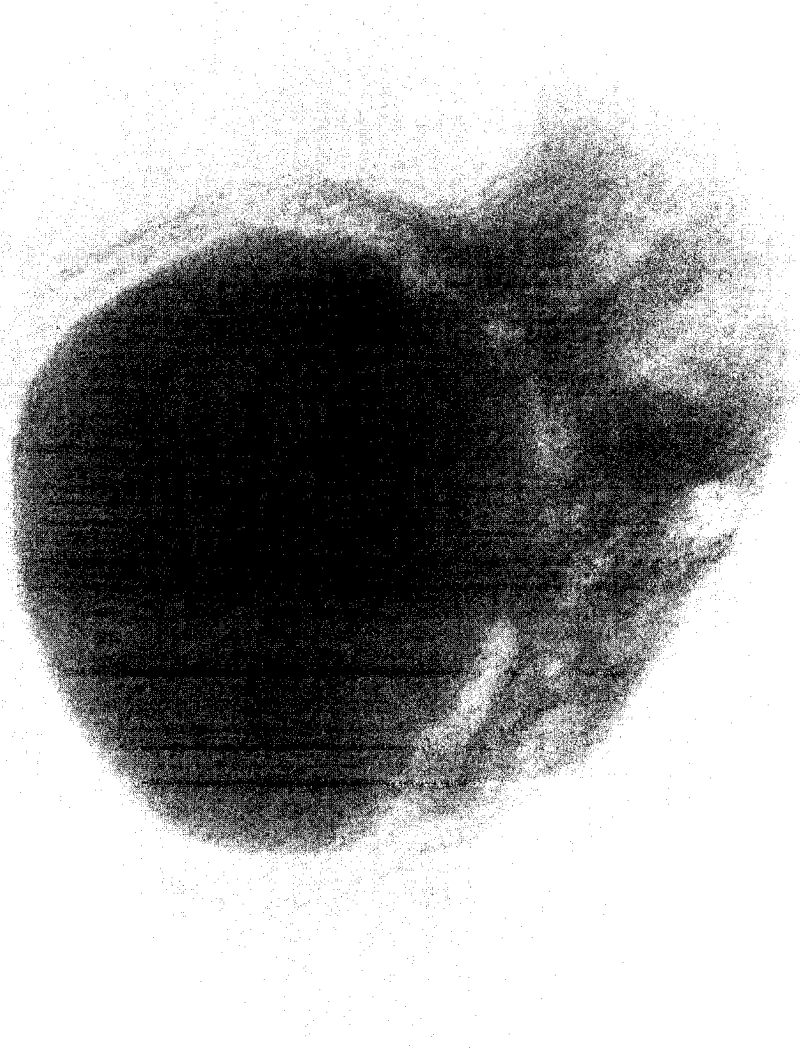

Figure 5.8: Result of simplification of CT Head, a resolution of 128x128x50, with 3,951,605 cells and 819,200 vertices. The $\Delta\varepsilon$ is 0.112453 and the decimation rate is 37%

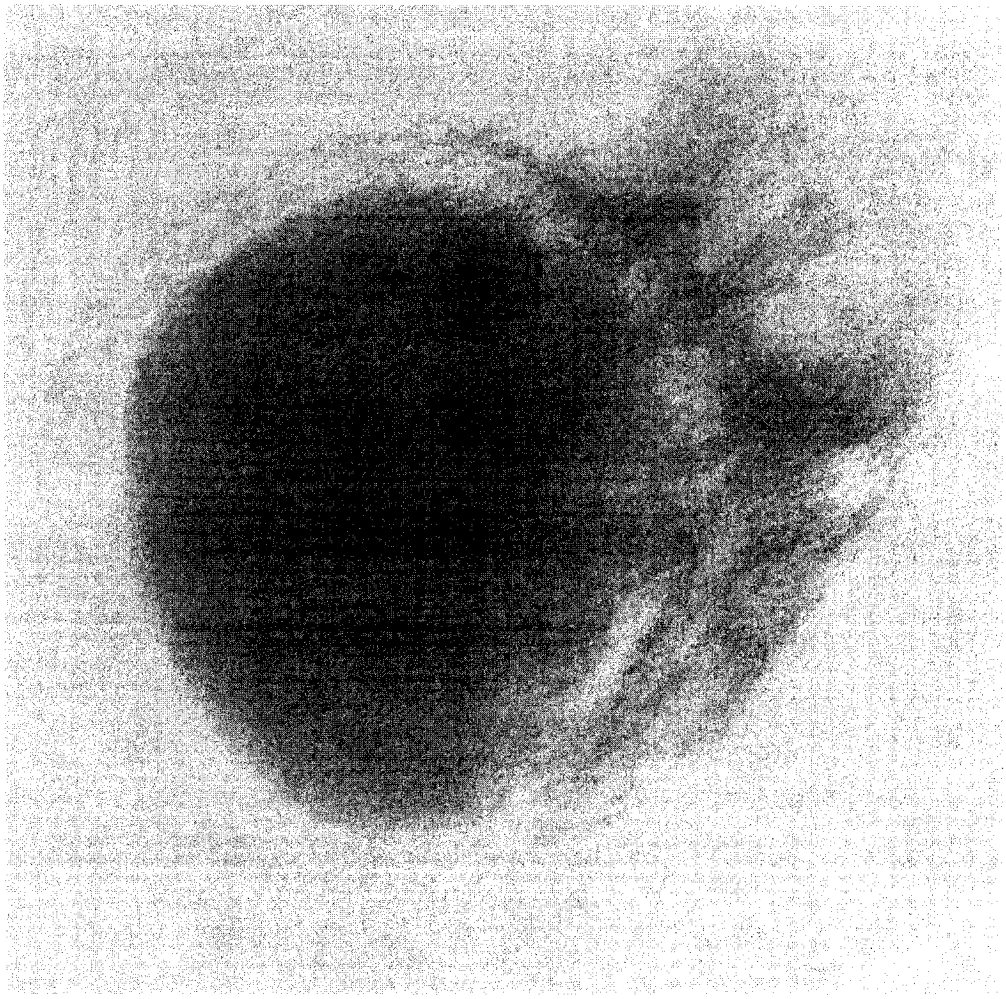

Figure 5.9: Result of simplification of CT Head dataset, a resolution of 128x128x50, with 3,951,605 cells and 819,200 vertices. The $\Delta\varepsilon$ is 7.334761 and the decimation rate is 48%

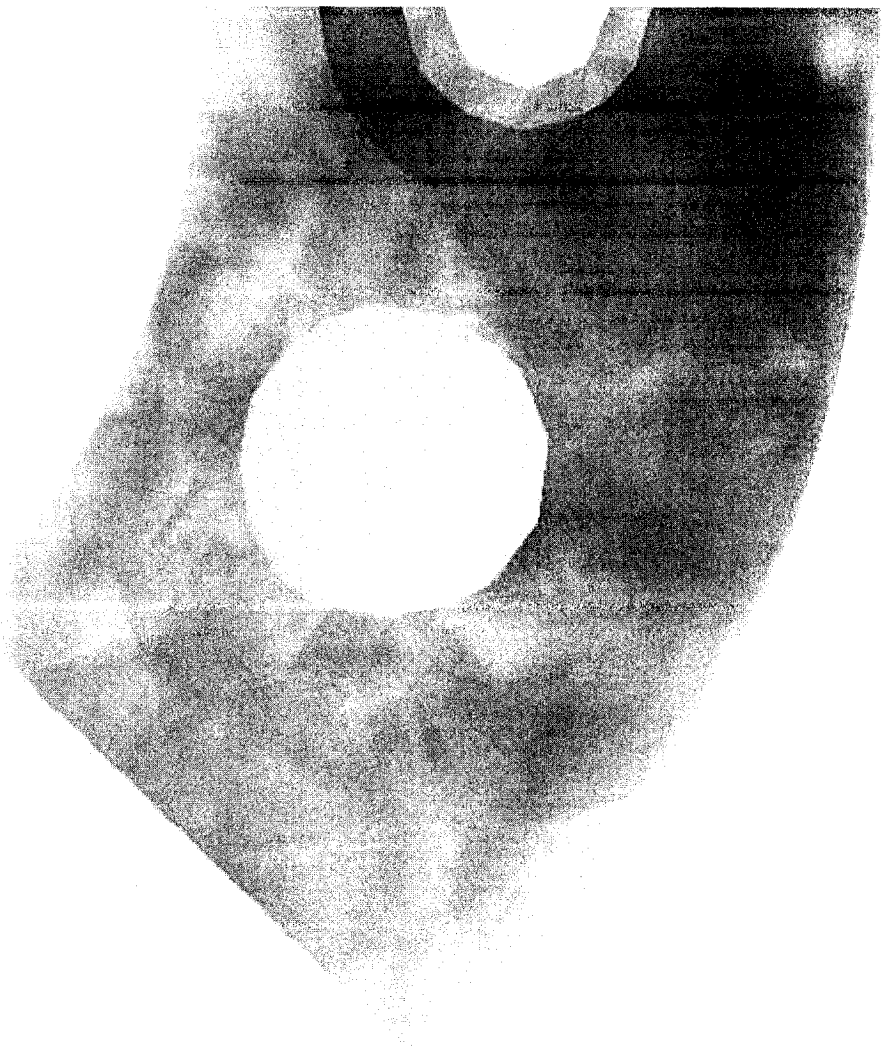

Figure 5.10: Result of simplification for SPX dataset. We achieve 61.6% decimation rates with combination of cell collapse and edge collapse.

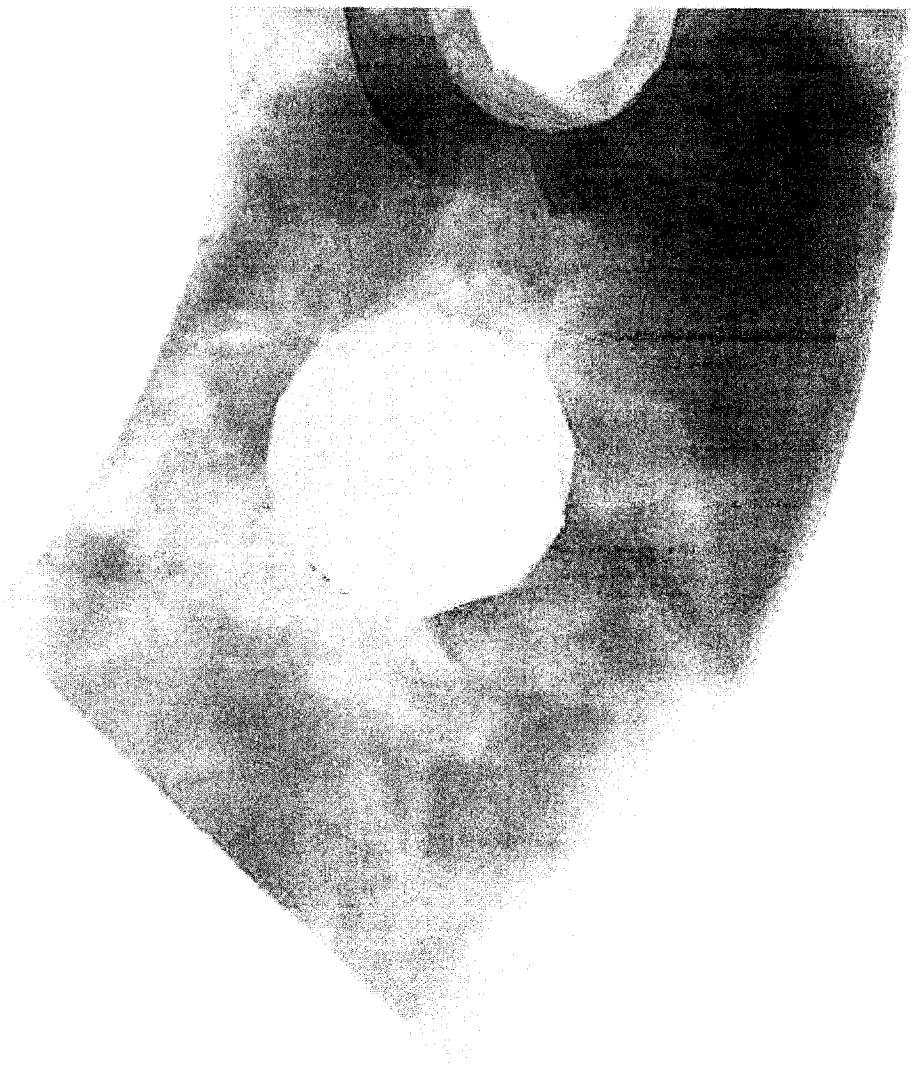

Figure 5.11: Result of simplification for SPX dataset. We achieve 68.3% decimation rates with combination of cell collapse and edge collapse.

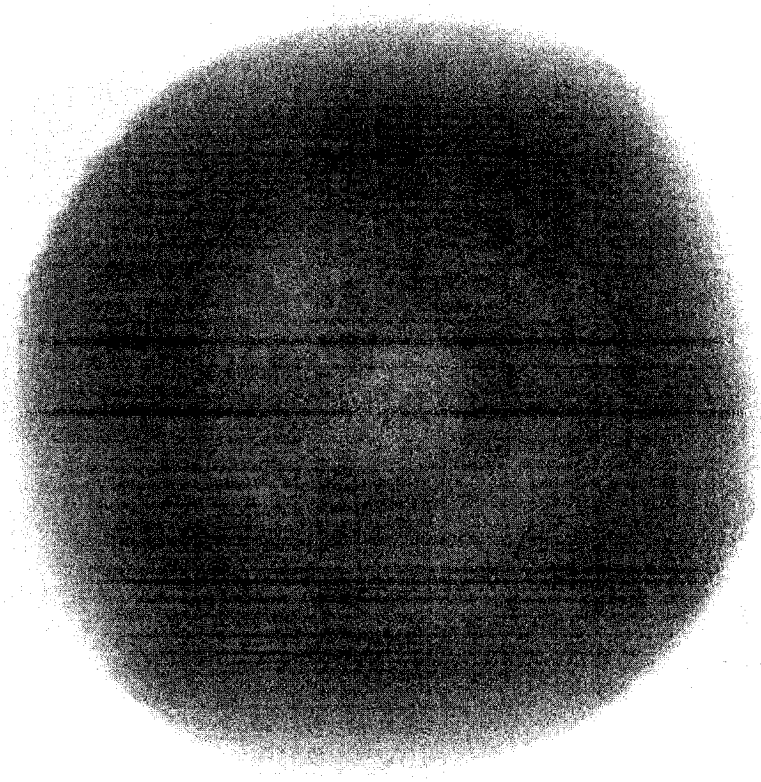

Figure 5.12: Original image of Nucleon dataset.

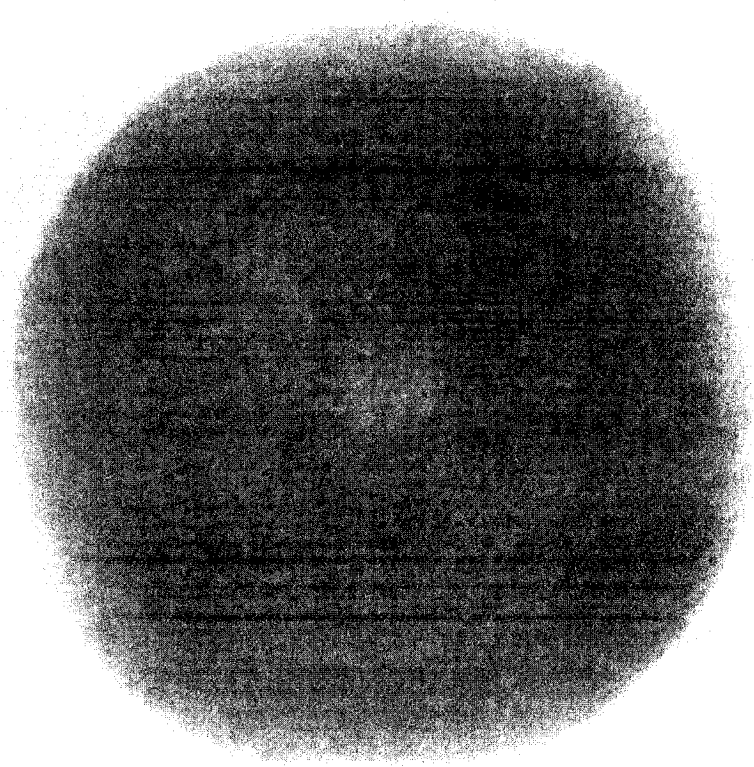

Figure 5.13: Result of simplification for Nucleon dataset. We achieve 80% decimation rates with feature vertices decimation.

Chapter 6

Conclusion and Future Work

In this thesis, we have proposed and implemented a feature preserving tetrahedral mesh simplification algorithm. Our method combines region based feature detection with traditional gradient feature detection. The algorithm has a preprocessing step for region based feature detection in which a level set segmentation on the original dataset is performed. We also avoid the tetrahedron flipping problem during simplification. The use of level set segmentation does introduce relatively more computational effort during preprocessing, but results in a number of the extremely significant benefits:

- robustness (to noise);
- preservation of complicated topology of interior regions;
- detecting of features which gradient based methods may miss.

Our results also clearly show that we preserve a richer set of features embedded in the original dataset as compared to similar algorithms, even at high decimation rates.

In table (6.1), we qualitatively compare this approach with a few of the earlier methods, namely, Chopra and Meyer's TetFusion [7], Chiang and Lu's work [25] and

	Interior Feature Preservation	Boundary Preservation	Restriction on Decimation	Error Metric(s)
Our Features method	features detected by level set segmentation & gradient	measure boundary changes with Hausdorff distance	no decimation operations should change interior boundary	Gradient, aspect ratio & regional variance
Chopra and Meyer, 2002 TetFusion	features detected by gradient method	boundary cells & their neighbor cells are untouchable	no decimation should change exterior boundary	aspect ratio
Chiang & Lu, 2003	topologically critical points & gradient based features	boundary cells are untouchable	Isosurface topologies	Gradient & edge length
Hong & Kaufman, 2003	topologically critical points & gradient based features	boundary cells are untouchable	No decimation or generation of topological critical points	Gradient

Table 6.1: Qualitative comparison of related algorithms.

Hong and Kaufman's work [6]. We consider four aspects: interior feature preservation, boundary preservation, restriction on decimation and error metrics used.

For our approach, we have not focused on reaching the maximum decimation rates. Although we could also obtain 99%, it is not very meaningful to achieve very high decimation rates without preserving features. We have worked on preserving features during the volumetric decimation. This new technique is very useful during the visualization. One shortcoming of our current approach is the computation times for the preprocessing part, which does level set segmentation. In our present implementation, the $300 \times 300 \times 100$ dataset took several hours. The level set segmentation method is not fully studied although it already shows its capability such as robustness to noise, suitable for disconnected region. With the further investigation in this area, we think we could get better results, both in feature detection and in computation

times.

We plan to perform the level set segmentation directly on tetrahedral meshes without the conversion which causes inaccurate and extra time consuming. Another possible extension of our current work is to add more criteria for error prediction when performing the exterior boundary decimation, such as sharp boundary features, described in [25]. Also, we want to create a user interactive decimation and visualization framework. It will allow users to specify the region they are interested or not. Based on users' choice, our frame work will decimate the volumetric dataset automatically, and render the results.

Bibliography

[1] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994. (Also available as Duke U. CS tech report, ftp://ftp.cs.duke.edu/dist/techreport/1994/1994-21.ps.Z).

[2] Carlos Andújar, Pere Brunet, and Dolors Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graph.*, 21(2):88–105, 2002.

[3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. In *Proc. 10th Annual ACM Symp. on Computational Geometry*, pages 293–302, 1994.

[4] Tony Chan and Luminita Vese. Active contour model without edges. *IEEE Trans. on Image Processing*, 24:266–277, 2001.

[5] Cheng Chia Chang and Ling Ling Wang. A fast multilevel thresholding method based on lowpass and highpass filtering. *Pattern Recognition Letters*, 18:1469–1478, 1997.

[6] Yi-Jen Chiang and Xiang Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. *Computer Graphics Forum*, 22(3):493–504, sep 2003.

[7] Prashant Chopra and Joerg Meyer. Tetfusion: An algorithm for rapid tetrahedral mesh simplification. In *In Proceedings Visualization '02*, pages 133–140, Oct 2002.

[8] P. Cignoni, C. Costanza, C. Montani, C. Rocchin, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings of the 11th IEEE Visualization 2000 Conference (VIS 2000)*. IEEE Computer Society, 2000.

[9] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[10] Charles A. Csuri, James Blinn, Julian Gomez, Nelson Max, and William Reeves. The simulation of natural phenomena (panel session). In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 137–139. ACM Press, 1983.

[11] G. Das and M. T. Goodrich. On the complexity of approximating and illuminating three-dimensional convex polyhedra. In S. G. Akl, F. Dehne, J.-R. Sack, and N. Santoro, editors, *Algorithms and Data Structures*, pages 74–85. Springer, Berlin,, 1995.

[12] H. Delingette, M. Hebert, and K. Ikeuchi. Shape representation and image segmentation using deformable surfaces. In *Conf. on Computer Vision and Pattern Recognition (CVPR '91)*, pages 467–472, 1991.

[13] Hervé Delingette, Martial Hebert, and Katsushi Ikeuchi. Shape representation and image segmentation using deformable surfaces. *Image and Vision Computing*, 10(3):132–144, Apr. 1992.

[14] Ricardo Farias, Joseph S. B. Mitchell, and Cláudio T. Silva. Zsweep: an efficient and exact projection algorithm for unstructured volume rendering. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 91–99. ACM Press, 2000.

[15] O. D. Faugeraus and M. Hebert. Polyhedral approximation of 3-d objects without holes. In *IEEE 1982 Conference on Pattern Recognition and Image Processing*, pages 593–598, 1982.

[16] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[17] Allen Van Gelder, Vivek Verma, and Jane Wilhelms. Volume decimation of irregular tetrahedral grids. In *Computer Graphics International*, pages 222–230, 1999.

[18] Thomas Gerstner and Renato Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In *Proceedings of the*

conference on Visualization '00, pages 259–266. IEEE Computer Society Press, 2000.

[19] Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Greg Schussman, and Isaac J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, 4 1998.

[20] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. Simplification and compression of 3d meshes. In *Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling (PRIMUS)*, pages 319–361, 2001.

[21] Bernd Hamann. A data reduction scheme for triangulated surfaces. *Comput. Aided Geom. Des.*, 11(2):197–214, 1994.

[22] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. Technical report, to appear.

[23] G. T. Herman and H. K. Liu. Dynamic boundary surface detection. *Computer Graphics and Image Processing*, 7:130–138, 1978.

[24] Michal Holtzman-Gazit, Dorith Goldsher, and Ron Kimmel. Hierarchical segmentation of thin structure in volumetric medical images. In *MICCAI*, Montreal, 2003.

[25] Wei Hong and Arie Kaufman. Feature preserved volume simplification. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 334–339. ACM Press, 2003.

[26] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[27] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78, July 1992.

[28] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM Press, 1993.

[29] Ian H. Jermyn and Hiroshi Ishikawa. Globally optimal regions and boundaries. In *Proceedings of the International Conference on Computer Vision-Volume 2*, page 904. IEEE Computer Society, 1999.

[30] T. Jiang, M. B. Merickel, and E. A. Parrish. Automated threshold detection using a pyramid data structure. *In 9th International Conference on Pattern Recognition*, 2:689–692, 1988.

[31] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86. ACM Press, 1998.

[32] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In Roni Yagel and Gregory M. Nielson., editors, *IEEE Visualization '96*, pages 311–318, 1996.

[33] Mike Krus, Patrick Bourdot, Franoise Guisnel, and Guillaume Thibault. Levels of detail & polygonal simplification. *ACM Crossroads Electronic Publication*, 3(4), 1997.

[34] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[35] H. K. Liu. Two and three dimensional boundary detection. In *Computer Graphics and Image Processing*, volume 6, pages 123–134, 1977.

[36] David Luebke. A survey of polygonal simplification algorithms. Technical report, University of North Carolina at Chapel Hill, 1997.

[37] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2002.

[38] David P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Comput. Graph. Appl.*, 21(3):24–35, 2001.

[39] Edmond Nadler. Piecewise linear best L_2 approximation on triangulations. In C. K. Chui et al., editors, *Approximation Theory V*, pages 499–502, Boston, 1986. Academic Press.

[40] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[41] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000.

[42] David M. Reed, Roni Yagel, Asish Law, Po-Wen Shin, and Naeem Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. In *Proceedings of the 1996 symposium on Volume visualization*, pages 55–ff. IEEE Press, 1996.

[43] William T. Reeves. Particle systems a technique for modeling a class of fuzzy objects. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 359–375. ACM Press, 1983.

[44] J. Rossignac and P. Borrel. Multiresolution 3d aproximations for rendering complex scenes. 1993.

[45] Günter Rote. Computing the minimum hausdorff distance between two point sets on a line under translation. *Inf. Process. Lett.*, 38(3):123–127, 1991.

[46] P. K. Sahoo, S. Soltani, A. K.C. Wong, and Y. C. Chen. A survey of thresholding techniques. *Comput. Vision Graph. Image Process.*, 41(2):233–260, 1988.

[47] Christophe Samson, Laure Blanc-Feraud, Gilles Aubert, and Josiane Zerubia. A level set model for image classification. In *Scale-Space Theories in Computer Vision*, pages 306–317, 1999.

[48] Cláudio Silva, Joseph S. B. Mitchell, and Arie E. Kaufman. Fast rendering of irregular grids. In *Proceedings of the 1996 symposium on Volume visualization*, pages 15–ff. IEEE Press, 1996.

[49] Marc Soucy and Denis Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision Image Understanding*, 63(1):1–14, 1996.

[50] Oliver G.and Staadt and Markus H. Gross. Progressive tetrahedralizations. In *I-vis*, pages 397–402, 1998.

[51] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 185–194. ACM Press, 1992.

[52] Issac J. Trotts, Bernd Hamann, and Kenneth I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.

[53] Issac J. Trotts, Bernd Hamann, Kenneth I. Joy, and David F. Wiley. Simplification of tetrahedral meshes. In *Proceedings of the conference on Visualization '98*, pages 287–295. IEEE Computer Society Press, 1998.

[54] A. Tsai, A. Yezzi, and A. S. Willsky. Curve evolution implementation of the mumford-shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Trans. Image Processing*, 10(8):1169 –1186, 2001.

[55] Wen-Hsiang Tsai. Moment-preserving thresholding: a new approach. *Document image analysis*, pages 44–60, 1995.

[56] Jarke J. van Wijk. Flow visualization with surface particles. *IEEE Comput. Graph. Appl.*, 13(4):18–24, 1993.

[57] Luminita Vese and Tony Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *IJCV*, 50(3):271–293, 2002.

[58] Gunther H. Weber, Gerik Scheuermann, Hans Hagen, and Bernd Hamann. Exploring scalar fields using critical isovalues. In *Proceedings of the conference on Visualization '02*, pages 171–178. IEEE Computer Society, 2002.

[59] Craig M. Wittenbrink. Cellfast: Interactive unstructured volume rendering. Technical report, Hewlett-Packard Labs, 1501 Page Mill Road, Palo Alto, CA 94304, 1999.

[60] Z. Wood, H. Hoppe, M. Desbrun, and P. Schrder. Isosurface topology simplification manuscript. Technical report, Computer Science Dept., California Polytechnic State University, San Luis Obispo, California, USA, 2002.

[61] Meihe Xu, P.M. Thompson, and A.W. Toga. An adaptive level set segmentation on a triangulated mesh. *IEEE Transactions on Medical Imaging*, 23(2):191–201, 2004.

[62] J. Yao and R. Taylor. Tetrahedral mesh modeling of density data for anatomical atlases and intensity-based registration. In *International Conference on Medical Image Computing and Computer-Assisted Intervention MICCAI'2000*, Lecture Notes in Computer Science, pages 531–540, Cambridge, England, 2000. Springer Verlag.

[63] Hong-Kai Zhao, Tony. F. Chan, B. Merriman, and Stanley Osher. A variational level set approach to multiphase motion. *J. Comput. Phys.*, 127(1):179–195, 1996.

[64] S. W. Zucker and R. A. Hummel. Optimal three-dimensional edge operator. Technical report, McGill Unversity, Montreal, Quebec, Canada, 1978.

[65] S. W. Zucker and R. A. Hummel. A three–dimensional edge detector. *PAMI*, 3(3):324–331, may 1981.