uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Lijun Chen

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Efficient 3D Multi-Resolution Modeling, Segmentation, and Segmentation-Based Mesh Compression

TITRE DE LA THÈSE / TITLE OF THESIS

Nicolas Georganas

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Richard Egli

Robert Laganière

Jochen Lang

Michiel Smid

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

EFFICIENT 3D MULTI-RESOLUTION MODELING, SEGMENTATION, AND SEGMENTATION-BASED MESH COMPRESSION

By

Lijun Chen

A Dissertation Submitted to the
Faculty of Graduate and Postdoctoral Studies
University of Ottawa
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa

September 2005

Your file Votre référence
ISBN: 0-494-10956-4
Our file Notre référence
ISBN: 0-494-10956-4

ACKNOWLEDGEMENTS

First and foremost, I am deeply indebted to my supervisor, Dr. Nicolas D. Georganas, for his advice and guidance throughout my graduate program. He has created a great work environment. The advanced computing facilities, weekly meetings and one-to-one academic discussions with Dr. Georganas have allowed me to develop my researching skills. He has always been supportive and encouraging to me when I faced difficulties not only at work, but also in my personal life. Without his support, I would not have the academic achievement today.

The fellow members of Dr. Georganas's group have helped me with useful and relaxing conversions over years. Special thanks go to Dr. Xiaojun Shen, Dr. Huiping Guo, Lei Guo, and François Malric.

My family has been a constant source of support and encouragement. I sincerely appreciate my parents for their love and support during my lengthy education. I thank my sisters for their caring of my parents and their encouragement for my graduate study.

Finally, I must express my gratitude to my wife, Hongchao Wang. Her encouragement and affection have always been giving me the strength. My eighteen-month old son, Alexander Chen, is the greatest gift I have ever received. His birth has brought me endless happiness.

ABSTRACT

3D polygonal models are increasingly being deployed in a wide range of Virtual Reality (VR) applications, including scientific visualization, collaborative design, entertainment, e-commerce, and remote education, etc. These models, produced by the 3D laser scanning systems, are often represented as complex polyhedral meshes with hundreds of thousands or millions of vertices and triangles. Although this representation can achieve high level of realism, these models usually demand a huge amount of storage space and/or transmission bandwidth in the raw data format. Also, a polygonal mesh does not capture a high-level structure, which is useful for managing data in applications, such as object registration, object retrieval and indexing, feature modeling, etc. One way to impose such a high-level description is through mesh segmentation. Therefore, mesh simplification, segmentation and compression have recently been the main areas in 3D mesh processing.

In this dissertation, we present an efficient 3D mesh multi-resolution modeling algorithm, which can output arbitrary resolutions of an input model. This algorithm considers edge curvature and neighborhood face area change as the error metrics for the edge collapse operation. Compared with most of the existing simplification algorithms, the proposed method is simple and effective. We also present an efficient and robust neighborhood based algorithm for 3D mesh segmentation. This approach uses discrete Gaussian curvature and concaveness estimation to detect the boundary vertices between

the distinct regions of a given mesh. To capture more accurate and relevant geometric information of a vertex on the mesh surface, we enlarge the 1-ring neighborhood to an eXtended Multiple-Ring (XMR) neighborhood. After feature detection, a fast marching watershed method is deployed, followed by an efficient region merging scheme. Simulation results show that this algorithm is efficient and robust to high-resolution models. Finally, we propose a segmentation based mesh compression scheme. Most of the existing 3D mesh coding algorithms compress the model as a whole. Our algorithm separately compresses the partitioned regions that are obtained by the segmentation method described above. The compressed data are put into one stream part by part. Each part is independent of the others. The boundary strips between the different regions are also encoded and appended to the end of the stream. In an inactive or selective application, the users can get the interested parts or the whole object after all the parts and the boundary strips that can connect all the parts together have been received.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1 INTRODUCTION

1.1 Research Motivation

3D computer graphic models became increasingly popular since the advent of the 3D laser scanning system and the boom of Virtual Reality (VR) applications. Highly detailed models are routinely acquired and used in design, manufacturing, and entertainment. These models are often represented as complex polyhedral meshes, which may have thousands or even millions of vertices and triangles.

Generally speaking, these models are expensive to render, awkward to edit and costly to transmit through the network. Wider applications of 3D graphics could be potentially limited due to these obstacles. Thus, in some real-time rendering applications, it is essential to reduce the resolution of the highly detailed models by deleting some vertices and polygons and keeping the original geometric shape efficiently. Since different users or applications may have different resolution requirements, multi-resolution based simplification is an active area in 3D object modeling. Also, in a Collaborative Virtual Environment (CVE), users may want the original models transferred from the server. An effective way to reduce the local storage amount and the network bandwidth requirements is to compress the models. Most of the existing methods compress and transmit the 3D object as a whole. In some interactive or selective

applications, such as Internet gaming, collaborative CAD design, or remote learning, the client may be interested in particular section(s) of the object. Therefore, before transmission, the server needs to segment the object into parts and send them individually or sequentially. Segmentation itself is also one of the main areas of 3D mesh modeling. It is useful for managing data in applications, such as object registration, object retrieval and indexing, feature modeling, etc.

The motivation of this research is to build a framework to produce different outputs depending on the application requirements in a CVE environment, where the existing literature is either lacking or less efficient. The framework is shown in Figure 1.1.

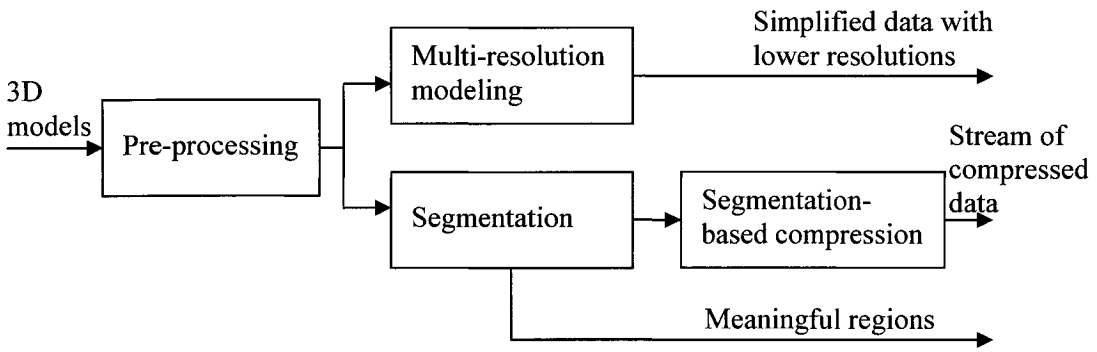


Figure 1.1 A framework for 3D object modeling.

1.2 Context of Prior Work

Various techniques aimed at reducing the 3D object complexity have been published [SZL92, RoB93, Lou94, Tur92]. Recently, the research in the 3D object simplification area has converged to 3D object multi-resolution modeling [Hop96, GH97, LT98, Lin98]. A multi-resolution model is one that captures a wide range of levels of detail (LOD) of an object and that can be used to reconstruct any one of those levels on demand. Among the existing algorithms, Progressive Mesh [Hop96] is the one with some satisfactory features,

such as smooth transition, preserving topology and natural suitability for progressive transmission. The drawback of this method is that it is very time-consuming because of its extremely high computational cost, e.g. for a large object it may take hours to simplify it. Thus, a fast and efficient scheme is necessary. Many researchers have worked in this direction to find a simple solution. The Quadric-based algorithm proposed by Garland [GH97] and the polygon reduction algorithm presented by Melax [Mel98] are efficient and simple. They only take seconds for the same model as in [Hop96]. But the quality of this method is not acceptable in some cases. Therefore, a more efficient and effective algorithm is still needed, as the real-time or interactive requirements are increasingly demanding in the applications of Virtual Environments.

Segmentation means breaking down an existing structure into meaningful connected sub-components. It is more common in the image processing area, and has been recently introduced into the 3D mesh area [EDD+95, WL97, MW99, RM01, LPR+02]. Among these algorithms, the watershed-based one becomes of more interest. This algorithm, described in [BL79], is a classic one in 2D image segmentation. Mangan and Whitaker [MW99] generalize the watershed method to arbitrary meshes by using the discrete curvature at each mesh vertex as the height field, analogous to the pixel intensity on an image grid that drives the 2D watershed segmentation. The curvature estimation for 3D meshes is not a trivial operation, as it is mathematically defined for a smooth surface only [MP77]. Most of the existing algorithms are computationally expensive. Zhang [ZPK+02] proposed a simple segmentation algorithm using Gaussian curvature analysis. It is efficient for certain 3D models. However, similarly to most of the existing methods, this algorithm has a drawback that it cannot successfully process a high resolution 3D model,

as the geometric characteristics of the polygons in such a model, such as normals and curvatures are so close that the algorithm fails to detect some feature parts. The goal of this dissertation is to design an efficient and robust algorithm for 3D mesh segmentation.

Many 3D mesh coding algorithms, including lossy and lossless, single resolution and multi-resolution ones, have been proposed [Hop98, TGHL98, PR00, LK97, KSS00]. Among them, the Rossignac's Edgebreaker [Ros99] method is an efficient and simple one. It can handle manifold meshes with multiple boundary loops and handles. There has been a significant progress in 3D mesh coding performance; however, most of the existing methods compress and transmit the 3D object as a whole. In some interactive or selective applications, such as Internet gaming, collaborative CAD design, or remote learning, the client may be interested in particular section(s) of the object. Therefore, before transmission, the server needs to segment the object into parts and send them individually or sequentially.

1.3 Contributions

The primary contributions of this dissertation are summarized as follows.

- An efficient Progressive Mesh based 3D multi-resolution modeling algorithm is presented [CG02]. In the Progressive Mesh algorithm, the key stage is to find the proper edge to collapse at each step. In most existing schemes, the error metrics for edge collapse are very complicated. In order to explore the geometric importance of a 3D-object, we define some simple and efficient criteria that involve edge curvature and neighbourhood face area change. Those factors

combined together can efficiently simplify the original model and effectively preserve its topology during the simplification procedure.

- An efficient and robust neighbourhood based 3D mesh segmentation algorithm is presented [CG04]. We use Gaussian curvature and concaveness estimation to detect important features of a 3D model in order to divide it into meaningful parts. And to make the algorithm robust to high-resolution meshes, we enlarge the normal 1-ring neighbourhood to an eXtended Multi-Ring (XMR) neighbourhood during the estimation for each vertex. After feature extraction, we develop a watershed-based segmentation algorithm with an efficient region merging scheme.
- A segmentation-based 3D mesh compression scheme [CG05] is proposed. After mesh segmentation, a 3D object is divided into multiple regions and the boundary triangle strips that connect those regions. At the encoder side, we compress each region individually, using the Edgebreaker algorithm, and put them into one stream. The boundary strips are encoded and appended to the end of the stream. The decoder can then get each individual region and restore the whole object using the boundary strips.

1.4 Document Organization

The remainder of this dissertation is organized as follows. A thorough background of the related research is presented in Chapter 2. The background information includes the literature review in 3D mesh simplification, segmentation and compression. An efficient 3D mesh multi-resolution modeling algorithm is proposed in Chapter 3. Chapter 4 introduces an XMR neighbourhood based 3D watershed segmentation method that can

effectively handle densely distributed objects. A simple yet efficient region merging algorithm is also proposed. Following the efficient and robust segmentation, a segmentation-based 3D compression scheme is described in Chapter 5. Experimental results and comparisons are given in Chapter 3, 4 and 5. Finally, Chapter 6 summarizes the proposed algorithms and points out the future research.

1.5 Publications Arising from This Dissertation

[CG02] L. Chen, and N. D. Georganas, "An Efficient Multi-resolution Modeling for 3D Objects in Applications of Virtual Environment", HAVE2002, pp. 49-54, Ottawa, Nov. 2002

[CG04] L. Chen, and N. D. Georganas, "An Efficient and Robust Algorithm for 3D Mesh Segmentation", Journal of Multimedia Tools, and Applications (to appear).

[CG05] L. Chen, and N. D. Georganas, "3D Mesh Compression Using an Efficient Neighbourhood-based Segmentation" Proc 9th IEEE International Symposium on Distributed Simulation and Real-time Applications (DS-RT 2005), Montreal, Oct. 2005.

CHAPTER 2

2 BACKGROUND AND RELATED WORK

In this chapter, we will describe the 3D mesh processing preliminaries, including some concepts in the domain of geometric representations and related data structures, and the surface approximation evaluation. Then, in the following sections, a literature overview of 3D mesh simplification, segmentation and compression will be given.

2.1 Preliminaries in 3D Mesh Processing

2.1.1 Topological Concepts and Mesh Representation

Definition 2.1: A topological space is a set X with a choice of family of subsets of X (its open sets), each of which is called a neighbourhood of its points, such that every point of X is in some neighbourhood and that the intersection of any two neighbourhoods of a point contains a neighbourhood of that point.

In this dissertation, we will focus on a special kind of topological space, the 3D Euclidean space R^3.

Definition 2.2: Given two points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ in R^n, the Euclidean distance between x and y is defined by

$$\|x - y\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \qquad (2.1)$$

The disc $D^n(x, r)$, centred at x with radius r, is defined as

$$D^n(x,r) = \{y \in R^n : \|x - y\| < r\} \tag{2.2}$$

Definition 2.3: Let a set $A \subseteq R^n$. A neighbourhood of a point $x \in A$ relative to A is a set of the form $D^n(x, r) \cap A$.

Definition 2.4: A simplicial complex K is a set of vertices {0, 1, ..., m} together with a collection of finite non-empty subsets S of the vertices (each non-empty subset is called a simplex), such that any set consisting of exactly one vertex is a simplex in K, and every non-empty subset of a simplex is also a simplex in K.

A simplex with n+1 non-empty elements is called n-dimensional simplex or n-simplex. The dimension of an n-simplex is defined as n. The dimension of simplicial complex is defined as the maximum dimension of all its simplices. A 0-simplex called a vertex; a 1-simplex is called an edge; and a 2-simplex is called a face or a triangle.

Definition 2.5: A triangular mesh M is a geometric realization $|K|_v$ of K in R^3, where K is a 2-dimensional simplicial complex and V is the vector set $\{v_0, v_1, ..., v_m\}$, $v_i \in R^3$, such that

- Each 0-simplex is a subset of some 1-simplex in K.
- Each 1-simplex is a subset of some 2-simplex in K.

A triangular mesh is also written as M= (K, V) [HDD+93]. An edge of the triangular mesh is defined by a pair of vertices $\{i, j\}$, and a face or triangle by a triple of vertices $\{i, j, k\}$.

In real applications, a triangular mesh may have either photometry data (color, normal, and texture coordinates) or general function values which stand for physical phenomena such as temperature and velocity. We will call such data as attached *attributes* or *property*. A triangular mesh with attributes is written as $\boldsymbol{M} = (\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{P}_1, \boldsymbol{P}_2, \ldots)$ with $\boldsymbol{P}_i$ ($i = 1, 2, \ldots$) being attached attributes. Attributes are usually bound to vertices and faces in the following three modes: per vertex binding, per face binding, and per corner binding. Let $\boldsymbol{P} = \{p_l\}$ be a set of attribute values. Obviously, p_l is a 3D vector if the attribute is either normal or color; it is a 2D vector if the attribute is texture coordinates; or it is a scalar if the attribute is temperature. For the per vertex binding, an attribute value p_i is associated with each vertex $\{i\}$. For the per face binding, a single attribute value p_{ijk} is associated with each face $\{i, j, k\}$. For the per corner binding, a triple values (p^i_{ijk}, p^j_{ijk}, p^k_{ijk}) are associated with each face $\{i, j, k\}$.

Definition 2.6: A face of a triangular mesh is simple, if all its three indices are different.

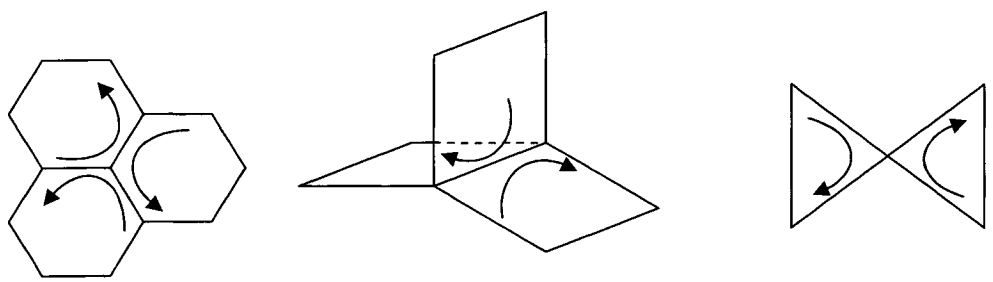
Definition 2.7: An edge of a triangular mesh is a boundary, if it belongs to exactly one face; it is internal if it is shared by exactly two faces; and it is singular if it is shared by three or more faces.

Internal edges and boundary edges are regular.

Definition 2.8: A vertex of a triangular mesh is regular, if the set of vertices of all its incident faces, except for itself, can form a single path; otherwise, it is singular.

Definition 2.9: A triangular mesh without any singular vertex or edge is called a manifold mesh; otherwise, it is called a non-manifold mesh.

Definition 2.10: The orientation of a polygon can be specified by the ordering of its bounding vertices. The orientations of two adjacent polygons are called compatible if they impose opposite directions on their common edges. A 3D mesh is said to be orientable, if there exists an arrangement of polygon orientations such that each pair of adjacent polygons are compatible.



(a) (b) (c)

Figure 2.1 Examples of (a) an orientale and manifold mesh, (b) a non-orientable and non-manifold mesh, (c) an orientable and non-manifold mesh.

Figure 2.1 illustrates polygon orientation and manifoldness. (a) is an orientable and manifold mesh; (b) is a non-orientable and non-manifold mesh; and (c) is an orientable and non-manifold mesh.

In the following chapters in this dissertation, we will focus on manifold 3D triangular meshes.

2.1.2 Error Measurement of Surface Approximation

In order to guide the 3D mesh processing, such as simplification and compression, a proper error measurement must be defined. Here we introduce two most well known error measurements, i.e., the Hausdorff distance and average squared distance.

Unlike measuring deviation in 2D domain, for 3D general surfaces we need to measure all distances between two sets of points. The distance from a point v to the model M is defined to be the distance to the closest point w on the model:

$$d_v(M) = \min_{w \in M} \| v - w \| \tag{2.6}$$

where $\| \bullet \|$ is the usual Euclidean vector length operator which is defined in (2.1).

One commonly used geometric error measurement is the Hausdorff distance, an analog of the L_∞ metric, which can be defined as

$$E_{\max}(M_1, M_2) = \max(\max_{v \in M_1} d_v(M_2), \max_{v \in M_2} d_v(M_1)) \tag{2.7}$$

The Hausdorff error measures the maximum deviation between the two models. If $E_{max}(M, M') < \varepsilon$, then we know that every point of the approximation is within ε of the original surface and that every point of the original is within ε of the approximation. Similarly, we can define E_{avg}, an analog of the L_2 metric, which measures the average squared distance between the two models as

$$E_{avg}(M_1, M_2) = \frac{1}{w_1} \int_{v \in M_1} d_v^2(M_2) + \frac{1}{w_2} \int_{v \in M_2} d_v^2(M_1) \tag{2.8}$$

where w_1, w_2 are the surface areas of M_1, M_2.

Note the symmetric construction of both E_{max} and E_{avg}. It is not sufficient to simply consider every point on M_1 and find the closest corresponding point on M_2. We must also

do the same for every point on M_2. The computation of these metrics can be prohibitively expensive for large polyhedral models because implementing algorithms are of $O(n^2)$ time complexity, where n is the number of vertices. In practice, some algorithms directly used these metrics for surface approximation; and many others developed simpler measurements to reduce the cost.

2.2 3D Mesh Simplification Algorithms

The problems of polygonal simplification and multi-resolution modeling have received increasing attention in recent years. The underlying concept of multi-resolution surface models is not particularly new; Clark [Cla76] discussed the general idea twenty-nine years ago. However, with the exception of work done on simpler objects such as curves and height fields, most of the results in the field are fairly recent. In this section, we go through some of the notable 3D surface simplification algorithms.

Most mesh simplification methods fall into four main categories: vertex clustering, wavelet decomposition, pruning based algorithms, and pairwise nearest neighbour based algorithms.

2.2.1 Vertex Clustering

Vertex clustering methods [RoB93, Low97, HHK+95] spatially partition the vertex set into a set of clusters. They are generally very fast and work on arbitrary collections of triangles. Unfortunately, they can often produce relatively poor quality approximations. The simplest clustering method is the uniform vertex clustering algorithm described by Rossignac and Borrel [RoB93]. A simple example of uniform clustering is shown in Figure 2.2. The vertex set is partitioned by subdividing a bounding box on a regular grid,

and the new representative vertex for each cell is computed using cheap heuristics based on criteria such as edge length. This process can be implemented quite efficiently. The algorithm also tends to make substantial alterations to the topology of the original model. Looking at Figure 2.2, we can see that the triangle in the upper left corner is reduced to a point, and two separate components along the top are joined together. Note that, much like uniform sub-sampling images, the results of this algorithm can be quite sensitive to the actual placement of the grid cells. It is also incapable of simplifying features larger than the cell size. A planar rectangle consisting of many triangles, all larger than the cell size, will not be simplified at all, even though it can be approximated without error using two triangles.

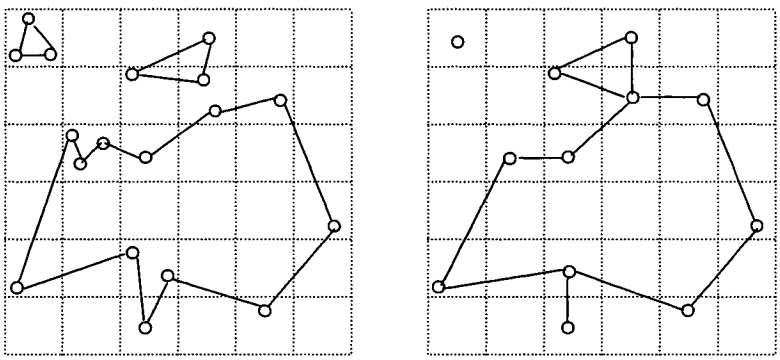


Figure 2.2 Uniform clustering in two dimensions.

The most natural way to extend uniform clustering is to use an adaptive partitioning scheme. Low and Tan [LT97] improved on [RoB93] by having vertex clusters form around important vertices rather than around locations in the model's bounding volume. Unlike [RoB93], the algorithm does not uniformly subdivide the model's volume. Initially the vertices of the model are weighted and sorted. Once all the vertices have been weighted, they are sorted. The vertex with the largest weight is removed from the list. If there is a cell at that location, then the vertex is added to that cell. If multiple cells are

intersecting at the point, then the vertex is added to the cell whose centroid is closest to the vertex. If no cell exists at that location, then a new cell is created and the vertex becomes the centroid of the cell. The process iterates until the list is empty. The size of the output model depends on the cell size. The larger the cell the smaller is the output model.

Definition 2.11: Voxel, short for volume pixel, is the smallest distinguishable box-shaped part of a three-dimensional image.

He *et al.* [HHK+95] proposed a voxel based simplification algorithm. This algorithm starts off by subdividing the bounding volume of the object into a regular three-dimensional grid. Then at each grid point, a low-pass filter is used to compute the centroid. Once all the grid points have been processed, a mesh fitting technique, i.e., marching cubes [Lore87], is employed to produce a simplified mesh. The density of the grid and the size of the low-pass filter dictate the size of the simplified model. Since marching cubes is known to create models with many polygons, models with a higher number of faces than the input may result. To avoid creating too many triangles each voxel is limited to five triangles. An advantage of this algorithm is that it is not only applicable to polygonal models but also to spline models, volume datasets, objects derived from range scanners, and algebraic mathematical functions such as fractals. This algorithm is slower than [RoB93] but produces better results.

Clustering methods tend to work well, if the original model is highly over-sampled and the required degree of simplification is not too great. They also tend to perform better when the surface triangles are smaller than the cell size. Since no vertex moves further

than the diameter of its cell, clustering algorithms provide guaranteed bounds on the Hausdorff approximation error sampled at the vertices of M and M'. However, to achieve substantial simplification, the required cell size increases quite rapidly, making the error bound rather weak. In particular, at more aggressive simplification levels, the quality of the resulting approximations can quickly degrade.

2.2.2 Wavelet Decomposition

Wavelet methods provide a fairly clean mathematical framework for the decomposition of a surface into a base shape plus a sequence of successively finer surface details. Approximations can be generated by discarding the least significant details. They have been used quite successfully for producing multi-resolution representations of signals and images. Due to its high computation cost, few researchers applied it into 3D-object simplification.

A two-stage method for multi-resolution wavelet modeling of arbitrary triangulated polyhedra was developed by Lounsbery, Eck *et al.* [Lou94, EDD+95]. This approach can be applied to any triangulated manifold with boundary. The algorithm first constructs a base mesh that is a triangulated polyhedron with the same topology as the input surface. Geodesic-like distance measures are used in this step. It then uses repeated quaternary subdivision of the base mesh to construct a new mesh that approximates the input surface very closely. A multi-resolution model of the new mesh is then built using wavelet techniques, after which an approximation at any desired error tolerance can be quickly generated. While this approach is very attractive for interactive surface design and surface optimization, it is not the best for multi-resolution modeling of static surfaces because of

the cost of resampling. Another disadvantage is that the method does not resolve creases at arbitrary angles well, since the final mesh subdivides the triangles of the base mesh on a regular grid.

2.2.3 Pruning Based Algorithms

The next class of simplification algorithms uses the pruning technique, which generally produces better simplifications than vertex clustering algorithms. These simplification algorithms work by locating relatively planar regions in a model, approximating those regions with a plane, and then retriangulating the area with fewer polygons. Some of these algorithms have trouble with simplifying models that contain many sharp edges and few planar areas.

The superfaces [KT96] algorithm simplifies a model by growing roughly planar patches on the surface of a model until the entire surface is covered. Once the whole surface is covered the patches are retriangulated with fewer polygons. The algorithm performs the simplification in three stages. In the first stage the so-called cells are created. Initially, the cells are empty. Faces are added to the existing cells if they satisfy the following conditions: 1) the face is adjacent to the patch in the cell, and 2) the face and the patch are relatively coplanar. The patches in the resulting cells are called superfaces. If the face does not meet the above criteria then the process starts a new cell. The process stops when all the faces in the original model have been added to the cells. In the second stage, the number of edges at the superface parameter is reduced. This is accomplished by creating superedges from the boundary edges of the superface. This procedure is similar to the one used to create the superfaces except it works on edges. A superedge is a single

edge that represents many smaller edges. The criterion used to determine if a superedge can be created for a set of edges is that all vertices in the original boundary are within a certain distance of the superedge. The third stage retriangulates the patches by subdividing them into star polygons and then triangulates each star polygon.

This algorithm is significantly slower than [RoB93] but produces much better simplifications. It cannot accept non-manifold surfaces as input but it preserves both topology and manifoldness of a model. Also, it does not explicitly preserve boundary edges.

Turk [Tur92] developed an algorithm that simplifies the surface by retiling it with fewer polygons. The algorithm randomly sprinkles a new set of vertices onto the original surface. The original vertices are then deleted, if possible, and the local area around the deleted vertices is retriangulated. The simplified model consists of the new vertices plus the original vertices that could not be deleted. To position a new vertex, an original face is selected using a random, area-weighted, and curvature-weighted choice, then the new vertex is randomly positioned in the face. This algorithm performs well on surfaces that have smooth curves, and which are mostly continuous. According to Turk, this algorithm does not work well on surfaces that contain many edges and sharp corners, e.g. man-made objects. One aspect of model simplification that [Tur92] does not handle is additional vertex information such as color and texture. No simplification speed results were presented and therefore it is difficult to say how fast this algorithm is, but its quality is comparable to [KT96]. This algorithm requires that the input models are manifold and it also preserves topology. The borders are not explicitly preserved.

The most widely used pruning based algorithm is vertex decimation, an iterative simplification algorithm originally proposed by Schroeder *et al.* [SZL92]. This method first classifies all the vertices into five classes based on their local geometry and topology: simple, complex, boundary, interior edge, and corner. In the second stage, the algorithm removes vertices from the original model by traversing a list of classified vertices. The list is traversed until all possible vertices are removed. A vertex is removed using the following criteria. If the vertex is a simple vertex and the distance from the vertex to a mean plane is below some threshold, then the vertex is deleted. The mean plane is constructed using the vertex's adjacent face normals weighted by the face area and the mean of all the face centers. If the vertex is a boundary vertex, then the threshold is compared to the distance from the vertex to the new edge that would be created if this vertex were to be removed. Usually, corner vertices are not eliminated. However, if the corner is not a significant visual detail, then the distance to plane criterion is used to decide whether or not to remove it. For each vertex removal, additional checks are made to ensure that the topology is preserved, and the resulting hole is retriangulated (Figure 2.3). The process stops when either no more vertices can be removed, or the number of vertices for the simplified model has been attained.

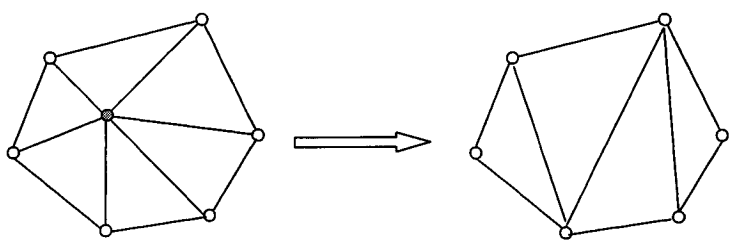


Figure 2.3 A vertex is removed and the resulting hole is retriangulated.

The algorithm of [SZL92] is reasonably efficient, both in time and space, and the simplification quality is comparable to that of [Tur92]. But it seems to have some difficulty preserving smooth surfaces, and again this algorithm is limited to manifold surfaces. The original vertex decimation algorithm [SZL92] used a fairly conservative estimate of the approximation error. More recent methods [KCS98, KLS96, SL96] use more accurate error metrics, like the localized Hausdorff error. They maintain a link between points on the original surface and the corresponding neighbourhood on the approximation, and the distances between these points and the associated faces define the approximation error. While these vertex decimation algorithms produce higher quality results, they are substantially slower and consume more space.

2.2.4 Pairwise Nearest Neighbour Based Algorithms

Simplification algorithms based on *pairwise nearest neighbour* are similar to algorithms based on *pruning* but are more complex and produce the highest quality simplifications. In terms of model simplification, *pairwise nearest neighbour* involves merging vertices in the polygonal mesh. These algorithms start with a set consisting of all vertices in the input model. The pair that would introduce the least distortion, if merged, is located. This pair is then merged. This process continues until the required model size or distortion is reached. The algorithms discussed below all simplify in the similar way. The primary differences between them are how they compute the distortion between patches before and after vertex merging and how they determine the position of the new vertex after vertex merging.

One of the first algorithms that use this simplification technique was developed by Hoppe *et al.* [HDD+93], which is called mesh optimization. This algorithm minimizes an energy function to compute the distortion and the new vertex. These two computations are unified in a single equation because they are interdependent and it is simpler to optimize both of them at the same time. The algorithm constraints the vertex merges to vertices that are connected by an edge and uses two additional transformations to simplify the model: edge split and edge swap. The algorithm randomly selects an edge from a set R of possible edges to be collapsed, split, or swapped. An edge collapse is applied first, then an edge swap, and finally an edge split. If any one of them reduces the energy function the transformation is performed and the edges affected by the transformation are added to the set R. If none of the transformations lowers the energy function, the edge is removed from the set and a new edge is selected. An edge collapse and an edge swap change the topology of the surface, hence checks are performed on the result of a transformation to ensure that the simplified surface still remains topologically equivalent to the original.

The simplifications produced by Mesh Optimization [HDD+93] are much better than the simplifications produced by the algorithms already presented. The algorithm is among the best ones in terms of the quality of simplification, but also one of the slowest.

An improved algorithm of Mesh Optimization, which is called Progressive Mesh, is also proposed by Hoppe [Hop96]. This algorithm iteratively gets rid of one vertex at a time by the operation of "edge collapse". As shown in Figure 2.7, an edge collapse transformation $ecol(\{u, v\})$ unifies 2 adjacent vertices u and v into a single vertex v'. The

triangles that contain the selected vertices now contain the vertex they were collapsed to, except that the two triangles (that initially contained both vertices) are now deleted. A position v' is specified for the new unified vertex. Thus, an initial mesh M^n can be simplified to a coarser mesh M^0 by applying a sequence of n successive edge collapse transformations. A key observation is that an edge collapse transformation is invertible. Hoppe called that inverse transformation a *vertex split*, as shown in Figure 2.7. We can therefore represent an arbitrary triangle mesh M as a simple mesh M^0 together with a sequence of n vertex split records.

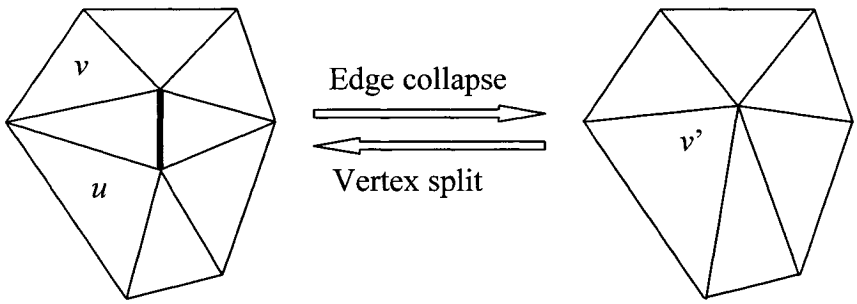


Figure 2.4 Illustration of the edge collapse and vertex split.

Compared to Mesh Optimization, this algorithm still uses and minimizes an energy function to compute the distortion and guide edge collapse, but it contains several new terms to account for additional values at the vertices. The number of edge transformations has been reduced to one: edge collapse, so that the computation of the distortion becomes simpler. This algorithm produces comparable quality to [HDD+93] and is faster than it, but it still may consume hours to simplify a complex object, which is considerably slower than many other simplification algorithms. However, its amazing progressive feature makes it suitable for progressive transmission and mesh compression. And many other

colleagues are making contributions to simplify the energy function and improve the speed.

The QSlim algorithm, developed by Garland and Heckbert [GH97], is considerably faster than those described in [Hop96, HDD+93]. This algorithm merges vertices that are connected by an edge or are within a certain threshold distance. The goal of QSlim is to find and merge a pair of vertices such that the resulting vertex will be as close to the original surface as possible. Two vertices will not be merged if the merge will cause a face to fold over onto another face. Distortion in [GH97] is a distance that measures the sum of squared distances from a vertex v to a set of planes. Each vertex v in the original model has an associated set of planes (polygons). These planes are adjacent to v and the sum of squared distances between these planes and v is zero. A 4×4 symmetric matrix Q representing the set of planes is associated with each vertex v. The error at the vertex is defined to be $v^{T}Qv$, and when a vertex pair is contracted, their matrices are added together to form the matrix for the resulting vertex. This method derives these matrices to calculate the sum of squared distance of the vertex to a set of planes. The distortion measure favours vertex that, when merged, produce a vertex that is closest to the original surface. For example, two vertices on a feature edge are merged before a vertex on a feature edge and a vertex that is not on the feature edge.

This algorithm is faster, by several orders of magnitude, than [Hop96, HDD+93]. Its simplifications are usually not as good as Hoppe's, but are considerably better than that of [RoB93]. It can simplify well those surfaces that are simplified poorly by pruning based algorithms. A big advantage of this approach is that it is possible to specify whether or not to preserve topology. By setting the vertex merge threshold distance to zero the

algorithm becomes a regular edge collapse algorithm and the topology will be preserved. By setting it to a value greater than zero, the topology will not be preserved. This algorithm expects a manifold model as the input.

The "memoryless" algorithm proposed by Turk and Lindstrom [LT98] is fast and memory efficient. Like [Hop96] [HDD+93], it constrains the vertex merges to vertices with edges between them. Unlike many simplification methods, it does not compare the current simplified model to the original after every simplification step. Instead, it tries to preserve the current simplification's geometric properties, such as volume and area, in the locality affected by each simplification step. For example, when an edge is collapsed, the local shape of the surface usually changes and there is a corresponding change in volume; or, if there is a border edge in the local area, the border may change in length and the local area it surrounds may change. This algorithm tries to minimize both of these changes. The resulting vertex is an optimization problem that tries to minimize the change in volume and area. The distortion is the weighted sum of terms minimized in the volume and boundary optimizations. The squared length of the edge is included as a scale in the distortion to ensure scale invariance. The triangle shape optimization is not included because it tends to penalize edges that otherwise have low values. As with other methods, simplification stops when the required size of the model is reached or no more edges can be collapsed.

This algorithm is faster than [Hop96] and slower than [GH97]. The simplification quality is also in between the above two methods. This method preserves the topology and is able to accept non-manifold models as the input. It also explicitly preserves the borders.

Another algorithm based on edge collapse was described by Ronfard and Rossignac [RR96]. The fundamental observation underlying their algorithm is that each vertex in the original model lies at the intersection of a set of planes, in particular, the planes of the faces that adjoin the vertex. They associate a set of planes with each vertex; they call this set the *zone* of the vertex. A vertex's zone is initialized to be the set of planes of the adjoining faces. The error at a vertex is measured as the maximum distance between the vertex and the planes in its zone. When contracting an edge, the zone of the resulting vertex is the union of the zones of the original endpoints. The error of this resulting vertex characterizes the cost of contracting the edge. At each iteration, the edge of the lowest cost is selected and contracted. The complexity of this algorithm is $O(nlogn)$.

Melax proposed a simple Progressive Mesh based polygon reduction [Mel98] scheme. Based on the face normals, the error metric is simply defined as:

$$E(u,v) = \max_{f \in T_u} \{\min_{n \in T_{uv}} \{(1 - f_{normal} \cdot n_{normal})/2\}\}$$

where T_u is the set of triangles that contain u and T_{uv} is the set of triangles that contain both u and v.

This algorithm is very efficient in terms of running speed. It can simplify the models with tens of thousands triangles within seconds. However, in some cases, it cannot preserve the model topology well during the simplification.

2.3 3D Mesh Segmentation Algorithms

Subdividing an image into its constituent parts or objects, segmentation is usually the first step in image processing. Autonomous segmentation is one of most important

and difficult tasks in image processing area [Gon92]. The segmentation was introduced to 3D areas recently, including convex polyhedra, range images and surface meshes. In this section, we will briefly introduce the first two areas and emphasis on the surface mesh segmentation review.

2.3.1 Convex Polyhedra Decomposition

Convex polyhedra decomposition is one of the 3D research areas in computational geometry. This problem seeks to decompose a non-convex polyhedron into smaller convex ones. The motivation of this work is to improve the rendering and shading in some computer graphics applications.

The original idea is introduced by Chazelle in [Cha84] and followed up in [CP97]. Other researchers [BD92, HS98, TCC+00] have also contributed with significant interest growing. Most of these algorithms seek to find the simplest decomposition possible. Lingas [Lin82] shows the minimum decomposition complexity is NP-hard.

Definition 2.12: A problem is NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problem (non-deterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem", although it might be harder.

Hence the heuristics are necessary, and as a result the main focus of the research attempts to bound the worst-case complexity of the problem. In [SS01], a computer vision based approach to convex decomposition is proposed. This method is less rigorous than the computational geometry approaches above and yields nice results. A volumetric distance transform guides the segmentation process.

This research fits nicely with the scene modeling goals for real-time visualization. However, the decomposition of an object as a collection of convex parts is not very meaningful. Additionally, these algorithms assume that the non-convex polyhedra are ideal models without measurement errors. Thus, it is restricted to certain computer graphics applications.

2.3.2 Range Image Segmentation

Another area related to the mesh segmentation is range image segmentation. Range images are obtained by different types of range scanners. 3D coordinates can be recovered from the range image if the calibration of the scanner is known. However, a single range image does not completely represent the scanned object due to occlusions. For this reason, the range image is often called 2.5 image. Hoover *et al.* [HJJ96] provide a review of range image segmentation approaches and establish a framework for comparing these algorithms. Suk *et al.* [SB92] also provide a survey and present the fundamental groundwork for the problem itself. More recent work is included in [BGG+96, BWA98, YL99, FRS02]. Of these methods, the watershed segmentation of [BGG+96] is noteworthy. Baccar *et al.* use image processing watersheds and data fusion techniques to identify surface discontinuities. We will further discuss the watershed algorithm in surface mesh segmentation in the next section and we will use this method in our work as well. In terms of part decomposition, the algorithm of [FRS02] demonstrates nice results. They add an additional level of complexity beyond segmentation of range images by assigning functionality to the resulting part decompositions.

Although this research is not interested in segmenting range images, the proposed method can also be applied to range image segmentation because a triangulated surface can be easily obtained from a single range image.

2.3.3 Surface Mesh Segmentation

We have discussed the convex polyhedra decomposition and the range image segmentation. These objects are not true 3D meshes. A 3D mesh has arbitrary connectivity in its topology. The topic of segmenting a 3D mesh is relatively new but has important applications in CAD and Virtual Reality Environments. Segmentation of a 3D scene can extract 3D objects from the scene, which may help scene understanding and 3D object recognition. The algorithms of 3D mesh segmentation can be classified into the following 5 categories:

1. Region growing
2. Clustering based algorithms
3. Spectral analysis
4. Watershed based algorithms
5. Other approaches

We will discuss each of these approaches in this section.

2.3.3.1 Region Growing

Region growing is the most popular 3D mesh segmentation method. It is a local-greedy approach. This process starts with selecting one or more "seed elements" in the 3D mesh; the seed element could be a point or a face. The number of seed elements is equal to the number of regions to be detected. Then a criterion for similarity between

elements must be formulated. The region growing process is then started by inspecting all elements surrounding the seed elements. If an element is sufficiently similar to its seed element, it is assigned to that seed element's region. For each region the process is continued by inspecting all elements surrounding the elements that are already assigned to that region until all elements in the mesh have been assigned. The main differences between various algorithms that use region growing are in the seeds selection mechanism and the criteria that determine if an element can be added to an existing region. Other issues in region growing include dealing with too small regions (for example, if a single face cannot be clustered to any of its neighbouring regions), and post-processing of the segmentation borders for smoothing or straightening.

In [SCG+02], Sorkine *et al.* develop a method that simultaneously segments the mesh and defines a parameterization. The seed faces are chosen randomly and greedy region growing is initialized that is capable of optimizing different criteria. For parameterization the criterion for adding a face to a region measures the distortion caused to a triangle during flattening to 2D. This is done using the singular values of the Jacobian of the affine transformation between the original 3D triangle and its counterpart in the 2D plane. The super-face algorithm [KT94, KT96] also uses region growing with random starting faces. It utilizes a set of representative planes for the cluster approximated by an ellipsoid. The clustering criteria used are an $L\infty$ face-distance along with a geometric constraint that prevents a face from folding-over its representative planes. The borders between the segments are straightened in a post-processing stage.

Eck *et al.* [EDD+95] employ a multiple source region growing to create a base triangle mesh with subdivision connectivity. The original purpose is for mesh simplification, but it can also segment the surface into patches. The main idea is to create Voronoi-like patches on the mesh and use the dual of the patches as the base triangular mesh. This imposes three constrains on the patches: a patch must be homeomorphic to a disk; two patches cannot share more than one consecutive boundary; and not more than three patches can meet at a vertex. An approximation of geodesic distance between faces is used as the priority for selecting faces. This algorithm starts with one seed and then iteratively adds another seed in places where one of the constraints are violated, until all the above constraints are met.

Levy *et al.* [LPR+02] use region growing for texture atlas generation. But instead of using seed faces and going outward, the algorithm first extracts feature contours and uses them as boundaries between charts to grow the region inward. This also simplifies the test criteria that determine if an element can be added to an existing cluster since the boundaries are somehow pre-determined.

2.3.3.2 Clustering Based Algorithms

Clustering of data is a method by which large sets of data are grouped into clusters of smaller sets of data. A clustering algorithm attempts to find natural groups of components (or data) based on some similarity. The clustering algorithm also finds the centroid of a group of data sets. To determine cluster membership, most algorithms evaluate the distance between a point and the cluster centroids. The output from a

clustering algorithm is basically a statistical description of the cluster centroids with the number of components in each cluster.

Definition 2.13: The centroid of a cluster is a point whose parameter values are the mean of the parameter values of all the points in the clusters.

Clustering has various applications, such as pattern recognition, artificial intelligence, cybernetics, biology, policy, and decision science, etc. It has been brought to the 3D mesh segmentation area recently. The clustering based algorithms available today can be classified into two broad categories as follows.

1. Hierarchical methods

In these methods, the input data is not partitioned into clusters in a single step. A series of successive fusions (merging) of data are performed until the final number of clusters is obtained. Compared with the region growing method, while hierarchical clustering is still a greedy approach, it can be seen as "global-greedy" since it always chooses the best merging operation for all clusters and does not concentrate on growing one. The difference between various hierarchical clustering algorithms lies mainly in the merging criteria.

In 3D mesh segmentation, hierarchical clustering starts initially when each face is its own cluster. Each pair of clusters is assigned a cost for merging. In [GWH01], the hierarchical face clustering algorithm proposed by Garland *et al.*, uses L_2 distance and orientation norms from representative planes as a measure of planarity, but formulates them using quadric error metric for efficient computation as in Garland's surface simplification algorithm described above. The algorithm also uses a bias term to create

circular compact cluster shapes by using the ratio between the square of the perimeter and $4\pi A$ where A is the area of the cluster.

Sander *et al.* [SSG+01, SWG+03] proposed a similar clustering algorithm for the creation of charts. They measure planarity as the mean-squared distance of a patch to the best fitting plane through the chart. However, the measure is integrated on all patch faces instead of only the vertices as in [GWH01]. Compactness of the patches is measured simply as the squared perimeter length. Additional tests are performed before merging two clusters to take care of topology constraints such that each clustered patch remains homoeomorphic to a disc. In the post processing, smooth boundaries between the charts are created by calculating constrained shortest path.

2. *K*-means clustering

In the hierarchical clustering method the number of the resulting clustering is unknown in advance. A different type of search for segmentation is defined by iteratively searching for the best clustering given that the number of clusters is fixed. The basis of this is the *k*-means algorithm, sometimes referred to as Lloyd or Lloyd-Max algorithm.

Definition 2.14: The *k*-means clustering method initially takes the number of components of the population equal to the final required number of clusters. In this step itself the final required number of clusters k is chosen such that the points are mutually farthest apart. Next, it examines each component in the population and assigns it to one of the clusters depending on the minimum distance. The centroid's position is recalculated every time a component is added to the cluster and this continues until all the components are grouped into the k clusters.

The key issue concerning the *k*-means clustering algorithm is convergence. The measure of optimal centroid (or representative) for an element and the computation of new representatives from the clusters should be chosen with care so that the process converges. Other issues such as the choice of initial centroids can also affect the convergence and the final result.

To create compatible segmentation of two objects for morphing purpose, a *k*-means based face-clustering algorithm is proposed in [STK02]. A distance measure between faces is defined as weighted combination of the approximate geodesic distance (the sum of distance from centroid to the center of edge) and the difference in dihedral angle. After representatives are chosen each face is assigned to the cluster of its closest representative. New representatives are chosen as the faces that minimize the sum of the distances of all the other faces in the cluster.

Mesh charts for geometry image creation are defined in [SWG+03] using *k*-means clustering. This algorithm also ensures connectivity by adding only neighbour triangles to the existing charts. The cost of adding is a measure of geometric distance between the face and its neighbour faces in the chart and the difference between the face normal and the chart normal. The new seeds for the next iteration are simply the central faces in each chart. To assure the disc topology of all the charts some face assignments are disallowed. This may lead to a possibility of an orphan face left not clustered. The solution to this is to add this face as a seed in the next iteration, hence enlarging *k* by one. This idea is also used to initialize the seed set by adding a new seed in the last face assigned in the

previous iteration as a new seed in the next iteration, starting from 1 seed until k seeds are created.

2.3.3.3 Spectral Analysis

Spectral graph theory [Chu97] states the relationship between the combinatorial characteristics of a graph and the algebraic properties of its Laplacian [Got03]. If A is the adjacency matrix of a graph G and D is a diagonal matrix that holds the degree (valance) of vertex i as $d_{i,i}$, then the Laplacian of G is defined as the matrix $L = D - A$. Let $\{\xi_0, \xi_1, \ldots, \xi_{n-1}\}$ be the eigenvectors of L. By embedding the graph G into the space R^d using first d eigenvectors, one can reduce the combinatorial graph-partitioning problem to a geometric space-partitioning problem [AY95, Got03].

The Laplacian matrix of the vertex adjacency graph was used for the mesh compression purpose in [KG00]. Due to high computation cost the mesh was segmented into smaller sub-meshes and each one was treated separately. However, these sub-meshes should be balanced in size and the edge straddling the different sub-meshes should be minimized in order to reduce the visual effects. Using a slightly different formulation in [LZ04] a symmetric affinity matrix $W \in R^{nxn}$ is constructed where for all i and j, W_{ij} encodes the probability that face i and face j can be clustered into the same patch ($0 \leq W_{ij} \leq 1$). This matrix may be viewed as the adjacency matrix of a complete weighted graph whose nodes are the mesh faces. The spectral analysis of this matrix creates a partitioning that can lead to a segmentation of the mesh.

2.3.3.4 Watershed Based Algorithms

We have introduced region growing, clustering based algorithms, and spectral analysis methods. Another important category for mesh segmentation is the watershed based algorithm. The watershed based algorithm is one of the most important approaches in image processing. Recently it has gained a lot of attention in the 3D mesh area. As we will discuss later, we have an interest in it as well. Here we will introduce the general watershed algorithm and review the 3D watershed segmentation literature.

The watershed transform can be classified as a region-based segmentation approach [RM01]. The intuitive idea underlying this method comes from geography: imagine the landscape being immersed in a lake, with holes pierced in the local minima. Basins (also called "catchment basins") will fill up with water starting at these local minima, and at points where water coming from different basins would meet, dams are built. When the water level has reached the highest peak in the landscape, the process is stopped. As a result, the landscape is partitioned into regions or basins separated by dams, called watershed lines or simply watersheds.

The watershed segmentation method was first generalized from 2D image processing to 3D mesh segmentation by Mangan and Whitaker [MW99]. In their work, the total curvature value at each vertex are estimated and viewed as the vertex height. The surface is segmented into patches based on the 3D watershed, where each patch has a relatively consistent curvature throughout. To find the curvature of surfaces represented by polygonal meshes, they use either the discrete curvature or the norm of the covariance of the surface normals adjacent to the vertex in question. For the magnitude of the curvature, they use the deviation from the flatness, which is defined to be the Euclidean norm of the

shape tensor. And to find the covariance matrix, the variance and covariance in all the three directions must first be found. Thus, this algorithm is computationally costly. In their work, they implemented the top-down method for the watershed algorithm. To address over segmentation, they have developed an intricate filter-and-merge algorithm. This algorithm looks for watersheds with relatively low water depths and merges such regions with neighbouring ones.

Several other algorithms have been proposed for the height function definition. Pulla *et al.* presented an improved discrete curvature estimation. They use a technique where the surface is approximated locally by a biquadratic polynomial whose second order partial derivatives are calculated. An added advantage of fitting a surface locally is that the sensitivity of the surface to noise can be adjusted by using smoothing equations. A more elaborate function is used in [PKA03] by Page *et al.*. It defines a directional curvature height function between each two adjacent vertices u and v using the Euler's formula: $f_{uv} = k_{max} \cos^2\theta + k_{min} \sin^2\theta$, where k_{max} and k_{min} are the maximum and minimum curvatures at u, and θ is the angle between the maximum principal direction and the vector connecting u to v in the tangent plane of u. The computational costs of these two algorithms are high too. Zhang *et al.* [ZPK+02] proposed a simple 3D segmentation algorithm using discrete Gaussian curvature only. It is efficient for small sized and certain types of meshes, but may fail to segment correctly for a large mesh with concave corners.

Rettmann *et al.* [RHX+02] proposed a mesh segmentation approach that is applied to extract regions of cortical surface that surround sulci. Gyral and sulcal regions are initially labeled based on its Euclidean distance to a shrink wrap surface that is a

deformable surface fitted to the original cortical surface. The height map is obtained by computing the geodesic distance from sulcal regions to gyral regions. This approach is effective; however, it is suitable for sulcal segmentation only.

2.3.3.5 Other Approaches

Wu and Levine [WL97] introduced simulated electrical charge distributions for surface mesh segmentation based on the physical fact that electrical charge on the surface of a conductor tends to accumulate at a sharp convexity and vanish at a sharp concavity. From an initial triangle that is the local minimum of the charge density, the algorithm proceeds to the neighbour with the lowest charge density. The tracing process detects the object part boundaries denoted by the sharp concavity. This approach can be regarded as edge-based segmentation. Similar to other 2D edge-based segmentation methods, the segmentation fails when the boundary is not connected due to noise.

Li *et al.* [LTH01] presented a mesh segmentation approach based on space sweeping. The first step is the skeletonization of the 3D object. Skeletal edges are extracted by mesh simplification. A skeleton tree is obtained to determine a traversal order by adding virtual edges to connect disjointed skeletal edges. In the course of sweeping plane movement along skeletal edges, the geometric and topological functions are computed and analyzed. When a consecutive pair of critical points is found, the part of the polygon mesh that is swept is extracted as a component.

2.4 3D Mesh Compression Algorithms

In general, data compression consists of taking a stream of symbols representing some "information", and transforming it into a new (usually more compact) stream of

codes. Various encoding schemes have been evolved to capture the structure in the data representing specific information in order to most effectively compress specific media types, such as text, images, video, and audio, etc. Several standards have been made, such as JPEG for image compress, and MPEG series for video compression. 3D polygonal mesh, as a relatively new type of digital media, has been in use in an increasing number of industrial, business and entertainment applications. And the 3D mesh compression is essential for reducing storage requirements and increasing transmission speed over the Internet. The 3D polygonal meshes are significantly different from other media types. A regular structure is not found in the 3D models. Hence there are no natural extensions of the compression techniques that were developed for the other media to compress 3D data. Basically, to present a 3D polygonal model, three types of data are used, i.e., connectivity data, geometry data, and attribute data. Attribute data includes color, normal and so on. A mesh compression algorithm encodes these three types of data separately as shown in Figure 2.5. Connectivity encoding is the central part of any 3D compression method. It guides both geometry coding and attribute encoding. Most of the existing methods focus on connectivity and geometry coding. And the early work only concentrates on the connectivity coding.

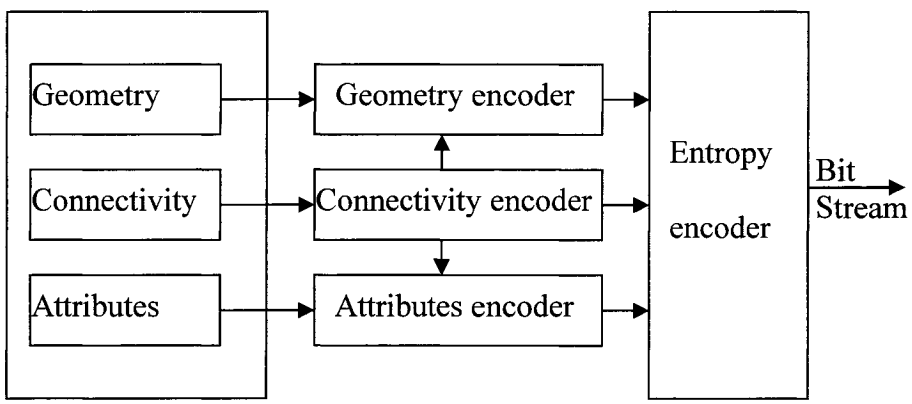


Figure 2.5 3D Mesh encoding.

In this section, we present an overview of the different classes of polygonal mesh compression techniques, including single resolution connectivity compression, progressive compression, and geometry compression.

2.4.1 Single Resolution Connectivity Compression

Most of the existing mesh compression algorithms are falling into the single-resolution category. Single-resolution mesh compression requires lossless connectivity coding. Typically the number of triangles in a mesh is roughly twice the number of vertices and each vertex is referenced in 5 to 7 vertices, which means that a large part of representation of the model is the definition of connectivity among the vertices. Encoding schemes that minimize repeated references to vertices result in compression. We classify the single rate connectivity compression algorithms into four classes: the triangle strip based techniques, the graph traversal methods, the spectral approach, and the remeshing based approaches.

2.4.1.1 Triangle Strip Based Techniques

Historically, the first 3D mesh compression algorithms aimed at reducing the transmission bandwidth between the CPU and the graphics card of a computer. The basic principle of these hardware-oriented compression methods consists of decomposing the mesh into sequences of adjacent triangles, so-called Generalized Triangle Strips (GTS), which can be encoded with less redundancy (about one vertex index per triangle) than the uncompressed indexed face set representation used in VRML format [VRML] (which requires three vertex indices per triangle). In his pioneering work [Dee95], Deering introduced the Generalized Triangle Mesh (GTM) representation that combines a GTS

with a 16-vertex buffer. Once a vertex is stored in this buffer, it can be accessed by a 4 bits index. Under the assumption of the reuse of each vertex from the buffer only once, Taubin and Rossignac [TR98] proved that GTM codes the connectivity at about 11 bits per vertex (bpv).

However, Deering did not provide any decomposition scheme to convert an arbitrary mesh into the GTS representation. Subsequent work has been so dedicated to finding some optimal GTS. In [Cho97], Chow was the first to propose a GTS decomposition algorithm that optimizes the Deering's vertex buffer use. The challenging issue of finding an optimal GTS decomposition was also studied in [AHM96] and some heuristic solutions were provided in [ESV96, XHM99]. As such techniques are designed for the purpose of display optimization, they are very simple, hardware oriented, as well as real time applications dedicated. However, they result in very low compression rates.

2.4.1.2 Graph Traversal Methods

The graph traversal methods aim at data storage optimization. They use the deterministic traversal to capture the adjacency relations between the vertices and do an implicit coding of the connectivity information.

Turan [Tur84] observed that the connectivity of a planar graph can be encoded with a constant number of bpv using two spanning trees: a vertex spanning tree and a triangle spanning tree. Based on this observation, Taubin and Rossignac [TR98] presented a topological surgery (TS) approach to encode mesh connectivity. The basic idea is to cut a given mesh along a selected set of cut edges to make a planar polygon. The mesh connectivity is then represented by the structures of cut edges and the polygon, i.e., two

spanning trees. The two trees are run-length encoded. A run is defined as a tree segment between two nodes with degrees not equal to 2. The degree (valance) of a vertex is the number of edges incident on the vertex. In both vertex and triangle spanning trees, a run is a basic coding unit. Thus the coding cost is proportional to the number of runs, which in turn depends on how the vertex spanning tree is constructed. The TS algorithm builds the vertex spanning tree in a way that is similar to what we peel an orange along a spiral path, to maximize the length of each run and minimize the number of runs generated. Compared with the triangle strip based method, TS is much efficient in connectivity coding, which costs 2.48 ~ 7.0 bpv. However, TS cannot directly handle non-manifold meshes. This method also demands a large memory buffer due to its global random vertex access at the decompression stage. Because of its simplicity, the TS compression scheme has been adopted within the MPEG-4 compression standard.

Bajaj *et al.* [BPZ98] proposed a connectivity coding method using a layered structure of vertices. It decomposes a triangular mesh into several concentric layers of vertices, and then constructs triangle layers within each pair of adjacent vertex layers. The mesh connectivity is represented by the total number of vertex layers, the layout of each vertex layer, and the layout of triangles in each triangle layer. Ideally a vertex layer does not intersect itself, and a triangle layer is a generalized triangle strip. In such a case, the connectivity compression is reduced to the coding of the number of vertex layers, the number of vertices in each vertex layer, and the generalized triangle strip in each triangle layer. However, in practice overhead bits are introduced due to the existence of branching points, bubble triangles and triangle fans. The layered decomposition method can encode the connectivity information at about 1.40 ~ 6.08 pbv. It also has a nice property, which is

that each triangle depends on at most two adjacent vertex layers and each vertex is referenced by at most two triangle layers. This property enables the error-resilient transmission of mesh data, since the effects of transmission errors can be localized by encoding different vertices and triangles independently.

The triangle conquest approach starts from the initial borderline, which divides the whole mesh into conquered and unconquered areas, and inserts triangle by triangle into the conquered parts. The output is a stream of building operations of new triangles. The two main representatives of triangle conquest approach are Gumhold and Straßer's Cut-border Machine [GS98] and Rossignac's Edgebreaker [Ros99]. At each step, the Cut-border Machine approach inserts a new triangle into the conquered area, closed by the cut-border, using one of the five building operations: "new vertex", "forward", "backward", "split", and "close". The sequence of the building operations is encoded using Huffman codes. The offset value associated with the split operation is used to re-play the split operation during reconstruction. This makes it possible to decode the mesh connectivity in a single forward traversal of all the operations, which allows encoding and decoding to run in parallel. Experimentally its compression performance lies within 3.22 ~ 8.94 bpv. Rossignac's Edgebreaker is nearly equivalent to the Cut-border Machine, except that it does not encode the offset data associated with the split operation. This algorithm gives the worst guaranteed bit-rates for triangle mesh connectivity. It requires at most 4 bpv for any triangular mesh that is homeomorphic to a sphere and supports the general manifold meshes with an arbitrary number of boundary loops or handles. King and Rossignac [KR99] modified the Edgebreaker method to guarantee a worst-case coding cost of 3.67 bpv for the simple meshes. Szymczak *et al.* [SKR00] optimized it for

meshes with high regularity by exploiting dependencies of output symbols. It guarantees a worst-case performance of 1.622 bpv for sufficiently large meshes with high regularity.

Similar to the triangle conquest approach, the valence-driven method introduced by Touman and Gotsman [TG98] starts from a seed triangle whose three edges form the initial borderline. The borderline divides the whole mesh into two parts, i.e., the inner part that has been processed and the outer part that is to be processed. Then the borderline gradually expands outwards until the whole mesh is processed. The main difference is that the valence-driven approach outputs a stream of vertex valences, from which the original connectivity can be reconstructed. The encoder maintains a list of vertices called an active list. At each step, one vertex of the active list is processed and its valence is output. The particular cases corresponding to self-intersections of the active list are handled by applying some specific "split" and "merge" operations, encoded by the dedicated exception codes. The connectivity of the mesh is completely represented by the sequence of valences and exception codes. Since for a regular mesh the valence distribution is concentrated around six with low dispersion, the valence sequence can be efficiently compressed by an arithmetic coder. Touma and Gotsman's algorithm achieves a compression ratio of 0.2 bpv for the connectivity of regular meshes and 2 ~ 3.5 bpv otherwise. This algorithm has been considered as the state-of-the-art single resolution compression scheme.

2.4.1.3 Spectral Compression

The spectral analysis of functions defined on polyhedral surfaces of an arbitrary topology was first introduced by Taubin [Tau95]. He generalized the Laplacian operator

to arbitrary connectivity enabling discrete Fourier analysis of 3D meshes. By analogizing with 2D images compression schemes, Karni and Gotsman [KG00] proposed to project the mesh geometry on the basis of the Laplacian eigenvectors. They showed experimentally that for smooth meshes the resulting coefficients rapidly decrease to 0. Moreover, truncating the high frequency coefficients leads to smooth versions of the mesh, visually very close to the original one. As the complete connectivity information is needed during both the coding and the decoding processes in order to build the Laplacian matrix and compute its eigenvectors. The spectral method can naturally support progressiveness by transmitting the coefficients in the increasing order of frequencies. But because of the required eigenvector estimation, the computation complexity is $O(n^3)$ (n being the number of vertices) which makes it prohibitive for large meshes that can have thousands of vertices. In order to overcome such limitation, Karni and Gotsman [KG01] presented a method that embeds the original mesh into a 6-regular one and then uses a sub-optimal fixed Fourier base. The 6-regular mesh is the mesh in which each interior vertex has exactly 6 neighbours.

2.4.1.4 Remeshing Based Approaches

Unlike the above compression techniques, remeshing based compression methods target the applications where the topological changes are allowed while the shape must be preserved. The initial connectivity can be completely discarded and transformed into a regular or semi-regular one. The advantage of this class of approaches is the gain obtained in term of bit-rate as only some minimal information is required for generating the regular connectivity.

Gu *et al.* [GGH02] introduced a new representation for a 2-manifold mesh, so-called "geometry image". It captures the geometry as a simple 2D matrix of vertex coordinates (x, y, z). To create a geometry image, they cut an arbitrary mesh along a network of edge paths, and parametrize the resulting single chart on a square. Surface signals like normals and colors are stored in similar 2D arrays using the same implicit surface parametrization. The resulting geometry images can then be encoded using traditional image compression algorithms, such as wavelet-based coders.

Szymczak *et al.* [SRK02] proposed a geometry-driven subdivision of the mesh. At first the initial mesh is segmented into reliefs, which are relatively flat regions that have smooth boundaries. The surface is then resampled in a regular manner within each of the reliefs. As a result, a piecewise regular mesh having a regular structure on large regions is obtained. The resulting regular mesh can be efficiently encoded by applying a version of the Edgebreaker algorithm optimized for regular meshes.

2.4.2 Multi-resolution Compression

Multi-resolution compression, as known as progressive compression, has been investigated for the progressive transmission of large scale models over networks with limited bandwidth. In progressive compression and transmission, a coarse mesh is transmitted and rendered first. Then refinement data are transmitted to enhance the mesh representation until the mesh is rendered in its full resolution or the transmission task is cancelled by the user. Generally speaking, a progressive coder is less effective than a single-rate coder in terms of the coding gain, since it cannot exploit the correlation in mesh data as freely as the single resolution coder.

In [Hop96], Hoppe first introduced the concept of progressive transmission and compression with a new mesh presentation called Progressive Mesh (PM). This algorithm simplifies a given orientable manifold mesh with successive edge collapse operations. The inverse operation of edge collapse is vertex split which inserts a new vertex into the mesh together with the corresponding edges and triangles. The PM algorithm has been reviewed in section 2.2.4. The connectivity of base mesh M^0 and the split operations can be encoded using any single resolution coder [Hop96]. But $O(n\log(n))$ bits are needed to encode the topology of a mesh with n vertices. Thus, it is not an efficient compression scheme. To improve the efficiency, Hoppe [Hop98] proposed another PM implementation. It reorders the vertex split operations to increase the compression ratio at the cost of quality degradation of intermediate meshes.

There are some other extensions to the PM coder. Popvic and Hoppe [PH97] generalized the PM representation to the Progressive Simplicial Complex (PSC), which handles more general topologies. Pajarola and Rossignac [PR00] proposed a modified PM. Instead of encoding every vertex split operation individually, their technique groups simplification and refinement operations into groups to increase the connectivity coding efficiency. As a result, the selective refinement property of PM is sacrificed. Similarly the Progressive Forest Split (PFS) algorithm [TGHL98] proposed by Taubin *et al.* uses a forest split operation, which can be seen as a grouping of several consecutive edge split operations. This algorithm provides tradeoff between compression ratio and granularity.

Li and Kuo [LK97] introduced the concept of embedded coding to encode connectivity and geometry data in an interwoven manner. The geometry data as well as

the connectivity data are encoded progressively. Thus, as the coded data stream is received and decoded by the receiver, not only new vertices are added to the model, but also the precision of each old vertex position is progressively increased. This coding scheme is applicable to any polygonal meshes that are not restricted to the triangular ones. This algorithm requires about 20 bpv to decode a mesh model at the acceptable quality. However, at this bit rate, only one third of the total number of vertices and triangles are reconstructed, since a significant portion of bits are used to increase the precisions of important vertices rather than to increase the number of reconstructed vertices.

Khodakovsky *et al.* [KSS00] introduced a wavelet based technique for progressive compression. Their algorithm requires the source mesh to have a regular structure. They have defined wavelet basis over manifold surfaces for multi-resolution analysis. The multi-resolution analysis of the surface naturally provides levels of detail progressively. Following the base model, the subsequent data stores the detailed coefficients to refine the model. Compression is achieved as the coefficients are incrementally stored and are much smaller in magnitude. These small values can be encoded efficiently. Also the higher order coefficients can be ignored due to their negligible values. However, this method only works for manifold meshes with regular connectivity.

2.4.3 Geometry Compression

All the single resolution compression methods encode connectivity data losslessly, since the connectivity is a necessary discrete mesh property. However, geometry data are generally encoded in a lossy manner for reducing the number of bits used for representing

the vertex coordinates. In order to exploit high correlation between the positions of adjacent vertices, most compression schemes follow a three-step procedure: quantization of vertex coordinates, prediction of quantized positions, and entropy coding of prediction residuals.

Uncompressed geometry data typically specify each coordinate component with an IEEE 32-bit floating-point number. However, this precision is beyond human eyes' perceiving capability and is far more than needed for most applications. Thus, quantization can be performed to reduce the data amount without serious impairment on visual quality. Quantization is a lossy procedure since it represents a large or infinite set of values with a smaller set. Quantization techniques can be classified into scalar/vector and uniform/non-uniform ones [GG92]. Typical mesh geometry coding schemes uniformly quantize each coordinate at 8 ~ 16-bit quantization resolutions. In [Dee95, TR98, TG98, BPZ99], the same quantization resolution is globally applied. In [Cho97], a mesh is partitioned into several regions, and then different quantization resolutions are adaptively chosen for different regions according to local curvature and triangle sizes. Within each region, the vertex coordinates are still uniformly quantized.

After the quantization of vertex coordinates, vertex positions are predictively encoded. The prediction step exploits the correlation between adjacent vertex coordinates and is most crucial in reducing the amount of geometry data. A good prediction scheme generates prediction errors that have a highly skewed distribution, which are then encoded with entropy coders, such as the Huffman coder or the arithmetic coder.

Taubin and Rossignac [TR98] employed a linear prediction scheme, where the position of a vertex is predicted from a linear combination of positions of K previous vertices in the vertex spanning tree. The K previous vertices are uniquely selected along the path from the root to the current vertex in the vertex spanning tree. More specifically, the position v_n of the n-th vertex is given by:

$$v_n = P(\lambda, v_{n-1}, \ldots, v_{n-k}) + \varepsilon(v_n) = \sum_{i=1}^{K} \lambda_i \cdot v_{n-i} + \varepsilon(v_n)$$

where $\lambda_1, \lambda_1, \ldots, \lambda_k$ are chosen to minimize the mean square error:

$$E\left\{\left\|\varepsilon(v_n)\right\|\right\} = E\left\{\left\|v_n - \sum_{i=1}^{K} \lambda_i \cdot v_{n-i}\right\|^2\right\}$$

and transmitted to the decoder as the side information. Note that the delta prediction used by Deering [Dee95] is a special case of the linear prediction with $K = 1$ and $\lambda_1 = 1$.

The predictor used by Touma and Gotman [TG98] takes advantage of the geometric structure of the triangle strips for predicting the position of the next vertex in the stream. In a triangle strip, two adjacent triangles that share an edge form a quadrilateral. The position of the fourth vertex is predicted using the parallelogram rule. This rule predicts the position of the next vertex assuming a planar extension, and the error in prediction is used for further encoding.

CHAPTER 3

3 AN EFFICIENT MULTI-RESOLUTION MODELING ALGORITHM

In the previous chapter, we have reviewed surface simplification algorithms. Each method has its own advantages and disadvantages. Among them, the pairwise nearest neighbour based approaches are suitable for generating multi-resolution models. Progressive Mesh (PM) is the representative of this approach; other methods in this class are the extensions of the PM algorithm. The PM algorithm has some promising features; such as it can generate continuous resolutions while the surface topology is well kept, the original mesh can be restored using the reverse operation of simplification, and so on. Thus, this algorithm can be used for progressive compression and transmission over the Internet. Moreover, multi-resolution modeling is essential for object rendering in the interactive virtual environments. For example, when a 3D model moves farther from the viewer, for rending efficiency a simpler version of that object can be used and the viewer cannot notice the difference. Figure 3.1 shows multi-resolutions of a model "cow" at different distances.

The PM algorithm has a high computational cost, for a large model with thousands vertices, it takes almost an hour [Hop96] to simplify. More efficient PM-based approaches have been proposed. Some of them are very fast, such as QSlim [GH97] and

Melex's Polygon Reduction (PR) method [Mel98], which can simplify the similar model in seconds. However, in some cases, these methods can easily break the original topology. In this chapter, we present an efficient yet effective PM-based multi-resolution modeling algorithm that can run fast and well preserve the topology.

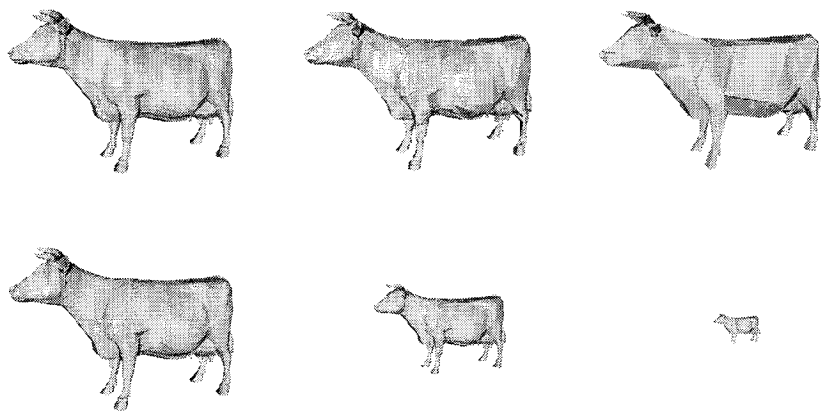


Figure 3.1 Multi-resolutions at different distances.

3.1 Analysis of Progressive Mesh based Multi-resolution Algorithm

3.1.1 Multi-resolution Representation

As mentioned in the last chapter, the core and basic operations of a Progressive Mesh algorithm are "edge collapse" and its inverse operation "vertex split" (Figure 2.4). Starting with the original model M^n, the PM based algorithms iteratively contract one edge for each step and generate a sequence of approximations

$$M^n \rightarrow M^{n-1} \rightarrow \ldots\ldots \rightarrow M^1 \rightarrow M^0$$

where M^0 is called base mesh. It is not necessary that the base mesh has 0 triangle. This successive representation encodes the original mode M^n, the final model M^0, and all intermediate approximations M^{n-1}, ..., M^1. This is a multi-resolution representation because it allows us to extract a fairly wide range of levels of detail.

The direct by-product of an iterative contraction (edge collapse) is an incremental representation, which we will term a simplification stream. During the process of simplification, we generate the sequence of models

$$M^n \xrightarrow{\phi^1} M^{n-1} \xrightarrow{\phi^2} M^{n-2} \xrightarrow{\phi^3} \cdots \xrightarrow{\phi^k} M^{n-k}$$

where each step $M^{i+1} \rightarrow M^i$ corresponds to the application of a single contraction ϕ^{n-i}. Thus each intermediate approximation M^i can be expressed as the result of applying some prefix of the total sequence of contractions

$$M^i = (\phi^{n-i} \bullet \phi^{n-i-1} \bullet \ldots \bullet \phi^1)(M^n) \qquad (3.1)$$

Suppose that along with the original mesh M^n we store a representation of each contraction ϕ^i. By simply applying (3.1), we can reconstruct any intermediate model M^i in addition to the original and final models. In essence, by storing a record of the simplification process, we can re-apply the same contraction sequence but choose an earlier stopping point. It should be noted that the key feature of the edge collapse is that it is invertible [Hop96]. The corresponding operation is a vertex split, which can be expressed as:

$$\varphi = \phi^1$$

Thus, starting with a simplified base mesh M^0, together with a sequence of vertex split records, we can also get any intermediate resolution representation as shown below.

$$M^0 \xrightarrow{\varphi^1} M^1 \xrightarrow{\varphi^2} M^2 \xrightarrow{\varphi^3} \cdots \xrightarrow{\phi^n} M^n$$

Each intermediate resolution can be expressed as

$$M^i = (\varphi^1 \bullet \varphi^2 \bullet \ldots \bullet \varphi^i)(M^0) \qquad (3.2)$$

This feature well fits the applications of the progressive transmission.

3.1.2 Implementation of Multi-resolution

A PM based simplification is a greedy algorithm. It begins with the original mesh and iteratively removes the vertices and faces from the model. Each iteration involves the application of a single atomic operation: edge collapse. An edge collapse, which is denoted as $(u, v) \rightarrow v'$, modifies the surfaces in three steps: (see Figure 3.2).

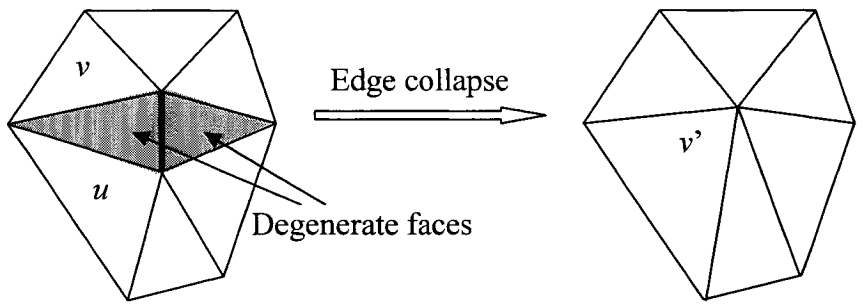


Figure 3.2 Edge (u, v) is collapsed to a new vertex v'.

1. Move the vertices u and v to the position v'.
2. Replace all occurrences of v with u.
3. Remove v and all faces, which become degenerated (no longer have three distinct vertices).

The first step modifies the geometry of the surface, and the second step modifies the connectivity of its mesh. Depending on the structure of the mesh, this may also alter the topology of the surface. The final step simply removes the elements of the surfaces that are no longer needed. For the example shown in Figure 3.2, one vertex and two faces are removed from the model.

The algorithm we have developed, like most related methods, is a simple greedy procedure. It produces a large-scale simplification by applying a sequence of edge

collapses. This sequence is selected in a purely greedy fashion; once a contraction is selected, it is never reconsidered. At a high level, the outline of the algorithm is as follows:

1. Preprocessing.
2. Select a set of candidate edge collapse.
3. Assign a cost of collapse to each candidate.
4. Place all candidates in a heap keyed on cost with the minimum cost edge at the top.
5. Repeat until only one face is left or the desired face number is reached:
    a. Remove the selected vertex u of the least cost from the heap;
    b. Remove the faces adjacent to vertices u and v;
    c. Update costs of all candidate vertices involving u.

Most of the various algorithms based on iterative contraction have this basic structure. Each pair of vertices being considered for contraction is assigned a "cost". The way in which this cost is determined is the primary differentiating factor among algorithms of this type. Generally, this cost of the collapse is meant to reflect the amount of error introduced into the approximation by the collapse in question. At each iteration, the lowest cost pair is contracted. In the next section, we will discuss how our algorithm calculates the contraction costs.

3.2 Error Metrics for Edge Collapse

In this section, we present the error metrics for the edge collapse including the placement of the new vertex and the cost function design that guides the edge collapse. A preprocessing step is also introduced.

3.2.1 Preprocessing

To facilitate future processing, our first step is to create proper data structures, which is called preprocessing as shown in Figure1.1. The data structures are a set of links within a 1-ring neighbourhood of a given vertex that describes the geometric and topological information. A triangular mesh M consists of a set of vertices $V = \{v_i\}_i \subset R^3$, and a set of faces $F = \{fk = \Delta(vk1, vk2, vk3)\}k$. Let $v \in V$ be a vertex of a triangular mesh M, the 1-ring neighbourhood of v is shown in Figure 3.3. The links are constructed as linked lists as follows.

Links for vertex: Given a vertex v, there are three links associated with it. The vertex-face link VF_{link} is the list of faces around v, $VF_{link} = (f_1, f_2, \ldots, f_i, \ldots f_n)$. The vertex-edge link VE_{link} is the list of edges around vertex v, $VE_{link} = (e_1, e_2, \ldots, e_i, \ldots e_n)$; The vertex-vertex link VV_{link} is the list of vertices around v, $VV_{link} = (v_1, v_2, \ldots, v_i, \ldots v_n)$.

Links for face: Given a face F, there are two links associated with it. The face vertex link FV_{link} is the list of vertices of F. As each triangular face has three vertices, which are directly from the raw data, $FV_{link} = (v_1, v_2, v_3)$. The face-edge link FE_{link} is the list of edges of F. As each face has three edges, $FE_{link} = (e_1, e_2, e_3)$.

Links for edge: Given an edge E, there are two links associated with it. The edge vertex link EV_{link} is the list of vertices of E. As each edge has two vertices, $EV_{link} = (v_1,$

v_2). The edge face link EF_{link} is the list of faces that share E. Normally, if the edge is inside the mesh, there are two faces sharing the edge; if the edge is a boundary edge, then there is only one face associated with it, $EF_{link} = (f_1, [f_2])$.

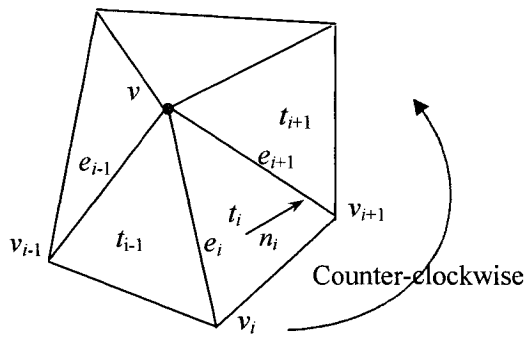


Figure 3.3 A vertex v and the related attributes for its 1-ring neighbourhood.

Note that all the links are sorted in a counter-clockwise direction. Other than the links, we also define a face normal n_i of face $f(v, v_i, v_{i+1})$ as:

$$n_i = (e_i \times e_{i+1}) / \| e_i \times e_{i+1} \| \tag{3.3}$$

3.2.2 Vertex Placement

When considering the collapse of an edge (u, v) (Figure 3.2), we need an approach of choosing the target position v'. Normally, there are three ways for vertex placement [HDD+93, GH97]. The choice between them must be made with the intended application in mind; one must trade efficiency against approximation quality.

To produce the best approximation, one can use the *optimal placement* method, which is to find an optimal position inside vertices u and v's neighbourhood area in such a way that this makes the cost function minimal. Although this way can create an optimal approximation, the computation cost is very high and the extra space is needed to store

the new produced vertices. The second method is *halfway placement*, which is to place the new vertex just at the middle point of a collapsed edge (u, v). Certainly, it is easy to find the new vertex v', but it still needs space to store it.

The third method is the simplest method, which is called *subset placement*. We simply select one of the endpoints as the target position. In other words, we will contract one endpoint to the other. This approach cannot produce optimal simplification results, but it is very simple and efficient. If we further investigate the subset placement method, we could find that it is a special case of *Vertex Decimation*. It can be considered as a special re-triangulation after a vertex is deleted. Figure 3.4 shows all the possible re-triangulations after the elimination of a vertex. The six re-triangulations in the middle row correspond to all the possible edge collapse operations, which eliminate the same central vertex and contract it to its six neighbour vertices. The first and the last rows show the re-triangulation that cannot be expressed by edge collapse.

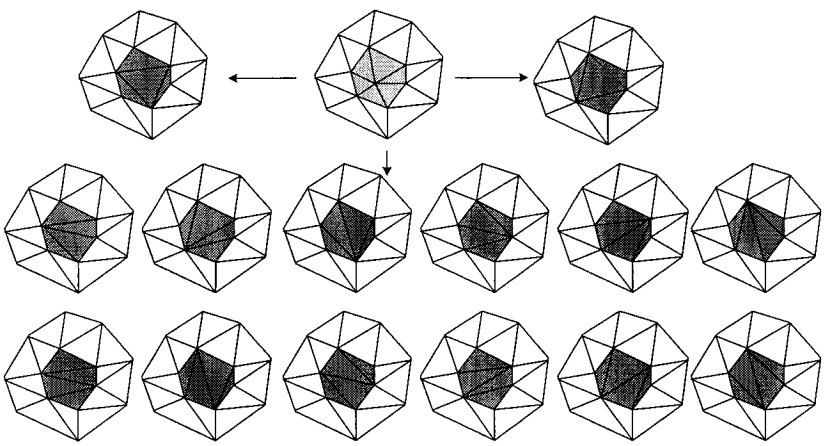


Figure 3.4 All the possible re-triangulations after removing the vertex in the center; the middle row shows all the possible edge collapse operations.

Due to its simpleness and efficiency, we choose the third method for edge collapse in our algorithm. This method is also called "half edge collapse".

3.2.3 Cost Function Design

The "half edge collapse" operation is contracting a vertex to one of its neighbour vertices. This is simple and does not request extra space to store the resulting vertex. The key thing of the simplification algorithm is the selection of which vertex to move to. Here, we define a cost function that guides the edge collapses in the way that can best keep the geometric topology. In this cost function we take into account the factors of edge curvature and face area change. During the cost computation, we also consider some special cases, such as flipping error and collinear vertices, which degenerate local topology significantly. A very high cost value will be assigned to these cases to prevent the degeneration.

3.2.3.1 Edge Curvature

Curvature for 3D graph is a very important criterion to describe how fast the surface is bending locally. From the mathematical point of view, the curvature estimation of a vertex is for analyzing smooth surface's behaviour, which involves sophistic mathematic expression and computation. Even the discrete simulation is not trivial. In our algorithm, the goal is to find a proper edge to contract. We define the edge importance by an edge curvature.

Assertion: An edge is sharper if the angle between its two adjacent surface normals is wider.

This can be shown in Figure 3.5, in which n_1 and n_2 are the normals of the two adjacent surfaces of edge uv, and γ is the angle between n_1 and n_2. Normally speaking, the wider angle between two faces, the more important is the shared edge. However, only γ cannot capture the features of the edge uv. Different edges with different shapes of the adjacent faces may have the same value of γ. We need a weight to take into account other geometric factors of the edge. In Figure 3.5, l, a and b are the lengths of the edges uv and its two adjacent edges up_1, and up_2; h_1 and h_2 are the Euclidean distances from the vertices p_1 and p_2 to the edge uv respectively; θ and φ are the angles between uv and its adjacent edges. Suppose the angle γ is fixed. It is straightforward that if the ratio $l/\sum h_i$ is higher then the edge uv is more geometrically significant, and vice versa. Moreover, the angle γ is closely proportional to the value (1-cos (γ)), in which cos(γ) can be expressed as the dot product of the normals of the two adjacent faces n_1 and n_2. Therefore, we can give a definition of edge curvature as follows.

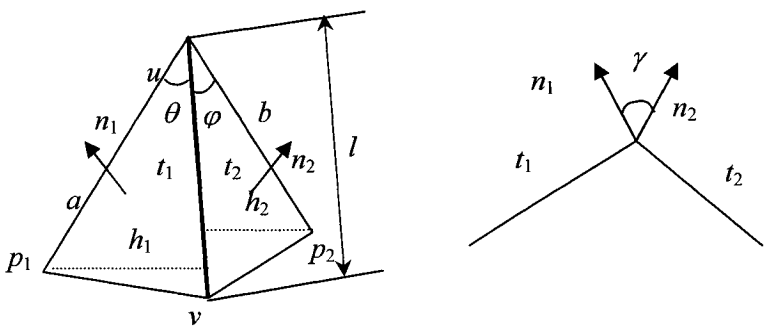


Figure 3.5 Factors for the edge curvature.

Definition 3.1: The curvature of the edge uv is given by:

$$E_C = \alpha \cdot (1 - \cos\gamma) = \left(\frac{l}{h_1 + h_2}\right)(1 - n_1 \cdot n_2) = \frac{l(1 - n_1 \cdot n_2)}{h_1 + h_2} \tag{3.4}$$

where $l = \|uv\|$, $h_1 = a\sin\theta = a\dfrac{\|\overrightarrow{up_1} \times \overrightarrow{uv}\|}{al} = \dfrac{\|\overrightarrow{up_1} \times \overrightarrow{uv}\|}{l}$, and similarly $h_2 = b\sin\varphi = b\dfrac{\|\overrightarrow{up_2} \times \overrightarrow{uv}\|}{bl} = \dfrac{\|\overrightarrow{up_2} \times \overrightarrow{uv}\|}{l}$. Actually, these norms of the vector products can be reused from the calculation of n_1 and n_2.

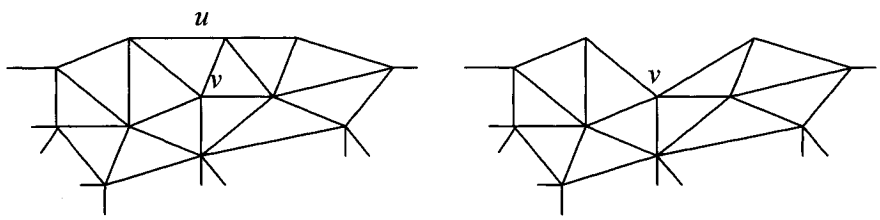


Figure 3.6 Collapse $u \rightarrow v$ causes mesh degradation.

Given the vertex coordinates, edge curvature can be computed efficiently. It is simple but also effective. Normally the edge curvature metric can help us to keep the sharp edges and contract the flat ones first. However, this metric itself is a singular value without considering the direction. In other words, for edge collapses from vertex u to v and from v to u the metrics are the same. As shown in Figure 3.6, this may cause mesh degradation when one of the end points of the considered edge is a boundary vertex and the collapse direction is from the boundary vertex to the inner vertex. Another failure scenario is illustrated in Figure 3.7. As the vertices s and t move closer to each other, the triangles Δusv and Δwsu become more coplanar, as well as the triangles Δtuv and Δtwu. Obviously, the curvature values of the edges us and ut are much less than the edges uv and uw. If we just follow the curvature metric, we shall move u to s. But as Figure 3.7 indicates, collapse $u \rightarrow s$ will cause more significant shape degeneration than collapse u

$\rightarrow v$. The better choice should be moving u to v. Therefore, another metric is needed to complement the edge curvature metric.

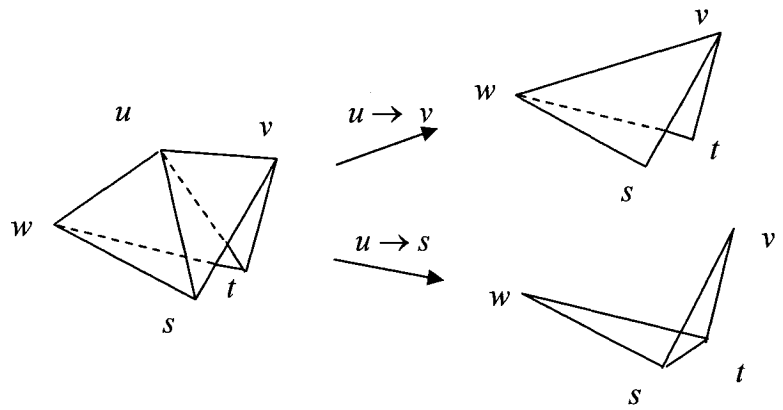


Figure 3.7 Comparison between two different edge collapses.

3.2.3.2 Change of Face Areas Around the Vertex

When collapsing an edge uv from the vertex u to the vertex v, we not only remove the faces adjacent to the edge uv, but also update the remaining faces that involve u. Thus, we should consider the whole neighbourhood of u during the edge collapse. Obviously, the change of the areas of the affected faces is a main consequence of an edge collapse operation. In the shadow area of Figure 3.8, the solid and dash lines denote the faces before and after the edge collapse. Apparently if the area change rate is higher, the geometrical topology will be affected deeper. The area change rate is the area difference after the edge collapse over the original area sum around the vertex u. Different triangle sizes may produce the same area change rate. Hence, we define the second metric that considers both of the area change rate and the triangle size, as written in (3.5).

$$E_A = \frac{\Delta A}{\sum A_{f_i}} \Delta A = \frac{(\Delta A)^2}{\sum A_{f_i}} = \frac{(\sum A_{f_i} - \sum A_{f_j})^2}{\sum A_{f_i}} \tag{3.5}$$

where f_i is the face around vertex u before the edge collapse, and f_j is the newly generated face after the edge collapse.

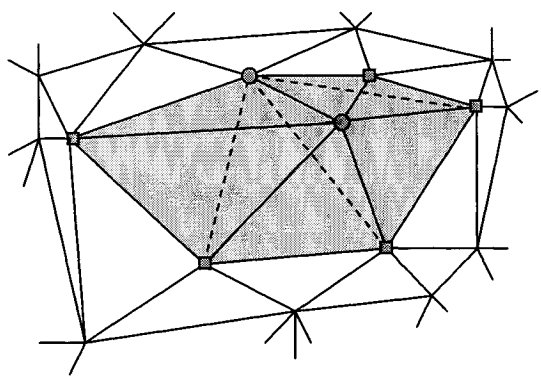


Figure 3.8 Change of face areas after edge collapse.

This area change metric can solve the degeneration problem mentioned above. It can prevent the most edge collapses from the boundary vertex to the inner vertex. Also it is able to guide the edge collapse along $u \rightarrow v$ instead of $u \rightarrow s$ as shown in Figure 3.7. By combining these two metrics together, the cost assigned to an edge collapse is expressed as the following.

$$E = E_C + E_A \qquad (3.6)$$

This cost function can guide edge collapse efficiently. We will show this in the following section. However, some special cases still need to be taken care of.

3.2.3.3 Detecting Collinear Vertices and Flipping Error

Normally, the edge collapse operation guided by a cost function eliminates a selected edge and two triangles that are associated with this edge, and updates other

adjacent triangles at each iteration. The cost function will minimize the local surface topology change to make the simplification smooth. However, there are always exceptional cases that can break this rule or drastically change the local topology. There are two common failure scenarios, i.e., collinear vertices and flipping error.

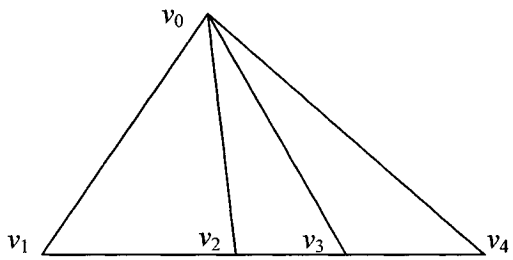


Figure 3.9 Collinear vertices.

It is possible that four or more vertices are on the same line and they share a same adjacent vertex. Figure 3.9 depicts this case, vertices v_1, v_2, v_3 and v_4 are collinear vertices, v_0 is their common adjacent vertex. If the edge collapse operation moves vertex v_0 to any one of these four collinear vertices, all of these three triangles will be removed. For example, edge v_0v_2 is collapsed to vertex v_2, then triangles $\Delta v_0v_1v_2$, $\Delta v_0v_2v_3$, and $\Delta v_0v_3v_4$ will be removed. Actually, after edge collapse the triangle $\Delta v_0v_3v_4$ becomes $\Delta v_2v_3v_4$, which is a line instead of a new triangle. To avoid this, we can use the second error metric. When we calculate the areas of the newly generated faces after edge collapse, we check the area values. If any one of the new triangles' area is zero, we assign a very large predefined value to E_A to ensure this edge will not be selected.

Another important and common error condition is face flipping. This happens when the boundary of a given vertex's 1-ring neighbourhood has concave portion. As shown in

Figure 3.10 (a), vertices v_1, v_2 and v_3 are on the boundary of vertex v_0's 1-ring neighbourhood. Within this neighbourhood, the consecutive edges v_1v_2 and v_2v_3 construct a concave outline. Suppose we contract the edge v_0v_1 from v_0 to v_1, the affected triangle $\Delta v_0v_2v_3$ becomes the triangle $\Delta v_1v_2v_3$, the orientations of $\Delta v_0v_2v_3$ and $\Delta v_1v_2v_3$ are almost opposite (see the normal vectors in Figure 3.10 (a)). Thus, we call this flipping face error. A special case of the flipping face error is illustrated in Figure 3.10 (b). In this case, $\Delta v_1v_3v_2$ is an existing triangle; the newly generated triangle $\Delta v_1v_2v_3$ after edge contraction has the same vertices as $\Delta v_1v_3v_2$ except that it has the exact opposite orientation. As shown in the right diagram of Figure 3.10 (b), edge v_1v_3 could be shared more than three triangles. This results in non-manifold mesh, and triangle $\Delta v_1v_3v_2/\Delta v_1v_2v_3$ stands out from the mesh surface, we call this an outstanding face error.

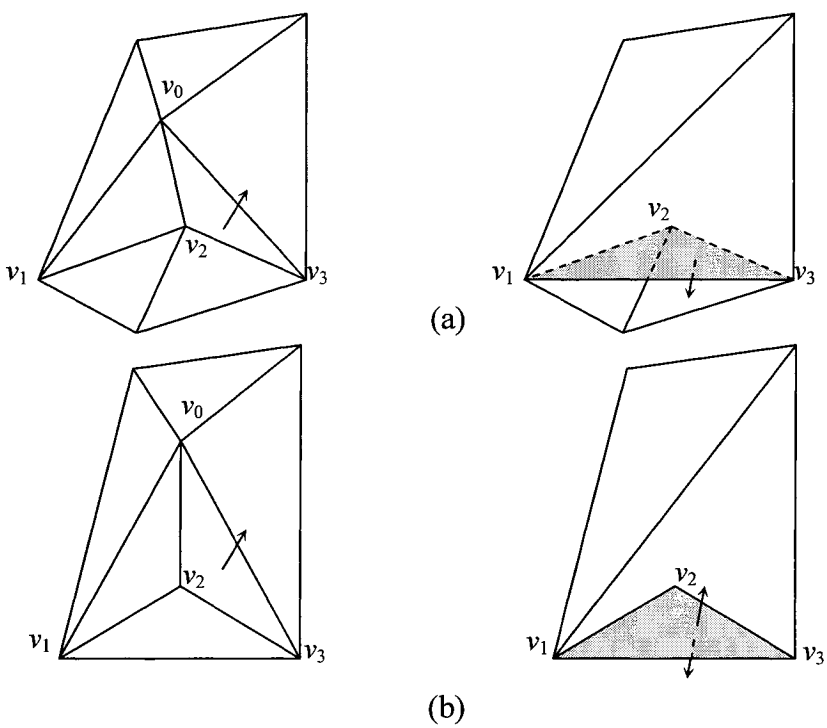


Figure 3.10 Flipping face errors. (a) normal flipping face error; (b) outstanding face error.

To prevent these flipping errors, we need to check the boundary concaveness of the given vertex's 1-ring neighbourhood. We define the convexity and concaveness of two successive boundary edges as follows:

Definition 3.2: Given a vertex v_0, v_1v_2 and v_2v_3 are two successive boundary edges of vertex v_0's 1-ring neighbourhood. Let α and β denote the angles $\angle v_0v_2v_1$ and $\angle v_0v_2v_3$, then the boundary interval $v_1v_2v_3$ is concave if $(\alpha+\beta) > \pi$, otherwise, it is convex.

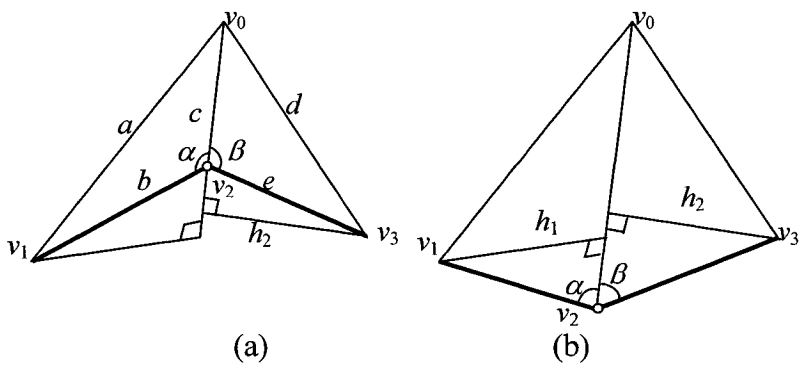


Figure 3.11 Boundary concaveness checking. (a) concave boundary; (b) convex boundary.

This can be shown in Figure 3.11. The boundary interval $v_1v_2v_3$ is concave in Figure 3.11 (a) and convex in Figure 3.11 (b). To determine if an angle is greater than π, we can easily use its sine function to justify: A is an angle, if $\sin(A) < 0$, $A > \pi$, if $\sin(A) > 0$, $A < \pi$. Let a, b, c, d, and e represent the edge lengths of the two triangles $\Delta v_0v_1v_2$ and $\Delta v_1v_2v_3$, h_1 and h_2 are the distances from v_1 to v_0v_2 and v_3 to v_0v_2 respectively. Then, we can have the sine function of $(\alpha+\beta)$ as:

$$\begin{aligned}
&\sin(\alpha+\beta)=\sin\alpha\cos\beta+\cos\alpha\sin\beta \\
&=\frac{h_1}{b}\frac{\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_3}}{ce}+\frac{h_2}{e}\frac{\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_1}}{bc} \\
&=\frac{h_1(\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_3})+h_2(\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_1})}{bce}
\end{aligned} \tag{3.7}$$

Since bce is always positive, we just need to check the sign of the expression $(h_1(\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_3})+h_2(\overrightarrow{v_2v_0}\cdot\overrightarrow{v_2v_1}))$. If it is negative, $(\alpha+\beta)>\pi$, otherwise, $(\alpha+\beta)<\pi$. Also h_1 and h_2 are obtained in the edge curvature estimation step, we can detect this flipping error when we compute the E_C metric. When consider an edge, e.g., v_0v_2, we measure its two associated boundary edges; if they are concave, we assign a very large predefined cost value to v_0v_2's two adjacent edges.

Combining the error detection, we can rewrite the cost function (3.6) as a more comprehensive one:

$$\begin{cases} E=\begin{cases} E_\infty & \textit{if one of the new triangle's area is } 0 \\ E_C+E_A & \textit{otherwise} \end{cases} \\ \textit{Assign } E_\infty \textit{ to adjacent edges if the associated boundary edge is concave} \end{cases} \tag{3.8}$$

where E_∞ is a very large pre-defined value.

3.3 Experimental Results and Comparisons

3.3.1 Simulation Results

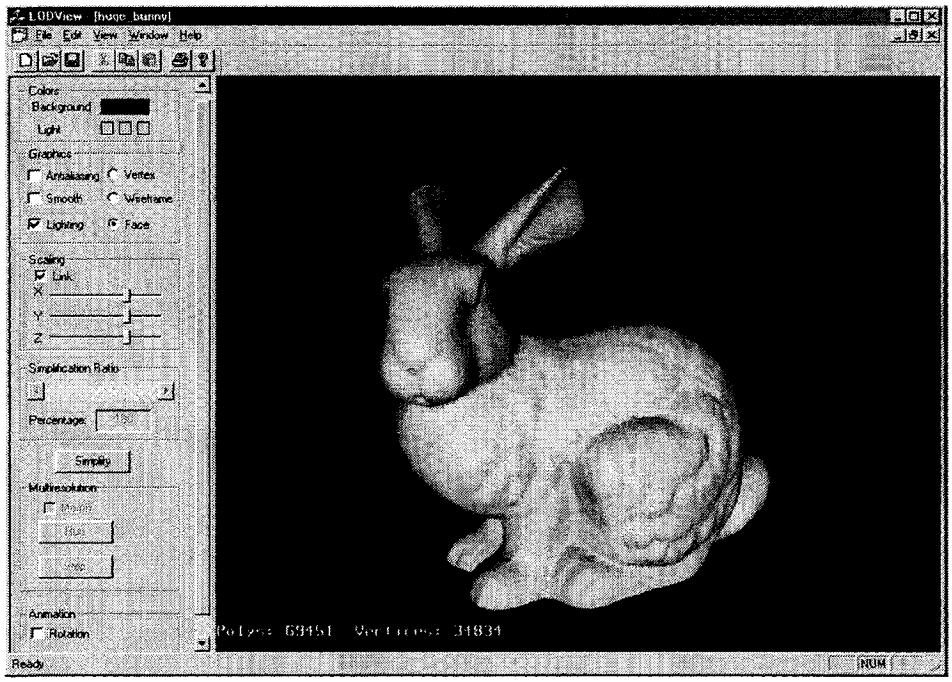
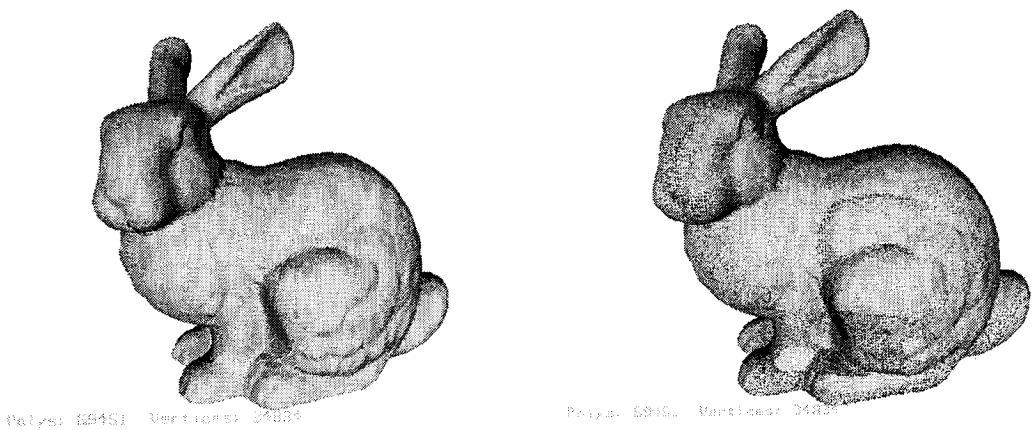
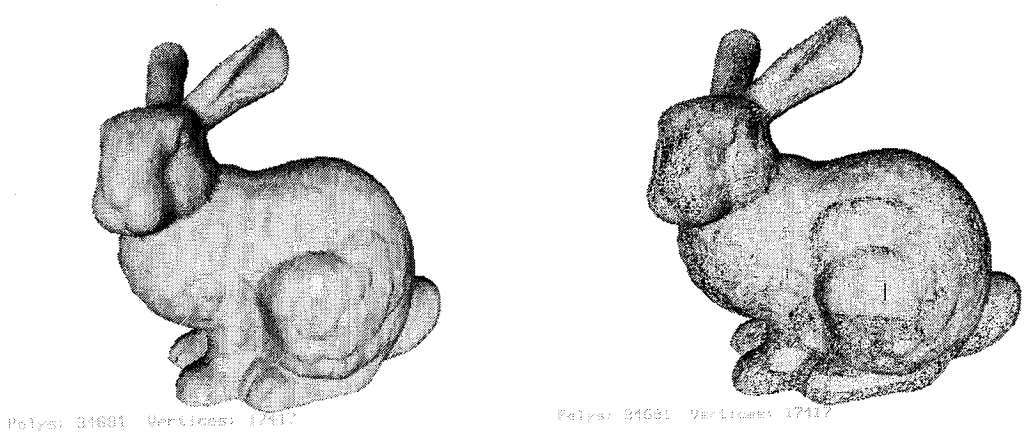


Figure 3.12 User interface for multi-resolution modeling.

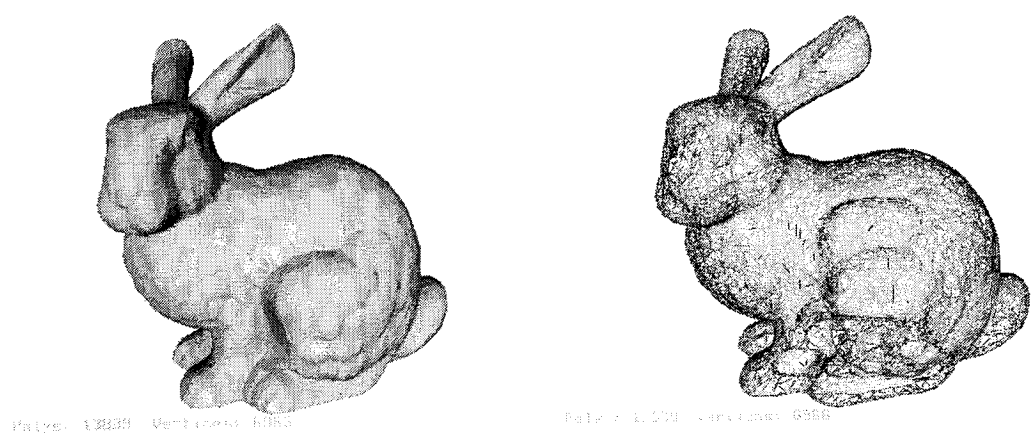
We have simulated the proposed multi-resolution modeling algorithm on an Intel PIII platform under a WIN2000 environment, using Visual C++ with an OpenGL library. The interface of the application is shown in Figure 3.12. A large number of 3D objects have been tested. Figures 3.13 through 3.15 show the original and simplified version of some models. For each object, the resolutions 100%, 50%, 20%, and 10% are displayed. In each figure, the left column is in face format, and the right column is in wireframe format. The numbers of vertices and triangles are listed in Table 3.1.



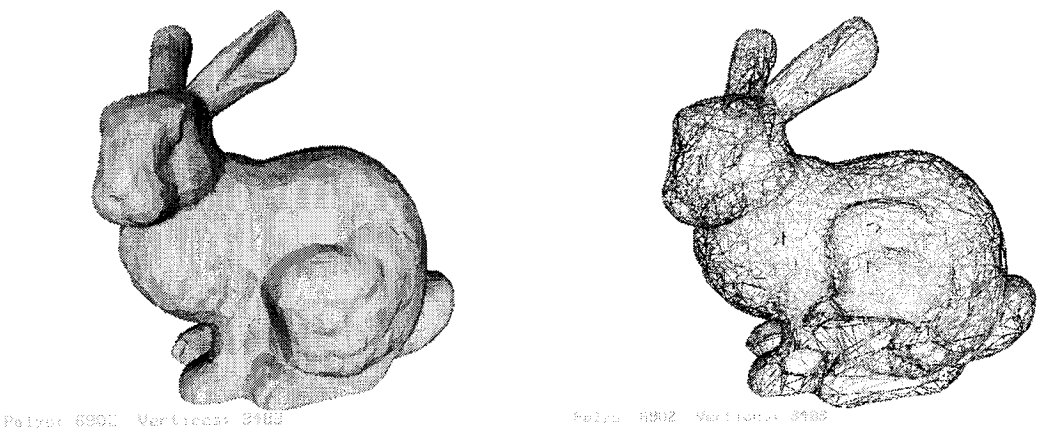
(a)



(b)

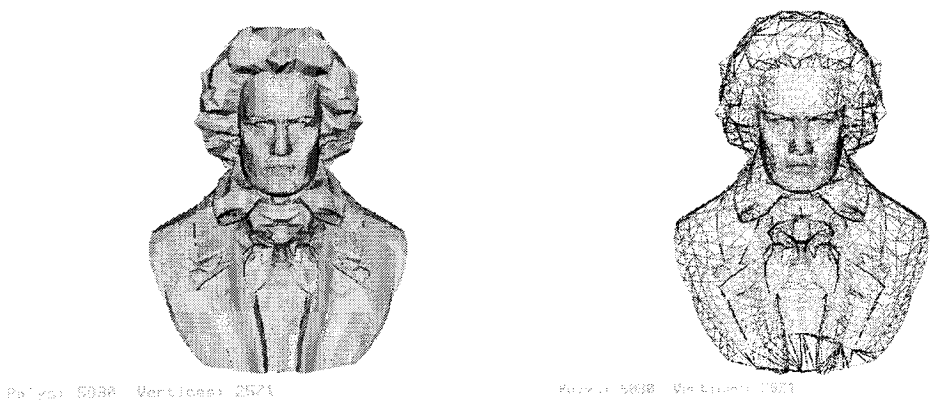


(c)

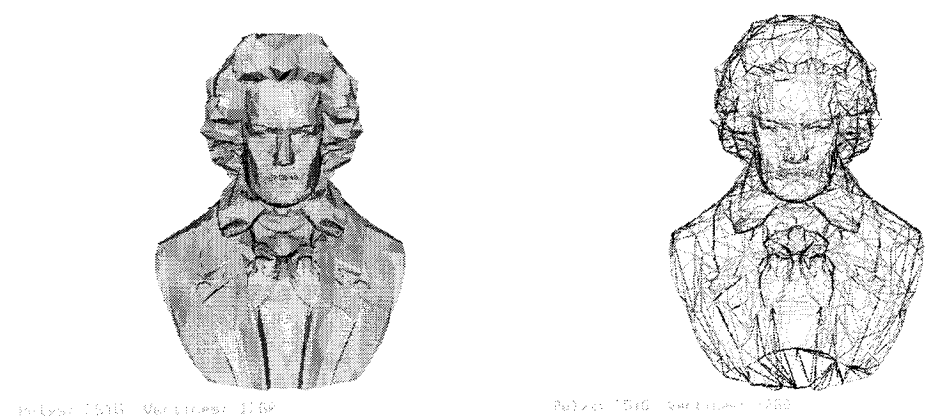


(d)

Figure 3.13 Bunny: (a) 100%, (b) 50%, (c) 20%, (d) 10%.

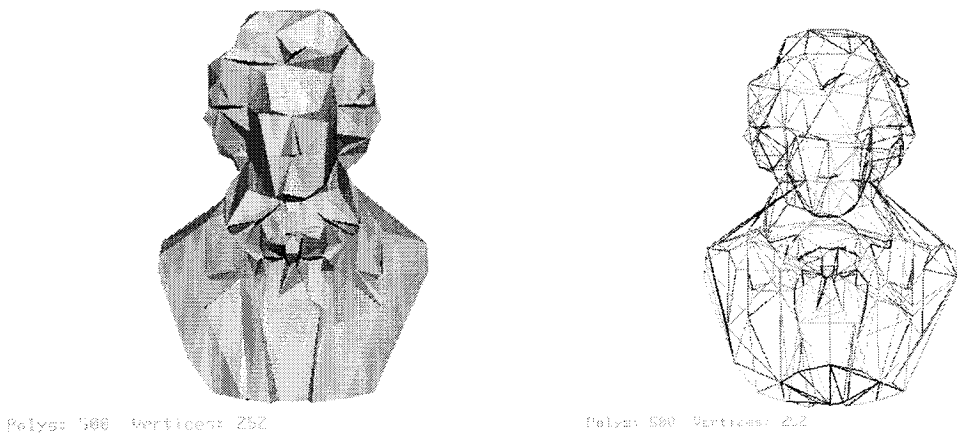


(a)



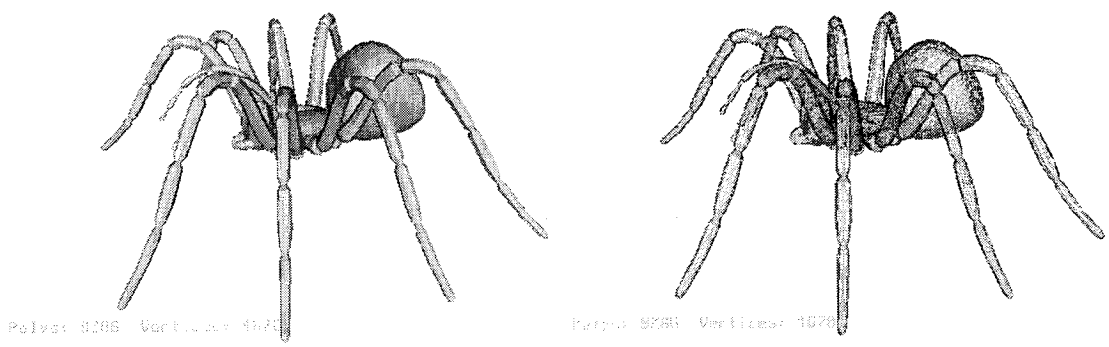
(b)

(c)

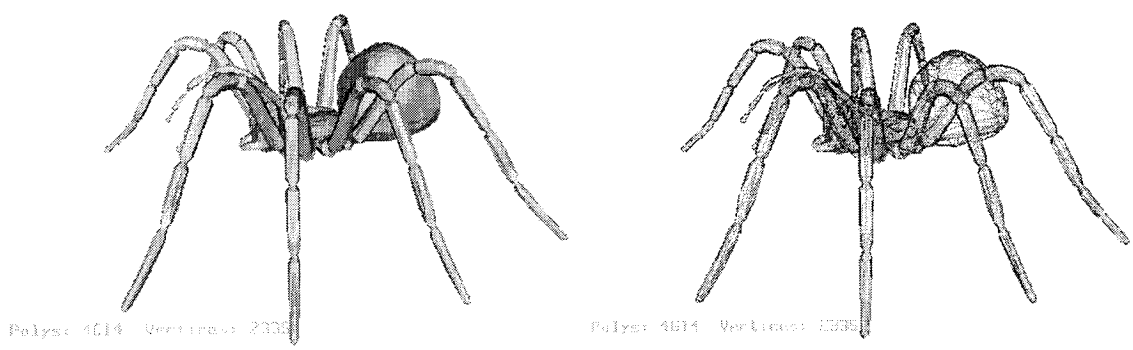


(d)

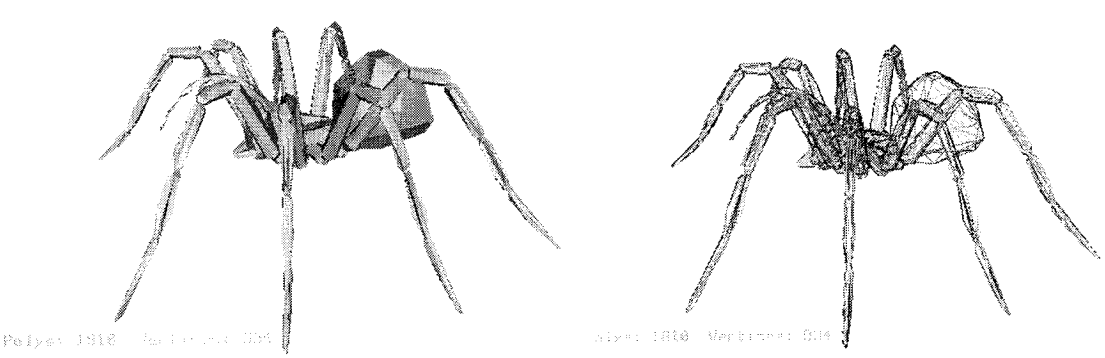
Figure 3.14 Beethoven: (a) 100%, (b) 50%, (c) 20%, (d) 10%.



(a)



(b)



(c)

(d)

Figure 3.15 Spider: (a) 100%, (b) 50%, (c) 20%, (d) 10%.

From the above figures, we can see the proposed multi-resolution modeling algorithm can effectively simplify various kinds of models. The important geometric features are well kept when the triangle number is10% of the original model.

3.3.2 Comparisons

Execution Time

Lindstrom and Turk [LT98] compared six existing simplification algorithms running on a four-processor, 195 Mhz R10000 Silicon Graphics Onyx machine with 1GB memory. Among these methods, QSlim [GH97] is the fastest one. It is considered the state-of-the-art simplification algorithm. For the typical Stanford bunny model, their results show that QSlim takes 32 seconds, while the other methods take minutes or even close to an hour for mesh optimization [HDD+93]. The polygon reduction (PR) method [Mel98] is a very simple and efficient algorithm too. We will compare QSlim, PR and our proposed edge and face area (EA) based algorithm on an Intel PIII 733 Mhz machine. The

speed performance of these three methods on various models are listed in Table 3.1, where N_v and N_t are the numbers of vertices and triangles respectively.

Table 3.1 Execution times for different simplification approaches (in second).

Model	N_v	N_t	QSlim	PR	EA
Horse	48,465	96,966	12.125	11.487	14.688
Bunny	34,834	69,451	7.547	6.953	9.525
Dodge	8,477	16,646	1.891	1.587	2.657
Spider	4,670	9,286	0.992	0.886	1.448
Street lamp	4,440	8,828	0.921	0.785	1.259
Cow	2,903	5,804	0.638	0.565	0.893
Beethoven	2,521	5,030	0.562	0.519	0.786

From the table we can see that the PR method is the fastest, QSlim is the second, our EA algorithm is a little bit slower and but close to them. However, in terms of quality, our algorithm is better than the other two as described below.

Quality Comparison

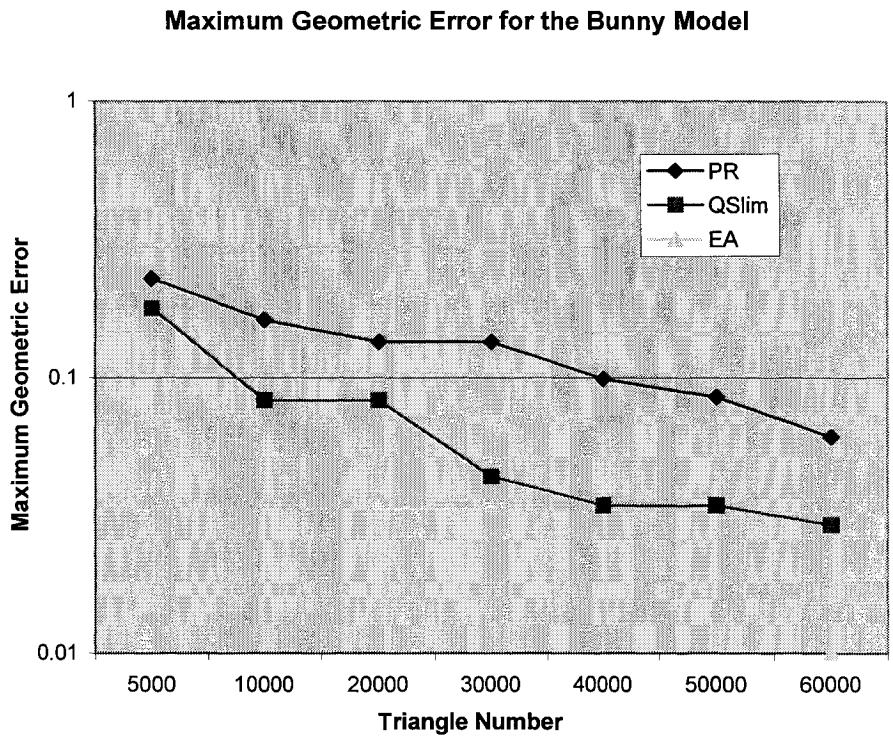


Figure 3.16 Maximum geometric error for the Bunny model.

In order to measure the surface deviation, we have chosen to use the well-known Metro geometric comparison tool [CRS98]. Metro accepts two polygonal models, a simplified and an original model, and computes the maximum and mean geometric errors of the simplified model with respect to the original. The maximum geometric error, which is actually the Hausdorff distance introduced in chapter two, is commonly used for geometric comparison. Figure 3.16 illustrates the maximum geometric errors created by QSlim, PR, and EA. The chosen model is Stanford bunny. The graph shows that our algorithm performs better than QSlim and PR.

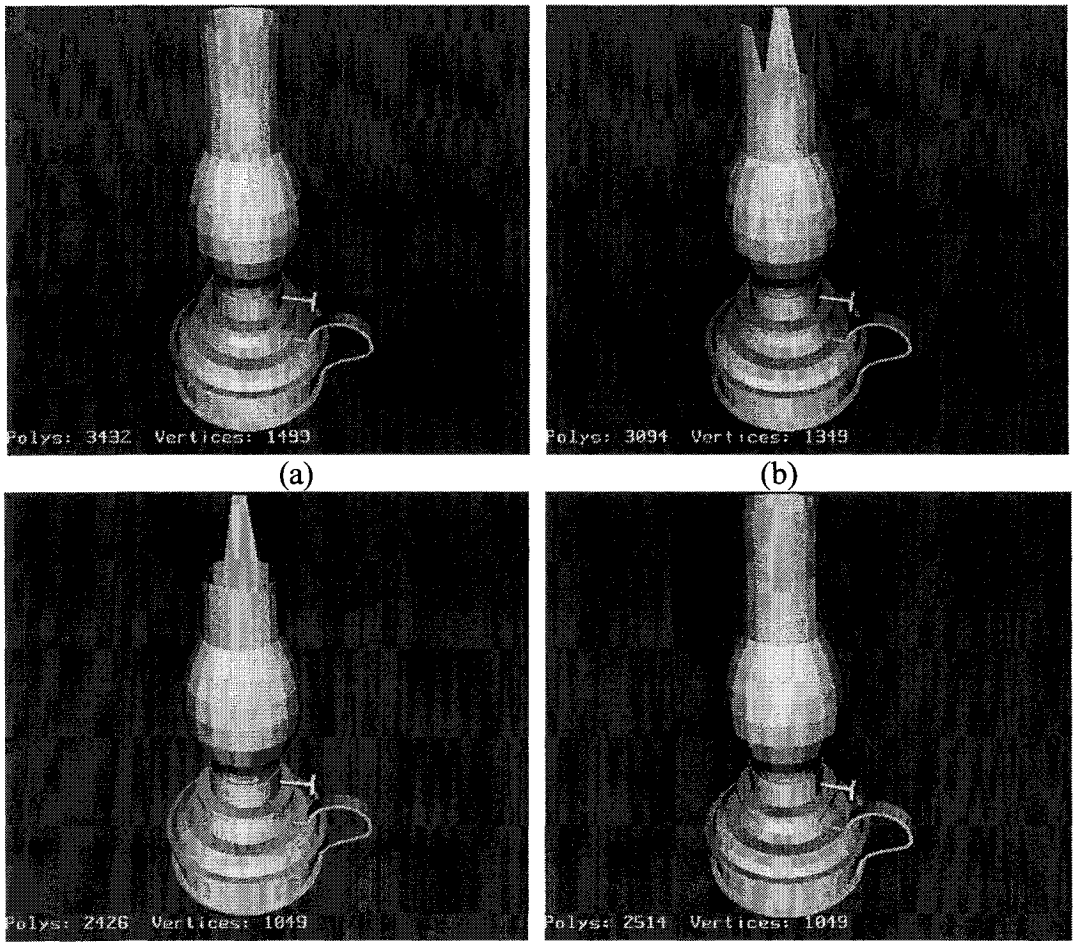


Figure 3.17 Kerolamp model. (a) original model; (b) simplified by PR; (c) simplified by QSlim; (d) simplified by our EA method.

Our algorithm can simplify a model to an extremely low resolution while keeping its semantic meaning. It not only works for the regular meshes as shown in Figures 3.13 through 3.15, but also deals with the special cases. The cost function in our method uses the area metric to prevent the boundary moving inwards, and detects different error conditions to avoid the corresponding failure scenarios. Figures 3.17 and 3.18 depict two examples. The simplified models produced by all these three methods have the same number of vertices. For the Kerolamp model in Figure 3.17, both QSlim and PR remove the boundary triangles during the early simplification stage. For the Box model in Figure

3.18, PR algorithm drills holes on the handle and generate outstanding faces inside the box; QSlim, on the other hand, is better than PR, but still creates obvious deviations on the box handle. However, our algorithm works very well for both of these cases.

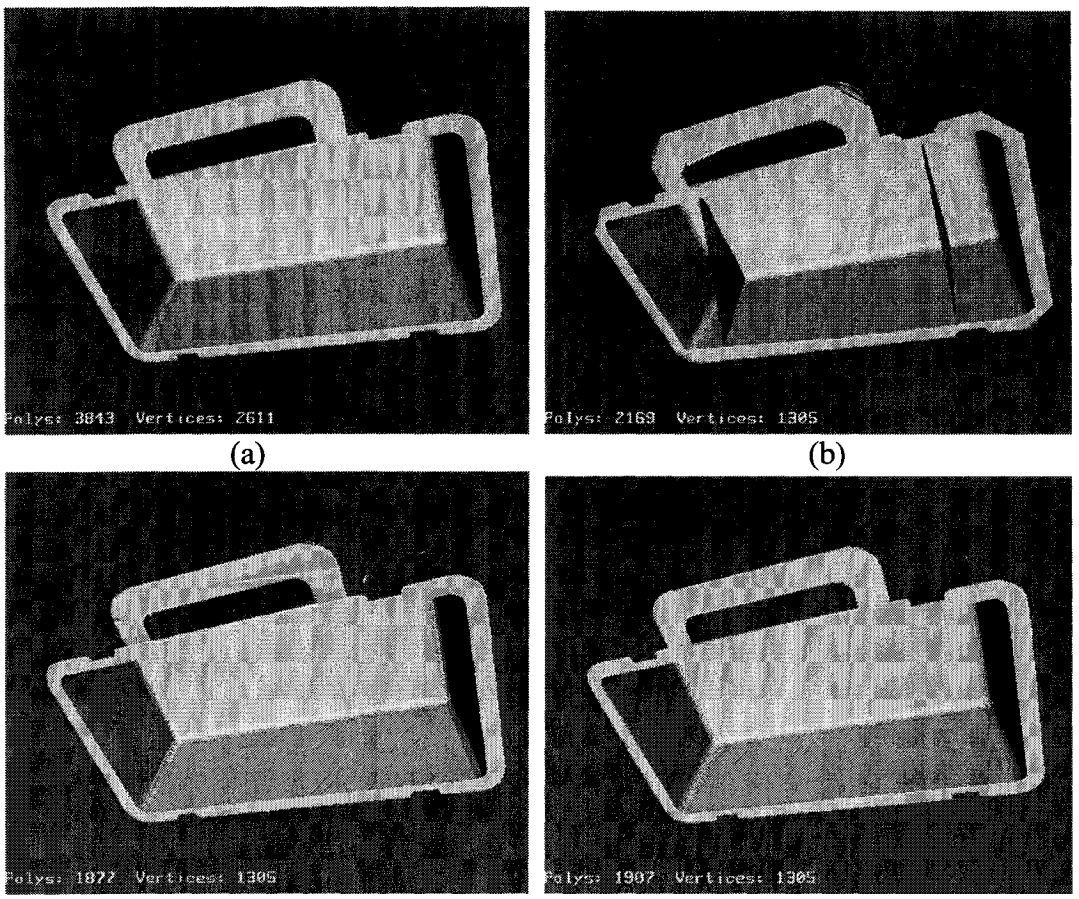


Figure 3.18 Box model. (a) original model; (b) simplified by PR; (c) simplified by QSlim; (d) simplified by our EA method.

CHAPTER 4

4 An Efficient and Robust Neighbourhood-based Mesh Segmentation

As we already know, a 3D triangular mesh can be described as a set of faces that share the edges, or a set of vertices that are connected to the neighbour vertices by the edges. This representation is among the most common data structures used for representing complex 3D objects due to the recent advances in 3D scanning and acquisition technology. In many cases, such meshes have no explicit high-level structure. In general, high-level abstraction is useful for many computer vision tasks such as model fitting, object retrieval and indexing, feature modeling, and scene understanding, etc. One way to impose such high-level description is through mesh segmentation. Segmentation means breaking down an existing structure into meaningful connected sub-components. It is more common in the image processing area, and has been recently introduced into the 3D mesh area. In Chapter 2, we have reviewed various 3D mesh segmentation algorithms. Among them, the watershed based algorithm becomes of more interest. This algorithm, described in [BL79], is a classic one in 2D image segmentation. Mangan and Whitaker [MW99] generalized the watershed method to arbitrary meshes by using the discrete curvature at each mesh vertex as the height field, analogous to the pixel intensity on an image grid that drives the 2D watershed segmentation. The curvature estimation for 3D meshes is not a trivial operation, as it is mathematically defined for a smooth surface only

[MP77]. Thus, most of the existing watershed segmentation algorithms are computationally expensive. Zhang *et al.* [ZPK+02] proposed a simple segmentation algorithm using the Gaussian curvature analysis. It is efficient for certain 3D models. However, similar to most of the existing methods, this algorithm has a drawback that in most cases it cannot successfully process the high-resolution 3D models. As the geometric characteristics of the polygons in such a model, such as normals, curvatures are so close that these algorithms fail to detect some feature parts. Therefore, an efficient and robust algorithm is needed for 3D mesh segmentation.

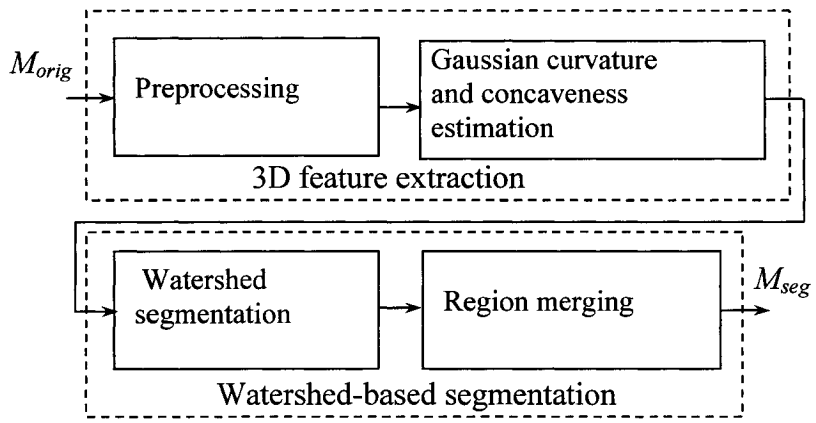


Figure 4.1 Flow chart of proposed 3D mesh segmentation.

In this dissertation, we present an efficient 3D mesh watershed segmentation algorithm using Gaussian curvature and concaveness estimation. Gaussian curvature can identify elliptic and hyperbolic behaviors of a 3D polygonal mesh. However, it cannot detect whether the corner vertex is convex or concave. Thus, we propose a concaveness detection algorithm to complement the Gaussian curvature. And more importantly, in our algorithm, we enlarge the normal 1-ring neighbourhood to an eXtended Multi-Ring (XMR) neighbourhood during the estimation for each vertex in order to get more accurate

geometric features for high-resolution meshes. After feature extraction using Gaussian curvature and concaveness estimation, we develop a watershed-based segmentation algorithm with an efficient region merging scheme. The flow chart of the whole procedure is depicted in Figure 4.1.

4.1 3D Feature Extraction using Gaussian Curvature and Concaveness Estimation

4.1.1 Discrete Gaussian Curvature

As shown in Figure 1.1, a preprocessing is necessary before any further operations. We have described the preprocessing step in section 3.2.1. Several links will be created within the 1-ring neighbourhood of a given vertex. For mesh segmentation, other than the links, we also need the angles between two edges. As illustrated in Figure 4.2, we define the angle between two successive edges $\alpha_i = \angle(e_i, e_{i+1})$.

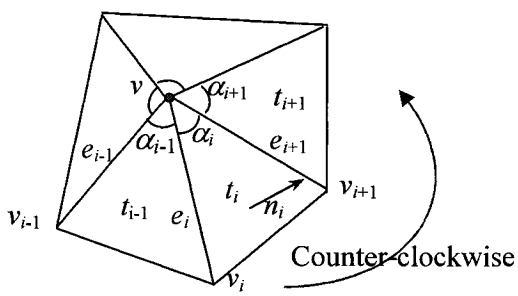


Figure 4.2 1-ring neighbourhood of vertex *v* revisited.

Curvature estimation is a fundamental tool for analyzing and describing a surface's behavior [Gra93, MP77]. For each point on a surface, we can define its curvature that tells us how fast the surface is bending locally. Consider the tangent plane at the point *p*

on the surface. The surface normal is defined as the normal to the tangent plane. Consider the planes passing through the line supporting the normal. These planes intersect the surface in open curves in an open neighbourhood of p. Among these curves, the two curves with maximum and minimal curvatures are of most importance. These two curves with the sign are called principal curvatures at p. The larger (in absolute value) of the two, say k_1, is called the maximum curvature and the smaller of the two, say k_2, is called the minimal curvature at p. The Gaussian curvature at p is defined as $K = k_1 \cdot k_2$. The arithmetic mean of k_1 and k_2 is called the mean curvature H at p.

Definition 4.1: Given a surface S, a point p on S belongs to the following categories [MP77]:

(1) Elliptic if $K > 0$

(2) Hyperbolic if $K < 0$

(3) Parabolic if $K = 0$

From this definition, knowing the Gaussian curvatures of the points on a surface, we can determine the behavior of the points, whether they are elliptic, hyperbolic or parabolic. A 3D mesh is composed of distinct objects or components. The points on each individual component have elliptic or parabolic behavior. And they are connected by the boundaries that have hyperbolic behavior. Therefore, we can divide the surface into disjunctive regions by detecting the boundaries with hyperbolic behavior ($K < 0$). For example, a bolt is composed of two parts, a long cylinder bolt body and a short cylinder bolt base. The points on the two cylinders are elliptic or parabolic. The points on the

connection part between two cylinders are hyperbolic. We can divide the bolt into two cylinders that share the hyperbolic boundary.

From a theoretical point of view, the polygonal faces are flat and the curvature is not properly defined along the edges and at the vertices because the surface is not C^2-differientiable there. Considering a polygonal surface as a piecewise linear approximation of an unknown smooth surface, one can estimate the curvatures of that unknown surface using only the information that is given by the polygonal surface itself. We are particularly interested in computing the discrete Gaussian curvature K. Given a vertex v on a polygonal mesh, the integral discrete Gaussian curvature $\overline{K(v)}$ with respect to the area $S = S_v$ attributed to v is defined by [Kuh02, MDS+02]

$$\overline{K(v)} = \int_S K(v) = 2\pi - \sum_{i=1}^{n} \alpha_i \qquad (4.1)$$

where α_i denotes the angle at a vertex v (see Figure 4.1).

To derive the discrete Gaussian curvature at the vertex v from the integral value, we assume the curvature to be uniformly distributed around the vertex and simply normalized by the area:

$$K(v) = \overline{K(v)} / S = \frac{3(2\pi - \sum_{i=1}^{n} \alpha_i)}{\sum_{i=1}^{n} A_i} \qquad (4.2)$$

where A_i is the area of the adjacent face around a vertex v.

Therefore, for a vertex v, we can easily determine if the vertex is hyperbolic or not by computing $K(v)$ according to (4.2). As what we want to know is the sign of $K(v)$, we can simply use $\overline{K(v)}$ as $K(v)$. This method is simple and efficient for 3D mesh segmentation. Zhang [ZPK+02] proposed a similar idea. However, hyperbolic behavior only cannot partition a model into meaningful parts successfully when there are concave corners on the polygonal mesh. This can be illustrated in Figure 4.3, where three distinct regions R_1, R_2 and R_3 share a concave corner C, which possesses elliptic behavior ($K > 0$). Therefore the boundary lines will be broken at C and consequently these three regions will become one region R'. Zhang did not actively address this issue. In [ZPK+02], the isolated vertices were removed by replacing their labels with their neighbour vertices' labels. If the broken corner is just one single vertex, this method can fix up that corner. However, if the faces are densely distributed on a mesh, the concave corner area is not just a single vertex. Thus, the corner vertices are not isolated. They will be further merged together and break the segmentation integrity.

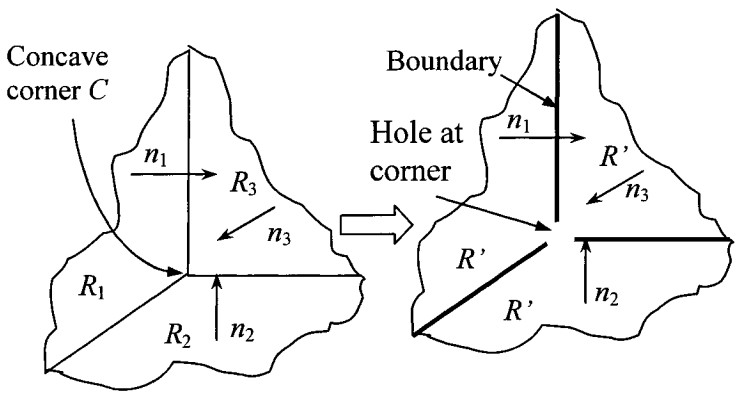


Figure 4.3 Concave corner breaks the segmentation integrity.

4.1.2 Concaveness Estimation

To repair the boundary hole at the concave corners, we will estimate concaveness and convexity of the vertices on a mesh. If a vertex is concave, then it will join the boundary section for future segmentation. In this section, we propose an algorithm to detect the concaveness and convexity of a vertex.

In [SZL92], given a vertex v and its 1-ring neighbourhood, the average plane P of v is defined by the normal vector N_v of v and a center point v_c:

$$N_v = \frac{\sum_{i=1}^{n} n_i A_i}{\sum_{i=1}^{n} A_i} \tag{4.4}$$

$$v_c = \frac{\sum_{i=1}^{n} v_i A_i}{\sum_{i=1}^{n} A_i} \tag{4.5}$$

and the distance d from v to P is defined as:

$$d = \left| \overrightarrow{v_c v} \cdot N_v \right| \tag{4.6}$$

where A_i and n_i are the area and the normal of the adjacent face around v respectively, and $\overrightarrow{v_c v}$ denotes the vector from v_c to v.

Instead of using this absolute norm as the distance, we define the signed distance from v to P:

$$d_s = \overrightarrow{v_c v} \cdot N_v \tag{4.7}$$

Based on the signed distance we can define the concaveness and convexity of a vertex v as follows.

Definition 4.2: Given a vertex v, the concaveness/convexity of v is defined by a Boolean operator C_v:

$$C_v = \begin{cases} concave & if\ d_s < 0 \\ convex & if\ d_s \geq 0 \end{cases} \tag{4.8}$$

This can be shown in Figure 4.4. If a vertex v is concave, the vectors $\overrightarrow{v_c v}$ and N_v are in roughly opposite directions, whereas if v is convex, these vectors are in roughly the same direction. With the Gaussian curvature and concaveness estimation obtained, we define the metrics for boundary detection as follows.

Definition 4.3: A vertex v is a boundary vertex if $K_v < 0$ or v is concave; otherwise it is an inner region vertex.

By this definition, both the hyperbolic and the concave vertices are set as the boundary vertices. This way improper segmentation can be prevented from concave corner holes. This can be proved in experimental results.

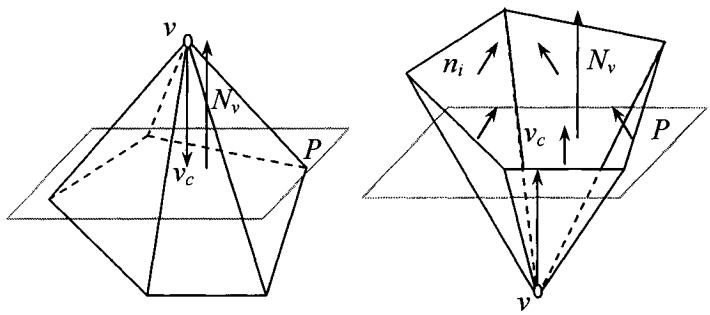


Figure 4.4 Convex vertex vs. concave vertex.

4.1.3 Robust Feature Extraction based on eXtended Multi-Ring (XMR) Neighbourhood

Using the Gaussian curvature and concaveness estimation, we can extract the boundary points that are used for future segmentation. This works well for the relatively small size 3D models. However, if a model has a large number of faces, the faces and vertices are so close that the discrete geometric characteristics are not accurate to reflect the real behavior of the model as the rectangle area shown in Figure 4.5. More specifically, the Gaussian curvature K_v of a vertex v may represent the wrong behavior for the vicinity of v; or the concaveness of v may be convex even if v is in a concave area or vice versa. These vertices can be scattered on the surface, which will result in serious over segmentation; or they can break the boundary lines, which will make the distinct adjacent parts unable to be partitioned correctly. In this section, we propose an algorithm that estimates the Gaussian curvature and concaveness based on an extended neighbourhood of a given vertex.

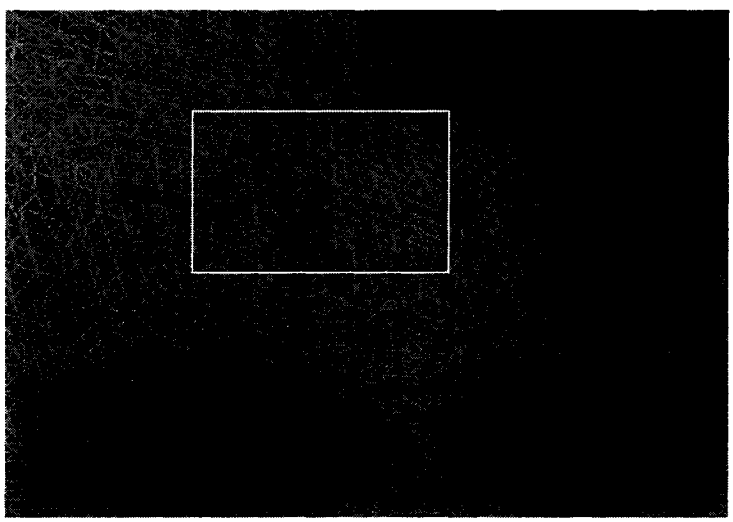
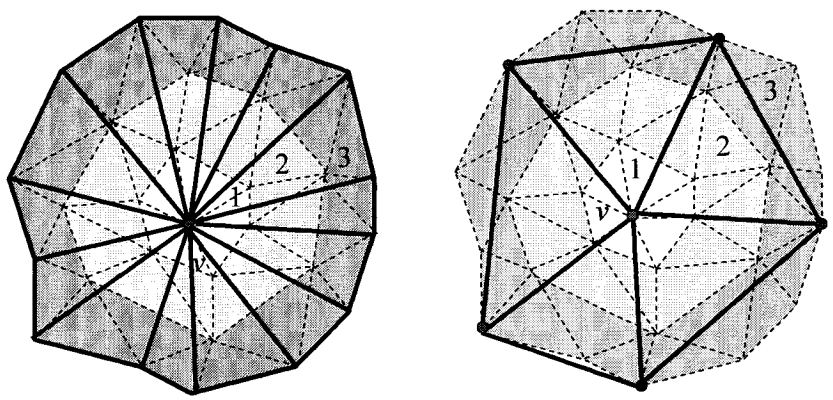


Figure 4.5 Densely distributed 3D Mesh.

Definition 4.4: Given a vertex v, we define the 0-ring adjacent vertices of v: $V_0 = \{v\}$; the 1-ring adjacent vertices of v: $V_1 = Adj(V_0)$, where $Adj(V)$ denotes the adjacent vertices of V. Then we extend the neighbour area of v to multiple rings, and define the L-ring adjacent vertices of v: $V_L = \{v_i \mid v_i \in Adj(V_{L-1})$ and $v_i \notin V_{L-2}\}$ $(L \geq 2)$. After the extension, we connect v and vertices from V_L to create updated adjacent faces F_L of v. V_L and F_L construct the eXtended Multi-Ring (XMR) neighbourhood of v at level L.

This definition can be illustrated in Figure 4.6 (a). The XMR neighbourhood of v at level 3 is composed of the bold lines.



(a) (b)

Figure 4.6 (a) XMR neighbourhood at level 3 of vertex v; (b) simplified XMR neighbourhood of vertex v.

Most of the existing 3D modeling algorithms are based on 1-ring neighbourhood, as we described in the above sections. In this dissertation, we use the XMR neighbourhood method to enlarge the neighbourhood area of the given vertex v to catch more relevant information for v. As the XMR neighbourhood covers more area, for the densely

distributed mesh, the geometric behavior of v can be better reflected by this new neighbourhood, i.e., the Gaussian curvature and concaveness estimation of v is more accurate.

As shown in Figure 4.6 (a), the number of the adjacent vertices of v within the XMR neighbourhood is approximately a linear function of the level number L as follows:

$$N_L \approx L * N_1 \qquad (4.9)$$

where N_L and N_1 are the numbers of the adjacent vertices at level L and level 1 respectively.

Obviously the computational expense for the XMR neighbourhood is more expensive than the 1-ring neighbourhood, as we need to construct the XMR neighbourhood and the numbers of the newly updated adjacent vertices and faces of a given vertex are much more than the corresponding numbers in the 1-ring neighbourhood. To simplify the curvature and concaveness estimation after XMR neighbourhood extending, we take the factor-L decimation onto the adjacent vertices, which means we only keep the $(n*L)$th vertex after decimation, where n = 0, 1, 2, ... In Figure 6 (b), the circled vertices are the remaining ones after the factor-3 decimation. The new XMR neighbourhood is then constructed by the remaining adjacent vertices. In an application, the number L can be specified by the user as an input. For a low-resolution model with thousands of faces, L = 1 is enough; for a high-resolution model with a large number of faces, we can get a good segmentation result with L in the range of [2, 5]. The 3D feature extraction algorithm is outlined in the following pseudo code:

Table 4.1 Pseudo code for feature extraction algorithm.

Algorithm 1: 3D feature extraction using XMR neighbourhood.
input: a mesh M, neighbourhood level L
for all vertices, edges and triangles $\in M$
build links
endfor
for all $f_i \in M$
calculate face area and normal vector
endfor
initialize a adjacent vertex vector V_a
for all $v_i \in M$
for level from 0 to L -1
update V_a of v_i
endfor
for all $v_j \in V_a$
form faces in counter-clockwise manner
compute α_i , face area, face normal
endfor
calculate Gaussian curvature K of v_i
calculate signed distance d_s of v_i
endfor

4.2 3D Segmentation Using a Fast Marching Watershed Scheme

4.2.1 Watershed Segmentation Algorithm

Watershed segmentation schemes have been researched for over two decades now, in the field of mathematical morphology, as a primary method for image segmentation. Mangan and Whitaker [MW99] first introduced the watershed method on a polygonal mesh, and it becomes more and more popular in 3D segmentation area [PRF02, RaB02, RHX+02, PKA03].

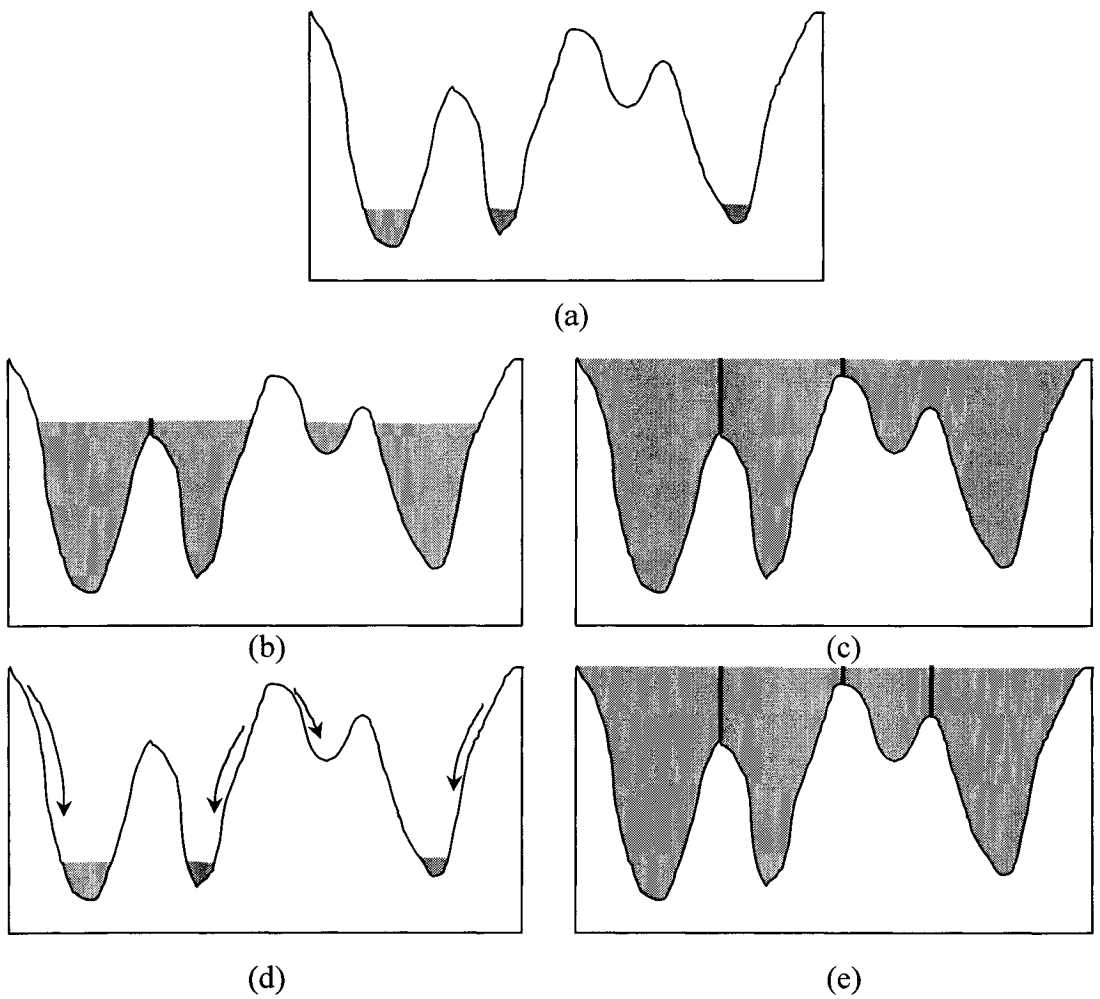


Figure 4.7 Bottom-up and top-down variants of the watershed algorithm. (a) initial local minima; (b) bottom-up in-progress segmentation; (c) bottom-up final segmentation; (d) top-down in-progress segmentation; (e) top-down final segmentation.

When simulate this process for image or 3D mesh segmentation, two different strategies are common for implementation [VS91]. The first strategy is a bottom-up approach where we form catachment basins and flood the data with water. The second strategy is a top-down approach where we descend down to the basins from the slope and ridges of the landscape. These two approaches are shown in Figure 4.7. Both of them

require an initial threshold to create a proper number of local minima as illustrated in Figure 4.7 (a). The number of the local minima is the number of the segmented regions before the further region merging. The non-minima areas are called the plateau. The initial boundaries of these minima are very small, but they expand as the algorithm progresses. For the flooding (bottom-up) approach, when two boundaries come in contacts as in the left side of Figure 4.7 (b), they are not allowed to merge. On the right side of that same figure, notice that a new basin has formed that has no correspondence, yet, to any other basins. As flooding continues in Figure 4.7 (c), this new basin will be merged to one of the original basins. For the top-down approach, the problem is a little bit different. As before we establish the initial local minima set, but instead of flooding, we take each point of the plateau area and flow downward until it encounters a minimum or a point which has already been associated with a minimum. Such cases require special consideration. One example is illustrated in the right side of Figure 4.7 (d), a new local minimum will trap and form a new region that requires merging to an appropriate basin afterwards.

Mangan and Whitaker's approach [MW99] falls into the second category. In this dissertation, we use the first one, i.e., the bottom-up approach. Our algorithm is described as follows:

1. Minima detection. Find the local minima and mark each minimum with a unique label. The rest areas are then the plateaus.
2. Plateau erosion. Erode the plateaus to their neighbour minima starting from the boundaries between the plateaus and the minima.

3. Region merging. Merge the less important regions to their neighbour regions.

These three steps will be described in the following sections respectively.

4.2.2 Minima Detection

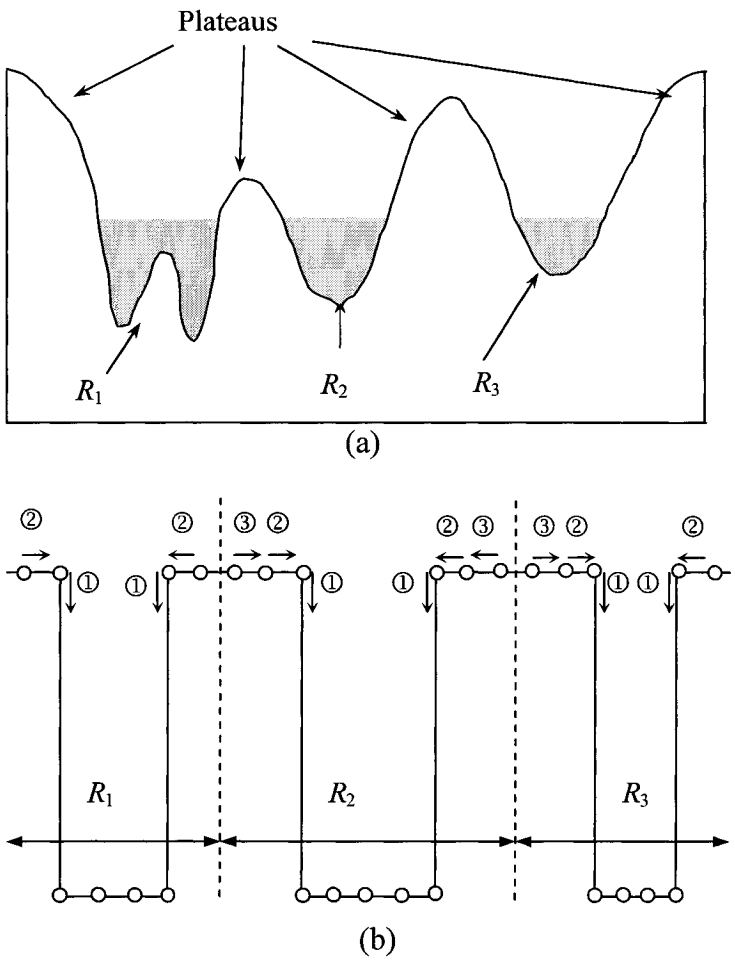


Figure 4.8 (a) Minima thresholding; (b) Plateau erosion.

The watershed algorithm first finds the local minima. Normally the local minima are detected based on a height map at each vertex. We use the feature extraction algorithm proposed in section 4.1 to construct the height map. In section 4.1, we already defined the criteria for boundary detection. Now we simply define the height value of the boundary

vertex as 1, and the height value of the non-boundary vertex as 0. Then the non-boundary regions are the local minima. The boundary regions are the plateaus. All the local minima are labeled with unique numbers. Because of the noise or the complexity of the polygonal surface, the criteria in Definition 4.3 create too many local minima. Thresholding the Gaussian curvature and the signed distance can merge a lot of the small minima and make the segmentation much easier. Figure 4.8(a) shows minima thresholding; two local minima are merged together as region R_1. The thresholding metric is defined as:

Definition 4.5: A vertex v is a plateau vertex if $K_v < \delta_1$ or $d_s < \delta_2$, where $\delta_1 = 0.01 \cdot \min(K_v)$ and $\delta_2 = 0.01 \cdot \min(d_s)$.

where min() is a minimum finding operation.

4.2.3 Plateau Erosion

After minima detection, the basins are the core areas of the final regions. The next step is to segment the plateaus. We use a fast marching method to erode the plateau and convert them to their neighbour basins as illustrated in Figure 4.8(b). Erosion starts at the boundaries between the plateaus and the minima. Step 1 denoted by ① is to erode the level 1 boundary and find the level 2 boundary. And similarly, each following step i will erode the ith level boundary and find the i+1th level boundary until all the plateau vertices are processed. This algorithm is outlined in the following pseudo code:

Table 4.2 Pseudo code for fast marching plateau erosion algorithm

Algorithm 2: Fast marching plateau erosion
input: a mesh with minima and plateaus labeled
define a double-ended queue P_{bound}
for all $v_i \in$ plateaus

```
    if an adjacent vertex v_adj of v_i is a minimum
      P_bound.push_back(v_i)
      label of v_i ← label of v_adj
    endif
  endfor
  while P_bound is not empty
    v = P_bound.pop_front()
    for v_i ∈ adjacent vertex of v
      if v_i is plateau vertex
        P_bound.push_back(v_i)
        label of v_i ← label of v
      endif
    endfor
  endwhile
```

4.2.4 Region Merging

Due to small details and/or noise on the polygonal surface, the watershed transformation always yields an over-segmented result. Thus, region merging is an essential stage for watershed-based segmentation. Region merging is a graph problem with the node being the individual region. The strategy is to define criteria for measuring region saliency and combining insignificant regions to their fitting neighbours. For 3D mesh watershed segmentation, there are a variety of metrics that may define insignificant regions. In [RHX+02], Rettmann *et al.* introduced a region merging metric that is based on the height of the ridge separating two catchment basins. If the relative depth of both catchment basins is less than a threshold, then they are merged. This method uses geodesic depth for each region. It is suitable for sulcal segmentation on the cortical surface, but not for the normal polygonal surface. Two big and distinct adjacent regions can be merged together as the relative depth of those regions may be small. Mangan and Whitaker [MW99] proposed an approach that merges the shallow region with minimum depth to one of its surrounding regions with the lowest boundary point. The watershed

depth is calculated as the difference between the lowest vertex in the region and the lowest vertex along the region's boundary. This metric is fairly reliable. However, it is computationally costly. As reported in [MW99], after a region merging, one must update the boundary vertices information for associated regions, and it is necessary to recalculate the lowest vertex and the lowest boundary vertex. This method uses total curvature to construct the height map and compute the region depth. In our case, we only use 0 and 1 to label the region minima and the boundaries. Thus this method cannot be applied to our case. To compare the computational efficiency, we will simulate this method using the distance from the vertex to its average plane as the height and merge to the same number of the final regions.

In this dissertation, we propose a simple yet efficient region merging algorithm with two criteria. The first criterion is the region size. Apparently, the region with a bigger size is more important than the region with a smaller size. Though area is more accurate to represent the region size, the area-based metric penalizes small areas too aggressively, as described in [MW99]. We simply use the vertex number as the measurement for the region size. The region merging is then a process that iterately combines the merge-needed regions to one of their adjacent regions until the vertex numbers of all the regions are more than a pre-defined threshold. The difficulty is how to determine the destination adjacent region to which the small region will merge. This is addressed by the second criterion, which is the boundary length of the adjacent region. We use boundary length to define the saliency of the neighbour regions of a merge-needed region. As shown in Figure 4.9, for a merge-needed region C, we compute the boundary length of the adjacent regions (A and B) of C. Since B has the longest boundary with C, we merge C to B. Note

that the boundary length is presented by the number of vertices of the boundary interval. The principle of this metric is straightforward. If a negligibly small region is mostly embraced by another adjacent big region, then merging them together is acceptable and will not degrade the overall segmentation quality. After each region merging, we update the neighbour information of associated regions. Because the boundary length is the unique neighbour information and is a scalar value, we do not need to maintain the lists of boundary vertices. The updating can be done by replacing, removing and adding boundary lengths of the merged region. Also there is no need for recalculation of the lowest vertices as in [MW99]. The region merging algorithm can be described in the following pseudo code:

Table 4.3 Pseudo code for region merging algorithm.

Algorithm 3: Region merging algorithm

```
Input: a mesh segmented with minima detection and
plateau erosion
define R[N_r][ N_r], NV[N_r] // N_r is the number of
// regions; NV is the vertex numbers of regions
for  v_i ∈ V
    NV[label of v_i]++
endfor
for e_i ∈ all boundary edges
    get 2 end vertices of e_i
    update R[N_r][ N_r] with row elements are
      boundary lengths, 0 indicates this is not a
      adjacent region
endfor
while(1)
    merging ← 0
    for all i from 0 to N_r
      if NV[i] < threshold
         find target adj region with longest boundary
         change label of region i to target region
         update array R[N_r][ N_r]
         merging ← 1
      endif
```

```
    endfor
    if merging == 0
      break
    endif
endwhile
```

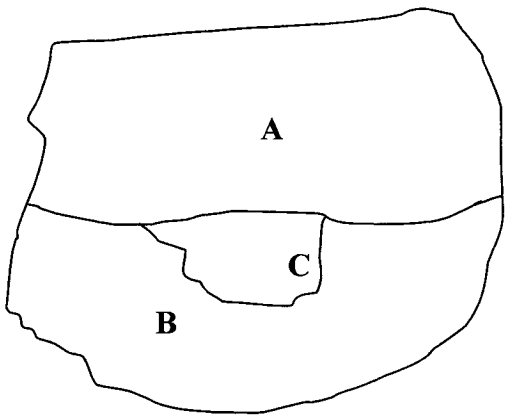


Figure 4.9 A small region will be merged to its adjacent region that has the longest boundary with this region; in this figure, region *C* will be merged to region *B*.

4.2.5 Post-processing — Branch Pruning

After the region merging step, a 3D object has been partitioned into meaningful regions. The vertices and triangles inside a region are assigned with the region number. Within a region, the region numbers of a triangle's three vertices are the same. If a triangle has different vertex region numbers, then this triangle is defined as a boundary triangle. Between each pair of adjacent regions, there is a boundary triangle strip. The boundary triangle strips glue these individual regions together as a whole model. Within a boundary triangle, the edge whose two edge vertices have the same region number is defined as the boundary edge. The boundary edges must construct the closed loops that wrap the segmented regions. However, exceptions always exist. There might be one-edge branches after the plateau erosion as demonstrated in Figure 4.10. In Figure 4.10 (a), edge

uv is a branch sticking out from the boundary of region 1. This results in all the adjacent triangles of vertex *v* are boundary triangles. We call this a boundary strip knob. To thin the knobby boundary strip, the branch pruning is necessary. In the example shown in Figure 4.10, we need to change the region number of the end point of the branch to its neighbour region number, and merge the corresponding triangles (t_1, t_2, t_3, t_4) to its neighbour region. The pseudo code for this branch pruning algorithm is listed in Table 4.4.

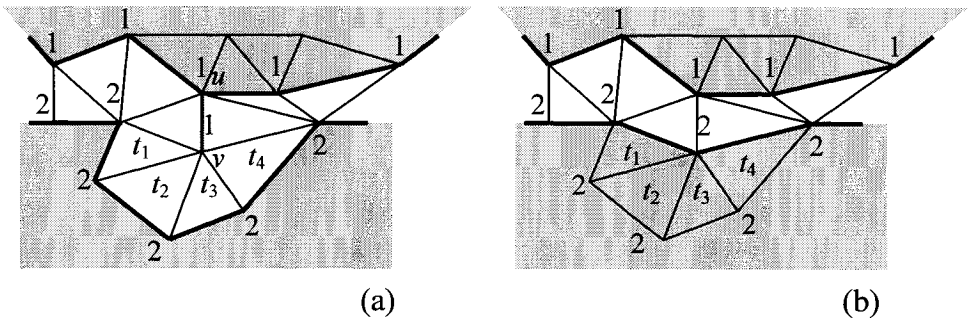


Figure 4.10 Branch pruning. (a) edge *uv* is a branch; (b) after branch pruning.

Table 4.4 Pseudo code for branch pruning algorithm.

Algorithm 4: Branch pruning algorithm
Input: a segmented mesh after region merging
for $v_i \in V$
determine whether v_i is a bounder vertex
endfor
for $v_i \in$ all boundary vertices
$N \leftarrow$ number adjacent triangles of v_i
$N_b \leftarrow 0$
for $t_j \in$ all adjacent triangles of v_i
if t_j is boundary triangle
N_b++
else
break;
endif
endfor
if $N_b == N$

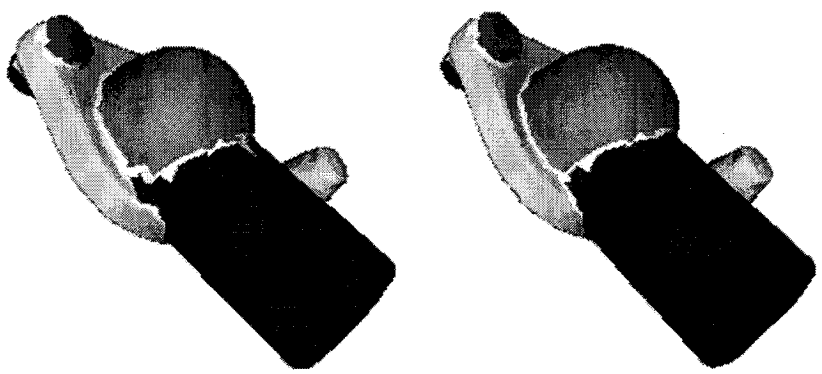
```
        region number of v_i ← its neighbour region number
        merge affected triangles with same vertex region
          number to corresponding region
      endif
  endfor
```

Until now, we have described our XMR neighbourhood based watershed segmentation algorithm. The experimental results will be given in the next section.

4.3 Experimental Results

We have implemented this XMR neighbourhood based 3D mesh segmentation algorithm under the same environment as described in Chapter three.

4.3.1 Comparison of the Feature Extraction Method

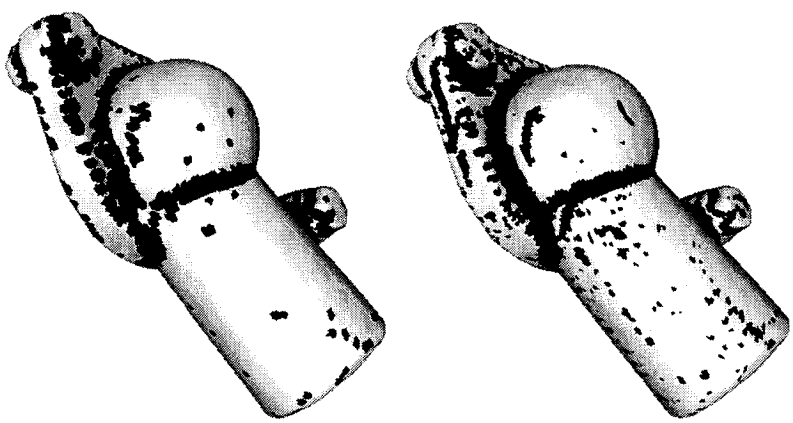


(a) (b)

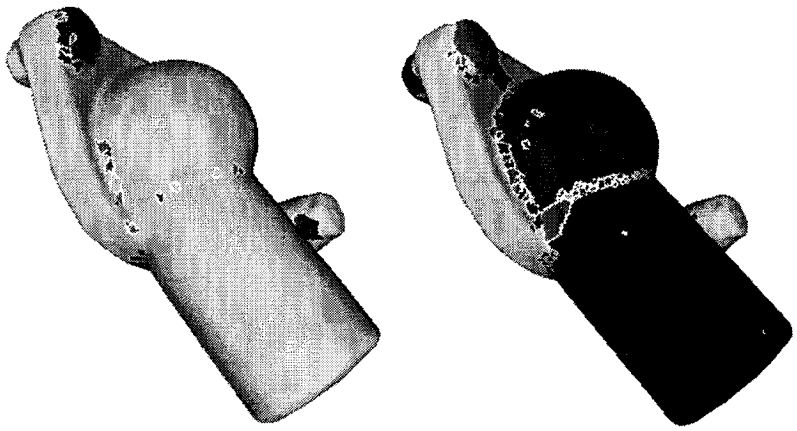
Figure 4.11 Feature extraction comparison for a low resolution Waterneck model. (a) Zhang's method; (b) our method using 1-ring neighbourhood. Both show the segmentation result after region merging.

In [ZPK+02], Zhang proposed a simple segmentation algorithm using the Gaussian curvature only. To remove the isolated vertices, this method just replaces the labels of the

isolated vertices with the labels of their adjacent vertices. As mentioned above, this method works well for the low-resolution models but may fail to detect the concave corners for the high-resolution models. Our algorithm combines the Gaussian curvature and the concaveness estimations and can effectively detect the whole boundaries. One of the models used in [ZPK+02] is Waterneck, but the resolution is not specified. We will compare those two methods using a low-resolution Waterneck model and a high-resolution Waterneck model. In Figure 4.11, a low-resolution Waterneck model with 2,502 vertices and 5,000 faces is used. We can see that both of those methods can create reasonable segmentation results, while our approach can create smoother boundaries. In Figure 4.12, a high-resolution Waterneck model with 58,784 vertices and 117,564 faces is used. In this figure, clearly there is a hole at the concave corner that breaks the boundary line for Zhang's algorithm. This results in an unfair segmentation. In contrast, our method can fix this problem and produce a meaningful segmentation.



(a) (b)



(c) (d)

Figure 4.12 Feature extraction comparison for a high resolution Waterneck model. The first row shows the boundary areas in blue; the second row shows the segmentation result before region merging. (a) Zhang's method; (b) our method using 1-ring neighbourhood.

4.3.2 1-ring neighbourhood vs. XMR neighbourhood

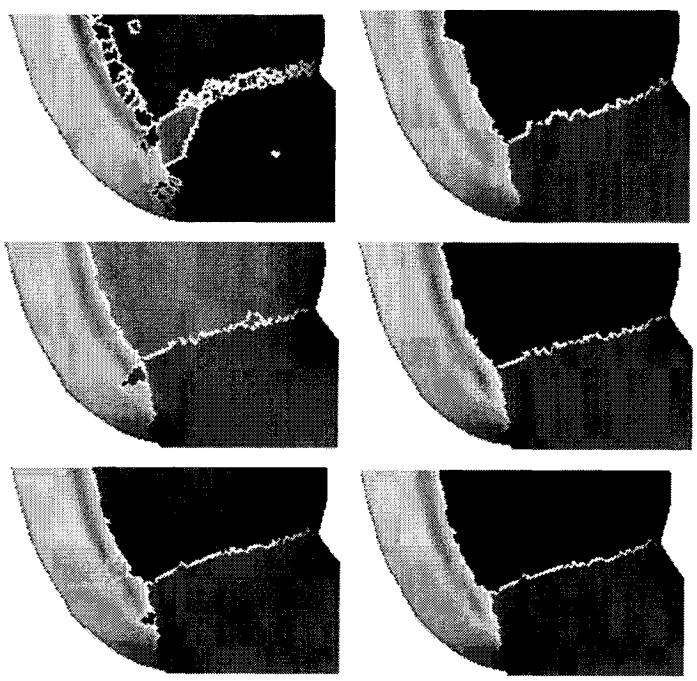


Figure 4.13 1-ring neighbourhood vs. XMR neighbourhood in segmentation. The three rows are based on 1-ring, 2-ring and 3-ring neighbourhood respectively. The left and right columns are before and after region merging respectively.

Most of the existing 3D segmentation algorithms are based on 1-ring neighbourhood. This is efficient when the models are simple. When a model is densely distributed with polygonal faces, its surface stretches smoothly like a real continuous surface. The 1-ring neighbourhood cannot catch the local geometric behavior, which will degrade the segmentation result. Our proposed XMR neighbourhood method can solve this problem, as the larger neighbourhood based estimation can better reflect the geometric behavior. The comparison of the segmentation results between 1-ring neighbourhood and XMR neighbourhood is demonstrated in Figure 4.13. The first column is the segmentation results before region merging; the second column gives the results after region merging. The three rows are based on 1-ring, 2-ring and 3-ring neighbourhoods respectively. As we can see, using the XMR neighbourhood can significantly improve the accuracy of segmentation.

4.3.3 More Segmentation Results and Performance Analysis

In Figure 4.14, we give more segmentation results for the different models. The models are listed in Table 4.5, where N_f and N_v denote the number of faces and the number of vertices respectively. As mentioned above, for the simple models like Teapot and Tommygun, our segmentation algorithm using a 1-ring neighbourhood is efficient enough; for the high-resolution models like Toilet, Bunny and Waterneck, using a XMR neighbourhood with level 2 or 3 can produce satisfactory results. Some models may need more XMR levels, e.g., we use XMR level 5 for the Hand model.

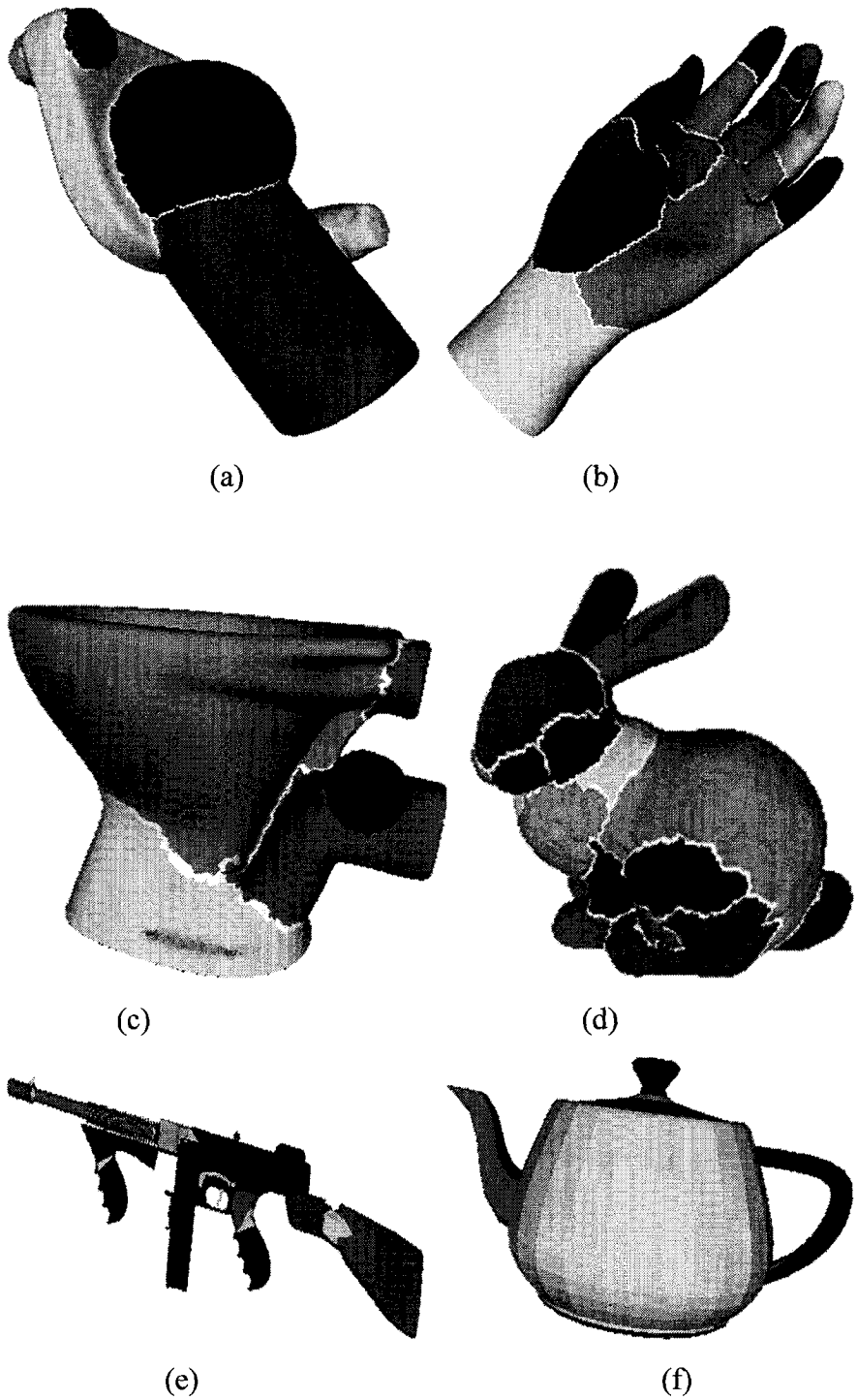


Figure 4.14 Segmentation examples. (a) Waterneck; (b) Hand; (c) Toilet; (d) Bunny; (e) Tommygun; (f) Teapot

Table 4.5 Numbers of vertices and triangles of the original models

Model	N_f	N_v	*XMR*
Waterneck	117,564	58,784	3
Hand	76,438	38,219	5
Bunny	69,451	34,834	2
Toilet	45,864	22,926	2
Tommygun	8,210	4,147	1
Teapot	2,256	1,177	1

Figure 4.15 shows the timing performance for the different XMR neighbourhoods. The x-axis is the level of the XMR neighbourhood from 1 to 5 (level 1 is 1-ring). The y-axis is the time spent for the segmentation. We can notice that the timing performance got degraded due to the XMR neighbourhood. However, our algorithm is still efficient. We can segment a large model Waterneck with 117k faces using the XMR level 3 within about 50 seconds, and obtain an accurate segmentation result (see Figure 4.14).

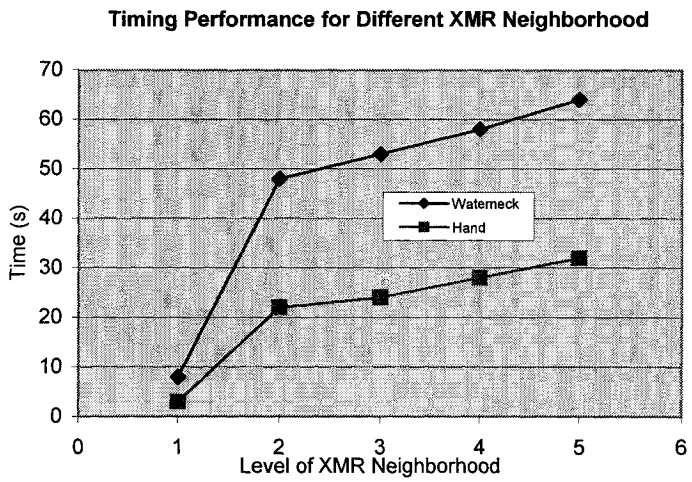


Figure 4.15 Timing performance for different XMR neighbourhood for the models of Waterneck and Hand.

Region merging is an important stage in mesh segmentation. The region depth based strategy is quite natural and reliable [MW99]. However, this method is not appropriate for

our case. We only use 0 and 1 to label the local minima and the boundaries, thus the region depth is always the 1. Yet, we still can compare the efficiency of region merging between the region depth based algorithm and our size and boundary length based algorithm. We will simulate the region depth based algorithm by using the distance from vertex to its average plane as the vertex height. We merge the regions until the final region number is the same as the result of our algorithm. Figure 4.16 demonstrates the efficiency comparison between those two methods. The x-axis is the initial region number before the region merging. The y-axis is time in seconds. The model size is also listed below the x-axis, as both initial region number and model size affect the timing performance of region merging. From Figure 4.16, we can clearly see our size and boundary length based algorithm is more efficient, in the mean time, it can create appropriate final segmentation results (Figure 4.14).

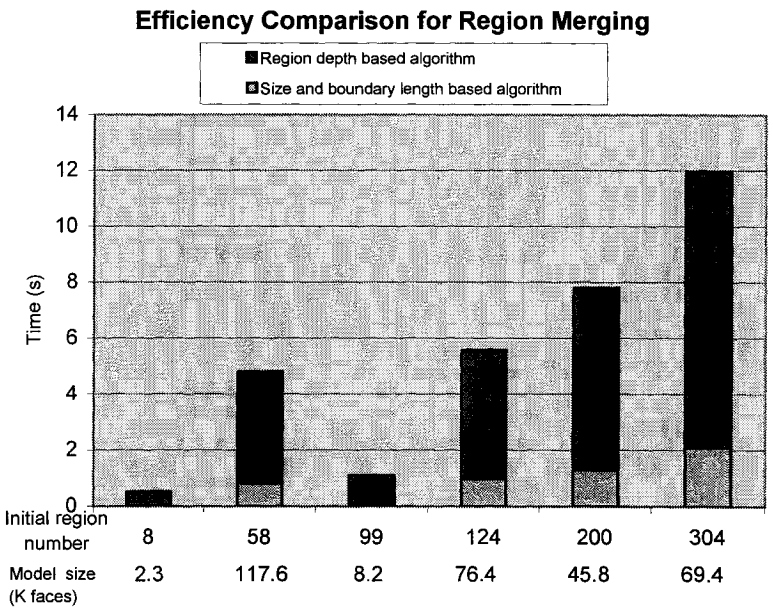


Figure 4.16 Efficiency comparison for region merging.

To compare the timing performance with the feature-based face clustering algorithms [LPR+02, SWG+03, KT03, ZMT03] that can partition an object into meaningful parts, we use their results directly. Their running environments are PIII or P4, which are close or superior to ours. The model we chose is the Stanford Bunny (69K faces). The comparison is listed in Table 4.6, where we can see that our algorithm is more efficient than these feature-based clustering algorithms. (Note that the model in [ZMT03] is the Venus with 67K faces.)

Table 4.6 Timing performance comparison between our algorithm and feature based clustering algorithms.

Our algorithm XMR level 2	Lévy's [LPR+02]	E. Zhang's [ZMT03]	Katz's [KT03]	Sander's [SWG+03]
18 s	30 s	67 s	244 s	**No more than 5 minutes**

CHAPTER 5

5 Segmentation Based 3D Mesh Compression

3D graphics models are more and more widely used in various applications, including video gaming, engineering design, architectural walkthrough, virtual reality, e-commerce and scientific visualization. To achieve a high level of realism, complex models are required, and they are obtained from various sources such as modeling software and 3D scanning. These highly detailed models may have thousands or even millions of vertices and polygons. They usually demand a huge amount of storage space and network bandwidth in the raw data format. Thus, it is essential to compress the 3D models efficiently. Over the last decade this research area has been very active. Many coding algorithms, including lossy and lossless, single resolution and multi-resolution, have been proposed [Dee95, GS98, TR98, LK97, Ros99, TG98]. We have reviewed the existing mesh compression algorithms in Chapter two. Among them, the Rossignac's Edgebreaker [Ros99] method is an efficient and simple one. It can handle manifold meshes with multiple boundary loops and handles. There has been a significant progress in 3D mesh coding performance; however, most of these existing methods compress and transmit the 3D object as a whole. In some interactive or selective applications, such as Internet gaming, collaborate CAD design, and remote learning, the client may be interested in particular section(s) of the object. Therefore, before transmission the server needs to segment the object into parts and send them individually or sequentially.

In this dissertation, we present a segmentation based mesh compression algorithm. We have chosen the efficient and robust neighbourhood based watershed segmentation method described in Chapter four. After segmentation, a 3D object is divided into multiple regions and the boundary triangle strips that connect those regions. We compress each region individually using an Edgebreaker algorithm and put them into one stream. Following that we encode the boundary strips and append the data to the end of the stream. The decoder can then get each individual region and restore the whole object using the boundary strips. We call this segmentation based compression scheme region-conquer-and-stitch. The overall format of the data stream is shown in Table 5.1.

Table 5.1 Overall data stream format.

Header	Region 1	...	Region n	Boundary strips

5.1 Region Compression

5.1.1 Edgebreaker

The Edgebreaker, proposed by Rossignac [Ros99], is a popular algorithm used to encode and decode the connectivity of a simply connected triangular mesh. It guarantees a worst-case coding cost of 4 bits per vertex (bpv). King and Rossignac [KR99] and Gumhold [Gum99] improved this upper bound to 3.67 and 3.522 bpv respectively. But the real attractiveness of Edgebreaker lies in the efficiency and simplicity of the algorithm. It has been recently enhanced to support meshes with multiple boundary loops or with multiple genus. This fits fairly well to our case, as after segmentation an individual region often has multiple boundary loops that connect to different neighbour

regions. Because we will use the similar idea for boundary strip encoding, we give a brief description of the Eedgebreaker in this section.

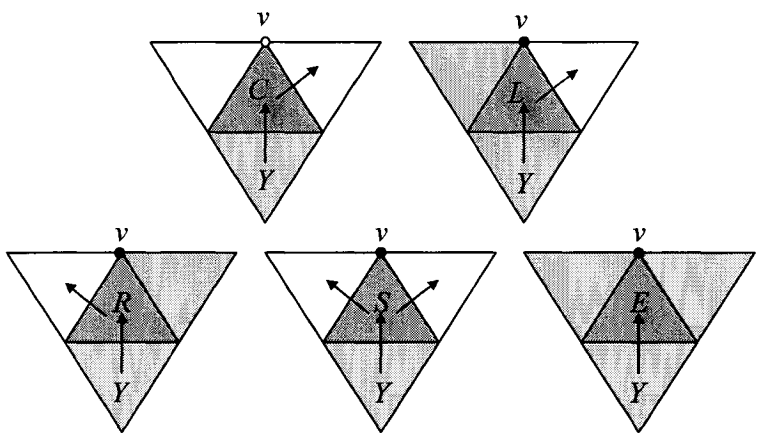


Figure 5.1 Edgebreak "*CLERS*"cases.

The Edgebreaker uses a triangle conquest approach to code the connectivity information. At each step, one triangle is conquered and a so-called op-code (operation code) is output. The connectivity information can be recovered from the op-code sequence. More specifically, the Edgebreaker is a state machine. At each state it moves from a triangle Y to an adjacent triangle X and marks all the visited triangles and their bounding vertices. Let $X.l$ and $X.r$ denote the left and right triangles that are incident upon X. Let v be the vertex common to X, $X.l$ and $X.r$. It also called the tip vertex of triangle X. Five situations are distinguished when it traverses the triangles as shown in Figure 5.1, and five corresponding op-codes are produced as follows:

1. C: if v, $X.l$ and $X.r$ are not visited.
2. L: if v and $X.l$ are visited and $X.r$ is not.
3. R: if v and $X.r$ are visited and $X.l$ is not.

4. *S*: if *v* is visited and *X.l* and *X.r* are not.

5. *E*: if *v* and *X.l* and *X.r* are all visited.

With these five cases, Edgebreaker compression is a recursive procedure that transverses the mesh along a spiraling triangle-spanning-tree. The recursion starts only at the triangles that are of type *S* and compresses the branch adjacent to the right edge of such a triangle while the left branch is pushed into a stack. When the corresponding *E* triangle is reached, the branch traversal is complete and the routine returns from the recursion to pursue the left branch. The encounter of an *E* that does not match an *S* terminates the compression process.

Figure 5.2 shows an example of the encoding process, where the arrows and the numbers give the order of the triangle conquest. The op-codes that are generated when the triangles are conquered are filled in the triangles. For this case, the encoder outputs the series of op-codes as *CCRSRLLRSEERLRE*.

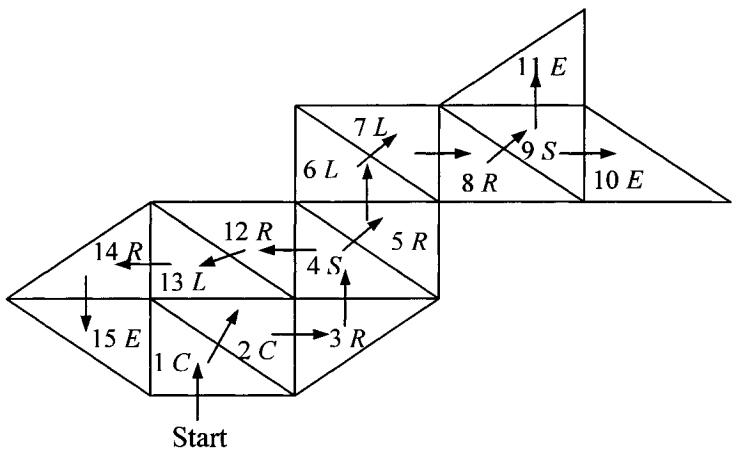


Figure 5.2 An example of the Edgebreaker encoding process.

The pseudo code of the Edgebreaker encoding algorithm is listed in Table 5.2; the decoding is the inverse procedure of the encoding process.

Table 5.2 Pseudo code for Edgebreaker encoding process.

```
Algorithm 5: Edgebreaker encoding
Input: a manifold mesh, select an initial triangle t
Recursive procedure compress(t)
 while (1)
   mark t as visited
   if the tip vertex of t was not visited
     encode tip vertex coordinates
     append 'C' to clers string
     mark tip vertex as visited, t ← t.r (right neighbour triangle)
   else if right triangle t.r was visited
     if left triangle t.l was visited
       append 'E' to clers string
       return
     else
       append 'R' to clers string
       t ← t.l (left neighbour triangle)
     endif
   else if left triangle t.l was visited
     append 'L' to clers string
     t ← t.r (right neighbour triangle)
   else
     append 'S' to clers string
     compress(right t.r)
       t ← t.l (left neighbour triangle)
   endif
 endwhile
```

5.1.2 Region Compression Using Edgebreaker

In this dissertation, we have chosen to use the Edgebreaker to compress the segmented regions. The source code of the Edgebreaker algorithm is publicly available, and can be found at http://www.gvu.gatech.edu/~jarek/edgebreaker/eb/EBsoftware.html. In the code, an OVTable (opposite corner vertex table) data structure is used, and the input/output 3D object files for compression/decompression are only in this OVTable

format. The OVTable format is specifically for the Edgebreaker algorithm, and is not efficient for 3D model representation. Also the original code only reads from and writes to the character string files. To make it more generic, we have extended the original source code to support the more popular 3D model file formats, such as PLY, OBJ, and WRL (VRML file format); and to produce the compressed data in a binary file format.

In our case, besides the individual region coding, we also need to encode a little bit more information that will be used for boundary strip coding, such as the number of boundary loops and the coordinates of the first vertex of each boundary loop. Details will be described in the next section. Therefore, we will add an extra header in the data stream for each region as shown in Table 5.3.

Table 5.3 Data stream format for each region.

Region number	Number of boundary loop(N)	List of first vertex of every boundary loop
Connectivity information (CLERS sequence)		Geometry information

5.2 Region Stitching

We have all the regions compressed and placed into the data stream. The decoder can decompress them independently. But there are gaps between the regions. To restore the whole model, we will need to encode the boundary triangle strips that connect each region together.

5.2.1 Boundary Triangle Strips

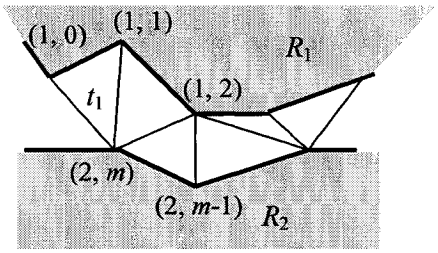


Figure 5.3 Boundary triangle is expressed as a (region number, boundary vertex index) tuple.

As demonstrated in chapter four, after the fast marching watershed segmentation followed by a region merging and a branch pruning, it can be ensured that there is only one single triangle strip between two adjacent regions. In other words, if we define the edge as the geodesic distance unit, then the geodesic distance between each boundary vertex and its adjacent region is one. This also implies that all the vertices have been visited. Thus, the region compression would have traversed all the vertices and encoded their geometric information. For the remaining boundary strips, we only have to encode the connectivity information. As the geometry data of the boundary strips have been stored in the compressed regions, what we need is to encode the boundary vertex indices. The Edgebreaker decompression does not guarantee the same vertex and triangle orders as the original, but we can easily get the boundary loops of each region and store the boundary vertex in the same order. Hence, as long as we know which vertex is the first one of the boundary loop, we can reorder the boundary vertices to be the same as the original. That is why we need the first vertex information of each boundary loop, as shown in Table 5.3. Therefore, the vertex in boundary strips will be expressed as a region number and a boundary vertex index. Figure 5.3 illustrates the expression of a boundary triangle t_1 as ((1, 0), (1, 1), (2, m)).

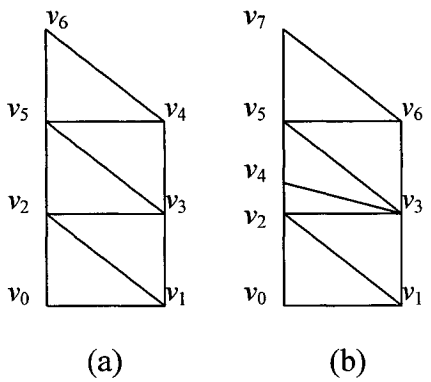


Figure 5.4 Triangle strips. (a) a sequential triangle strip; (b) a generalized triangle strip.

In computer graphics, triangle strips are utilized by software and hardware for efficient rendering of triangular meshes. They reduce the data transfer rate between the main memory and the graphics engine by allowing the reuse of vertices for up to three consecutive triangles. This requires the graphics hardware to have a built-in buffer for two vertices, which is very common in today's graphic boards.

The triangle strip primitive is supported in many graphics libraries such as OpenGL and Irix-GL. In Figure 5.4 a sequential triangle strip is shown on the left. It is called sequential because it turns alternating to the right and to the left. The order of vertices is $(v_0, v_1, v_2, v_3, v_4, v_5, v_6)$, and the triangle can be described as $\{(v_i, v_{i+1}, v_{i+2})\}$ for even i and $\{(v_{i+1}, v_i, v_{i+2})\}$ for odd i with $0 \leq i < 5$. The distinction between odd and even assures a consistent orientation of all triangles. Using the sequential strip, the transmit cost of n triangles can be reduced from $3n$ to $n+2$ vertices. The strip on the right in Figure 5.4 is not sequential because it contains consecutive turns in the same direction. Such a strip is called a generalized strip. In order to describe the triangle (v_5, v_3, v_6), vertex v_3 needs to be

inserted after v_4. Thus the order of vertices for this strip is (v_0, v_1, v_2, v_3, v_4, v_3, v_5, v_6, v_7). The decoder will need to delete the degenerated zero-area triangle (v_4, v_3, v_3).

For our case, if a region has only one adjacent region, then the boundary strip between them is a generalized strip. However, if a region has multiple neighbours, the boundary strips have a complex structure with multiple loops. If the complex boundary strips can be decomposed into a number of generalized triangle strips, then we can represent them as a number of vertex sequences. The problem lies in how to cut the complex strip and how to form the vertex sequence in the right order.

5.2.2 Region Stitching Using Boundary Strips

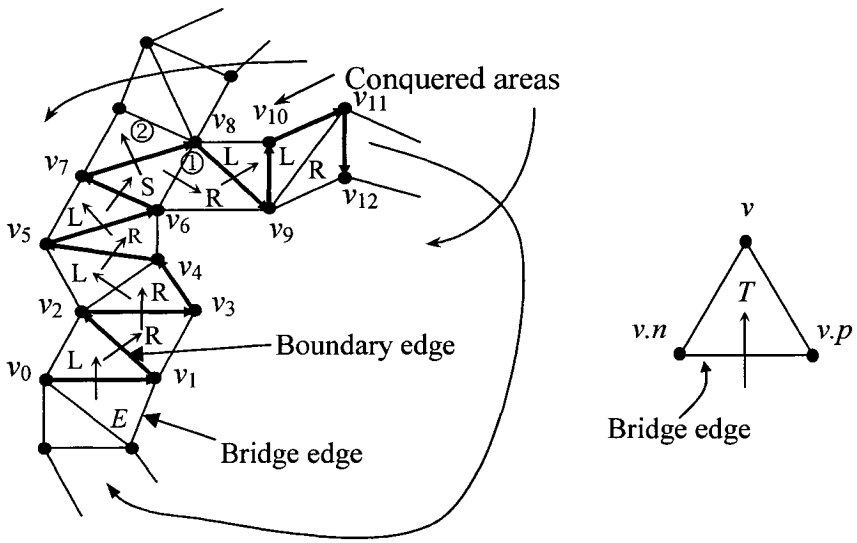


Figure 5.5 Boundary strip encoding.

A nice solution for this is that we can use the Edgebreaker-like method to traverse the boundary strips. Instead of producing the "*CLERS*" symbol, we will be storing the tip

vertex of the visited triangle. For a boundary triangle, we define the edge property as follows:

Definition 5.1: If the two end vertices of an edge in a boundary triangle belong to the same region, it is a boundary edge; otherwise, it is a bridge edge.

A boundary triangle has one boundary edge and two bridge edges. This is depicted in Figure 5.5. On the right of the Figure, v is the tip vertex of triangle T that is being processed; $v.n$ and $v.p$ denote the next and the previous vertices of v, and we are assuming the counter-clock-wise orientation. In the boundary strips, there will be only the "*LERS*" type of triangles because all the vertices have been visited during the region conquest. When the encoding process starts, we select an initial boundary triangle and find a bridge edge in that triangle. Two end vertices and the corresponding tip vertex are stored as v_0, v_1 and v_2. Then we start to traverse the boundary strips from this initial triangle and store the tip vertices into the vertex sequence. Similar to the Edgebreaker, the encoding proceeds if the triangle is L or R type and splits at a S triangle. When there is a S triangle it branches to the right triangle and pushes the left edge into a stack that are shown in Figure 5.5 as steps ① and ②. When we encounter an E triangle, a generalized triangle strip is generated. Then we pop up the stack and start another one. If the stack is empty we start from another un-visited triangle. The process stops when all the boundary triangles are traversed. On the left of Figure 5.5, the thin arrows across the bridge edges show the triangle traversing order; and the thick arrows along the bridge edges show the vertex order in the boundary vertex sequence.

Normally the generated triangle strips are generalized strips. Therefore, special care needs to be taken for inserting vertices in the vertex sequence as described above. The vertex insertion algorithm is as following:

if (T's type == last T's type) && (type==R)

insert v.n before v

if ((T's type == last T's type) && (type==L)) || (type==S)

insert v.p before v

In our example shown in Figure 5.5, the triangle list is (*LRRLRLSRLLR*...), and the vertex sequence after insertion will be (v_0, v_1, v_2, v_3, $\boldsymbol{v_2}$, v_4, v_5, v_6, v_7, $\boldsymbol{v_6}$, v_8, v_9, v_{10}, $\boldsymbol{v_9}$, v_{11}, v_{12}...). The inserted vertices are v_2, v_6 and v_9. The encoded data for boundary triangles is then a combination of a number of generalized strips. The data format of each strip is shown in Table 5.4.

Table 5.4 Boundary strip data stream format.

Number of strips	Number of vertices of strip 1	Number of triangles of strip 1
Vertex sequence of strip 1		Next strip ...

5.3 Experimental Result

Under the same environment as in the previous chapters, we have implemented the region-conquer-and-stitch algorithm using Visual C++ with an OpenGL library.

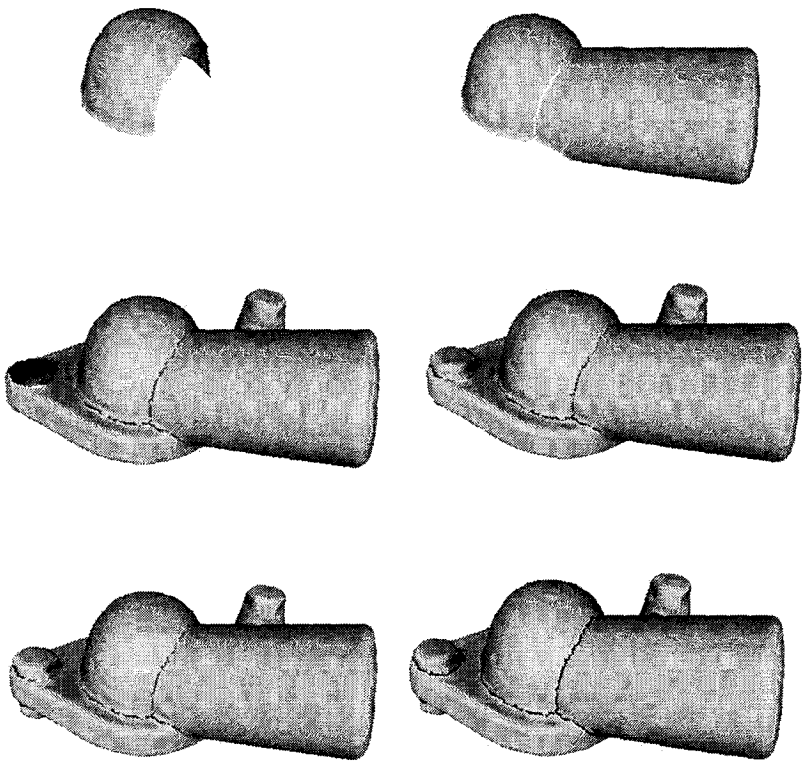


Figure 5.6 Part-by-part object transmission.

The output of the neighbourhood based segmentation as described in Chapter four is a series of independent patches and the skeleton-like boundary triangle strips. We compress them separately and place the compressed data into a stream. In a client-server environment, the client can ask for particular part(s) or the whole object. If all data have been received, the client can stitch the parts together using the final boundary strips and restore to the original object. Figure 5.6 is a simulation of this part-by-part transmission. The model we used is the Waterneck with 118K triangles and 59K vertices. The receiver gets one region at each step until the full Waterneck has been resumed.

Compared with the original Edgebreaker algorithm for compressing a whole object, our algorithm only has some minor extra information, which is some headers and the final

boundary vertex sequence. For example, for the Waterneck model, the connectivity compression ratios of the Edgebreaker and our algorithm are 4.0 bpv and 4.2 bpv respectively. With a similar compression ratio, we have the nice part-by-part feature.

CHAPTER 6

6 Conclusion and Future Work

This chapter summarizes our research described in the previous chapters and points out opportunities for future work.

6.1 Conclusion

In this dissertation, we have studied a framework for 3D mesh processing. Our contributions in this research focus on the following three items.

Efficient 3D Mesh Multi-resolution Modeling

We proposed an efficient 3D mesh multi-resolution modeling algorithm to generate an arbitrary resolution model. This method uses two error metrics for the edge collapse operation, which are edge curvature and 1-ring neighbourhood face area change. The edge curvature defines the visual importance of a given edge while the face area change considers the consequence of the edge collapse operation. Moreover, our approach also detects failure scenarios, such as collinear vertices and triangle flipping error. In our algorithm, the collinear vertices detection is a byproduct of the face area change error metric; and the triangle flipping error detection can be easily derived from the edge curvature error metric. Experimental simulations show that the proposed algorithm is

efficient in terms of the running speed, and yet effective that it can well preserve the key features of the original models while the other fast methods cannot at the same resolution.

Efficient and Robust Neighbourhood Based 3D Mesh Segmentation

We presented an efficient and robust neighbourhood based 3D mesh segmentation algorithm. To partition a 3D model into semantic meaningful regions, we have to detect the key features of the given mesh. Though the Gaussian curvature can effectively identify elliptic and hyperbolic behaviors of a 3D polygonal mesh, it fails to detect whether the corner vertex is convex or concave. We proposed a concaveness detection algorithm to complement the Gaussian curvature. This works well for the low-resolution models, but not the high-resolution models, because the vertex behaviors are very close on a densely distributed mesh. To capture more relevant geometric information, in our algorithm, we enlarged the normal 1-ring neighbourhood to an eXtended Multi-Ring (XMR) neighbourhood during the estimation for the high-resolution meshes. After the feature extraction, we developed a fast marching watershed segmentation algorithm followed by an efficient region merging and a branch pruning. Experimental results demonstrate that this XMR neighbourhood base segmentation method can successfully segment the high-resolution models.

Segmentation Based 3D Mesh Compression

We proposed a segmentation based 3D mesh compression algorithm. Most of the existing 3D mesh compression algorithms, no matter they are single-resolution or multi-resolution based, compress the object as a whole. Our segmentation based approach

compresses the different regions of the object separately and puts them into one stream. After the mesh segmentation described as above, a 3D model is partitioned into multiple sections and the boundary strips that connect these regions. The proposed encoder uses the Edgebreaker to compress each region individually and store the data into one stream in sequence. Afterwards, the boundary triangle strips are encoded and appended to the end of the stream. The boundary strip encoding is similar to the Edgebreaker, instead of the "*CLERS*" op-codes, it outputs the boundary vertex number sequences. After receiving the compressed data stream, the decoder can re-create the regions independently and store the whole object using the boundary strips. This is useful in interactive or selective applications, where the users can ask for the interested parts of the models or the complete models, if they want.

6.2 Future Work

Multi-resolution modeling, segmentation and compression are three main areas of 3D mesh processing. Our research proposed some new ideas in these areas. There are still some opportunities to improve the proposed algorithms or extend the work to the new directions. Future work is listed as following:

- Explore the possibility of mathematic approximation of the edge curvature and flipping error detection equations to further improve the simplification efficiency. Though our multi-resolution modeling algorithm is fast and effective, there still is room to improve for real-time applications.
- Boundary smoothing after mesh segmentation. In our algorithm, currently, the boundaries between the regions are formed directly from the region contours

after the plateau erosion and the region merging, which are often jagged and not visually pleasant. Thus, boundary edge smoothing is necessary in the post-processing step.

- Segmentation based 3D mesh simplification. Another avenue of future research is to incorporate the segmentation into mesh simplification. After segmentation, we can extract key points from the boundary edges and from each individual region based on the Gaussian curvature. Those points form a base model and will retain at any resolution during simplification. This way, the skeleton of a model is preserved even it has been simplified to a very low resolution. Although this method requires preprocessing for the key points extraction, it is still an interesting topic of future research.
- Segmentation based multi-resolution compression. We have presented a segmentation based single-resolution coding scheme. One attractive future direction is to encode each region in a multi-resolution fashion. Similar to but not the same as the progressive coding, we can define a fixed number of resolutions for each individual region. In the data stream of each region, the lower resolution data is followed by the higher resolution ones. Users can request different resolutions for different regions based on the priorities of the regions.

REFERENCES

[AHM96] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. Skiena, "Hamiltonian Triangulations for Fast Rendering", The Visual Computer, Vol. 12, no. 9, pp. 429–444, 1996.

[AY95] C. Alpert, and S. Yao, "Spectral Partitioning: the More Eigenvector, the Better", 32nd ACM/IEEE Design Automation Conference (San Francisco), pp. 195-200, 1995.

[BD92] C. L. Bajaj, and T. K. Dey, "Convex Decomposition of Polyhedra and Robustness", SIAM Journal of Computing, 21(2): 339-364, 1992.

[BGG+96] M. Baccar, L. A. Gee, R. C. Gonzalez, and M.A. Abidi, "Segmentation of Range Images via Data Fusion and Morphological Watersheds", Pattern Recognition, 29(10): 1671-1685, 1996.

[BL79] S. Beucher, and C. Lantuéjoul, "Use of Watershed in Contour Detection", In Proc. International Workshop on Image Processing, Renes, Sept 17-21, 1979.

[BPZ98] C. L. Bajaj, V. Pascucci, and G. Zhuang, "Compression and Coding of Large CAD Models", Technical Report, TICAM, The University of Texas at Austin, 1998.

[BWA98] S. G. Burgiss, R. T. Whitaker, and M. A. Abidi, "Range Image Segmentation through Pattern Analysis of the Multiscale Wavelet Transform", Digital Signal Processing, 8: 267-276, 1998.

[CG02] L. Chen, and N. D. Georganas, "An Efficient Multi-resolution Modeling for 3D Objects in Applications of Virtual Environment", HAVE2002, pp. 49-54, Ottawa, Nov. 2002

[CG04] L. Chen, and N. D. Georganas, "An Efficient and Robust Algorithm for 3D Mesh Segmentation", Journal of Multimedia Tools, and Applications (to appear).

[CG05] L. Chen, and N. D. Georganas, "3D Mesh Compression Using an Efficient Neighbourhood-based Segmentation" Proc 9th IEEE International Symposium on Distributed Simulation and Real-time Applications (DS-RT 2005), Montreal, Oct. 2005.

[Cha84] B. Chazelle, "Convex Partitions of Polyhedra: a Lower Bound and Worst-case Optimal Algorithm", SIAM Journal of Computing, 13(3): 488-507, 1984.

[Chu97] F. R. K. Chung, "Spectral Graph Theory", No. 92 in CBMS Regional Conference Series in Mathematics, American Mathematical Society, 1997.

[Cho97] M. M. Chow, "Optimized Geometry Compression for Real-time Rending", Proceedings of IEEE Visulization'97, pp. 347-354, Phoenix, AZ, 1997.

[Cla76] J. H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms", CACM, 19(10), pp. 547-554, October 1976.

[CMO97] J. Cohen, D. Manocha, and M. Olano, "Simplifying Polygonal Models Using Successive Mappings", Proceedings IEEE Visualization'97, pp. 395-402, 1997.

[CMS98] P. Cignoni, C. Montani, and R. Scopigno, "A Comparison of Mesh Simplification Algorithms", Computers and Graphics, 22(1): 37-54, 1998.

[CMT+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. V. Wright, "Simplification Envelopes", SIGGRAPH '96 Proc., Aug. 1996, pp. 119-128.

[CP97] B. Chazelle, and L. Palios, "Decomposing the Boundary of a Nonconvex Polyhedron", Algorithmica, 17: 245-256, 1997.

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces", Computer Graphics Forum, 17(2), pp. 167-174, June 1998.

[Dee95] M. Deering, "Geometric Compression", Computer Graphics (SIGGRAPH'95 Proceedings), pp. 13-20, 1995.

[Dee98] M. Deering, "Geometic Decompression Hardware", In Course Notes of SIGGRAPH'98, 1998.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes", Computer Graphics SIGGRAPH 95 Proceedings, pp. 173-182.

[ESV96] F. Evans, S. S. Skiena, and A. Varshney, "Optimizing Triangle Strips for Fast Rendering", IEEE Visualization, pp. 319–326, 1996.

[FRS02] G. Froimovich, E. Rivlin, and I. Shimshoni, "Object Classification by Functional Parts", Proceedings of the International Symposium on 3D Data Processing, Visualization, and Transmission, pp. 648-655, Padova, Italy, 2002.

[Gar99] M. Garland, "Quadric-Based Polygonal Surface Simplification", Ph. D. thesis, Carnegie-Mellon University, 1999.

[GCC+95] R. M. Gray, L. Craver. M. Cohn, A. Gersho, T. Lookabaugh, F. Pollara, and M. Vetterli, "Non-US Data Compression Research", Technical Report, Standford University et al, USA, 1995.

[GGH02] X. Gu, S. Gortler, and H. Hoppe, "Geometry Images", SIGGRAPH 2002, pp. 355-361, 2002.

[GH97] M. Garland, and P. S. Heckbert, "Surface Simplification Using Quadric Error Metrics", in Proc. Computer Graphics Ann. Conf. Series, Aug. 1997, pp. 209-217.

[GH98] M. Garland, and P. S. Heckert, "Simplifying Surfaces with Color and Texture Using Quadric Error Metrics", Proceedings IEEE visualization'98, pp. 263-269, 1998.

[Geo97] N. D. Georganas, "Advanced Distributed Simulation Technology and Collaborative Virtual Environments", Research Reports, CRC, Ottawa, Canada, 1997.

[GG92] A. Gersho and R. M. Gray, Vector Quantization and Signal Compression, Kluwer Academic Publishers, 1992.

[Gon92] R. C. Gonzalez. Digital Image Processing. Addison-Wesley, 1992.

[Got03] C. Gotsman, "On Graph Partitioning, Spectral Analysis, and Digital Mesh Processing", Proceedings of Shape Modeling International (Seoul), pp. 165-169, 2003.

[Gra93] A. Gray, "The Gaussian and Mean Curvatures", §14.5 in Modern Differential Geometry of Curves and Surfaces. Boca Raton, FL: CRC press, pp. 279-285, 1993.

[GS98] S. Gumhold and W. Straßer, "Real Time Compression of Triangle Mesh Connectivity", ACM SIGGRAPH, pp. 133–140, 1998.

[GTLH97] A. Gueziec, G. Taubin, F. Lazarus, and W. Horn, "Cutting and Stitching: Efficient Conversion of a Non-manifold Polygonal Surface to a Manifold", Technical Report RC-20935, IBM T.J. Watson Research Center, 1997.

[Gum99] S. Gumhold, "Improved Cut-border Machine for Triangle Mesh Compression", Erlangen Workshop '99 on Vision, Modeling and Visualization, IEEE Signal Processing Soc., 1999.

[GWH01] M. Garland, A. Willmott, and P. Heckbert, "Hierarchical Face Clustering on Polygonal Surfaces", Proc. ACM Symposium on Interactive 3D Graphics, March 2001.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, "Mesh Optimization", SIGGRAPH 93, pp. 19-26.

[HHK+95] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang, "Voxel-Based Object Simplification", Proceedings IEEE Visualization'95, pp. 296-303, 1995.

[HJJ96] A. Hoover, G. Jean-Baptiste, and X. Jiang, "An Experimental comparison of range image segmentation algorithms", IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(7): 673-689, 1996.

[Hop96] H. Hoppe, "Progressive Meshes", SIGGRAPH 96 Conference Proceedings, pp. 99-108, August 1996.

[Hop97] H. Hoppe, "View-Dependent Refinement of Progressive Meshes", SIGGRAPH 97 Conference Proceedings, pp. 189-198, August 1997.

[Hop98] H. Hoppe, "Efficient Implementation of Progressive Meshes", Computers & Graphics, Vol. 22, No. 1, 1998, pp. 27-36.

[HS98] J. E. Hershberger, and J. S. Snoeyink, "Erased Arrangements of Lines and Convex Decompositions of Polyhedra", Computational Geometry: Theory and Applications, 9: 129-143, 1998.

[Huf52] D. Huffman, "A Method for the Construction of Minimum-redundancy Codes", Proceedings of the IRE, 40(10): 1098-1101, 1952.

[ILG96] V. Isler, R. W.H. Lau, and M. Green, "Real-time Multi-resolution Modeling for Complex Environments", Synmposium on Virtual Reality Software and Technology, pp. 11-19, July 1996.

[KCS98] L. Kobbelt, S. Campagna, and H. P. Seidel, "A General Framework for Mesh Decimation", In Proc. Graphics Interface'98, pp43-50, June 1998.

[KG00] Z. Karni, and C. Gotsman, "Spectral Compression of Mesh Geometry ", In Proceedings of ACM SIGGRAPH 2000, pp.279-286, 2000.

[KG01] Z. Karni, and C. Gotsman, "3D Mesh Compression Using Fixed Spectral Bases", In Proceedings of Graphics Interface, pp. 1–8, 2001.

[KJK98] S. J. Kim, W. K. Jeong, and C. H. Kim, "LOD Generation with Discrete Curvature Error Metric", Korea, 1998.

[KLS96] L. Klein, G. Liebich, and W. Straβer, "Mesh Reduction with Error Control", Proceedings of Visualization'96, pp. 311-318, October 1996.

[KR99] D. King and J. Rossignac, "Guaranteed 3.67v bit encoding of planar triangle graphs," in 11th Canadian Conference on Computational Geometry, pp. 146–149, 1999.

[KSS00] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in ACM SIGGRAPH, 2000, pp. 271–278.

[KT94] A. D. Kalvin and R. H. Taylor, "Superfaces: Polyhedral Approximation with Bounded Error", SPIE Proceedings 2164, pp. 2-13, 1994.

[KT96] A. D. Kalvin and R. H. Taylor, "Superfaces: Polygonal Mesh Simplification with Bounded Error", IEEE Computer Graphics and Applications, Vol. 16, No. 3, pp. 64-77, May 1996.

[KT03] S. Katz, and A. Tal, "Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts", SIGGRAPH 2003, ACM Transactions on Graphics, Vol. 22 , Issue 3, July 2003, pp. 954-961.

[Kuh02] W. Kuhnel, "Differential Geometry: Curves - Surfaces - Manifolds", American Mathematical Society , 2002.

[LE97] D. Luebke, and C. Erikson, "View-Dependent Simplification of Arbitrary Polygonal Envrionments", SIGGRAPH 97 Conference Proceedings, pp. 199-208, August 1997.

[Li98] J. Li, "Progressive Compression of 3D Graphics", Ph.D. Dissertation, University of South California, 1998.

[Lin82] A. Lingas, "The Power of Non-rectilinear Holes", Proceeding if the Ninth International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, pp369-383, New York, NY. Springer-Verlag, 1982.

[Lin98] P. Lindstrom, "A Survey of Multiresolution Techniques for Arbitrary Polygonal Meshes", Technical report, Georgia Institute of Technology, 1998.

[LK97] J. Li, and C.-C. J. Kuo, "Embedding coding of 3D graphic models", ICIP'97, vol 1, pp. 57-60.

[Lou94] J. M. Lounsbery, "Multiresolution Analysis for Surfaces of Arbitrary Topological Type", Ph.D. thesis, University of Washington, September 1994.

[LPR+02] B. Levy, S. Petitijean, N. Ray, and J. Maillot, "Least Squares Conformal Maps for Automatic Texture Atlas Generation", ACM Computer Graphics, Proc. SIGGRAPH 2002, pp. 362-371, 2002.

[LT97] K. Low, and T. Tan, "Model Simplification Using Vertex-clustering", 1997 Symposium on Interactive 3D Graphics. ACM SIGGRAH, 1997.

[LT98] P. Lindstrom, and G. Turk, "Fast and Memory Efficient Polygonal Simplification", Proceedings IEEE Visualization'98, pp. 279-286, 1998.

[LTG97] R. W.H. Lau, D. To, and M. Green, "An Adaptive Multi-resolution Modeling Technique Based on Viewing and Animation Parameters", IEEE Virtual Reality Annual International Symposium, pp. 20-27, 1997.

[LTH01] X. Li, T. W. Toon, and Z. Huang, "Decomposing Polygon Meshes for Interactive Applications", Proceedings of the 2001 Symposium on Interactive 3D Graphics, pp. 35-42, 2001.

[LZ04] R. Liu, and H. Zhang, "Segmentation of 3D Meshes Through Spectral Clustering", Proceedings Pacific Graphics, pp. 298-305, 2004.

[McC96] S. R. McCanne, "Scalable Compression and Transmission of Internet Multicast video", Ph.D. Dissertation, U.C. Berkeley, 1996

[MDS+02] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds", VisMath, Berlin, 2002.

[Mel98] S. Melax, "A Simple, Fast, and Effective Polygon Reduction Algorithm", Game Developer Magazine, Nov, 1998.

[MP77] R. S. Millman and G.D. Parker, *Elements of Differential Geometry*, Prentice-Hall Inc, 1977.

[MW99] A. Mangan, and R. Whitaker, "Partitioning 3D Surface Meshes Using Watershed Segmentation", IEEE Trans. Visualization and Computer Graphics, Vol. 5, No. 4, pp. 308-321, Oct.-Dec. 1999.

[PH97] J. Popovic, and H. Hoppe, "Progressive Simplicial Complexes", Computer Graphics (SIGGRAPH'97 Proceedings), pp. 217-224, 1997.

[PKA03] D. Page, A. Koschan, and M. Abidi, "Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds", in Conference on Computer Vision and Pattern Recognition (CVPR'03), Vol. II, pp. 129-137, 2003.

[PR00] R. Pajarola and J. Rossignac, "Compressed progressive meshes," IEEE Trans. Visualization and Computer Graphics, vol. 6, no. 1, pp. 79–93, 2000.

[Pre75] P. M. Prenter, "Splines and Variational Methods", John Wiley & Sons, New York, 1975.

[PRF02] S. Pulla, A. Razdan, and G. Farin, "Improved Curvature Estimation for Watershed Segmentation of 3-Dimensional Meshes", Submitted to IEEE Trans. Visualization and Computer Graphics, 2002

[RoB93] J. Rossignac and P. Borrel, "Mult-resolution 3D Approximation for Rendering Complex Scenes", Modeling in Computer Graphics: Methods and Applications, pp. 455-465, 1993.

[RaB02] A. Razdan, and M. Bae, "A Hybrid Approach to Feature Segmentation of 3-Dimensional Meshes", Computer-Aided Design, Apr. 2002.

[RHX+02] M. E. Rettmann, X. Han, C. Xu, and J. L. Prince, "Automated Sulcal Segmentation Using Watersheds on the Cortical Surface", NeuroImage, No. 15, pp. 329-344, 2002.

[RKS00] C. Rossl, L. Kobbelt, and H.-P. Seidel, "Extraction of Feature Lines on Triangulated Surfaces Using Morphological Operators", Smart Graphics 2000, AAAI Spring Symposium, Stanford University, pp. 71-75.

[RM01] J. R. Roerdlink, and A. Meijster, "The Watershed Transform: Definitions, Algorithms and Parallelization Strategies", Fundamenta Informaticae, 41:187-228, 2001.

[Ros99] J. Rossignac, "Edgebreaker: Connectivity Compression for Triangle Meshes", IEEE Trans. Visualization and Computer Graphics, vol. 5, no. 1, pp. 47–61, 1999.

[RR96] R. Ronfard, and J. Rossignac, "Full-range Apporximation of Triangulated Polyhedra", Proc. Eurographics'96, Aug. 1996.

[RS99] J. Rossignac and A. Szymczak, "Wrap and Zip Decompression of the Connectivity of Triangle Meshes Compressed with Edgebreaker," Computational Geometry, vol. 14, no. 1-3, pp. 119–135, 1999.

[RT98] Y. Rubner, and C. Tomasi, "Texture Metrics", In Proc. IEEE Intl. Conf. On Systems, Man, and Cybernetics, pp. 4601-4607, 1998.

[SB92] M. Suk, and S. M. Bhandarkar, "Three-Dimensional Object Recognition from Range Images", Springer-Verlag, Tokyo, 1992.

[SCG+02] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski, "Bounded-distortion Piecewise Mesh Parameterization", Proceedings of IEEE Visualization, 2002.

[SKR00] A. Szymczak, D. King, and J. Rossignac, "An Edgebreaker-based Efficient Compression Scheme for Regular Meshes," Proceedings of 12th Canadian Conference on Computational Geometry, pp. 257–264, 2000.

[SL96] M. Soucy, and D. Laurendeau, "Multiresolution Surface Modeling Based on Hierarchical Triangulation", Computer Vision and Image Understanding, 63(1):1-14, 1996.

[SMR98] A. Swan, S. McCanne, and L. A. Rowe, "Layered transmission and caching for the multicast session directory Service", ACM Multimedia 98 Proceedings, 1998.

[SRK02] A. Szymczak, J. Rossignac, and D. King, "Piecewise Regular Meshes: Construction and Compression", Graphic Models, pp. 183-198, 2002.

[SS01] S. Svensson, and G. Sanniti di Baja, "A Tool for Decomposing 3D Discrete Object", International Conference on Computer Vision and Pattern Recognition, Volume 1, pp. 850-855, 2001.

[SSG+01] P. Sander, J. Snyder, S. Gortler, and H. Hoppe, "Texture Mapping Progressive Meshes", Proceedings of ACM SIGGRAPH, pp. 409-416, 2001.

[STK02] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of Polyhedral Surfaces Using Decomposition", Computer Graphics Forum, vol. 21, No. 3, 2002. Proceedings of Eurographic 2002.

[SWG+03] P. Sander, Z. Wood, S. Gortler, J. Snyder, and H. Hoppe. "Multi-chart geometry images", Symposium on Geometry Processing, pp. 146-155, 2003.

[SZL92] W. Schroeder, J. Zarge, and W. Lorensen, "Decimation of Triangle Meshes", Computer Graphics, Volume 25, No. 3, (Proc. SIGGRAPH `92, pp. 65-70), July, 1992.

[Tau95] G. Taubin, "A Signal Processing Approach to Fair Surface Design", ACM SIGGRAPH, vol. 29, pp. 351–358, 1995.

[TCC+00] K. Tang, S.-T. Chen, L.-L. Chen, and T. C. Woo, "Tetrahedral Mesh Generation for Solids Based on Alternating Sum of Volumes", Computers in Industry, 41: 65-81, 2000.

[TG98] C. Touma and C. Gotsman, "Triangle mesh compression," Proceedings of Graphics Interface, 1998, pp. 26–34.

[TGHL98] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus, "Progressive Forest Split Compression", SIGGRAPH 98 Conference Proceedings, pp. 123-132, August, 1998.

[THLR98] G. Taubin, W. Horn, F. Lazarus, and J. Rossignac, "Geometri coding and VRML", Technical Report RC-20925, IBM T.J. Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, 1998.

[TR98] G. Taubin, and J. Rossignac, "Geometric Compression through Topological Surgery", ACM Transactions on Graphics, 17(2):84-115, 1996.

[Tur84] G. Turan, "On the succinct representation of graphs", Discrete Applied Mathmetics, 8:289-294, 1984.

[Tur92] G. Turk, "Re-tiling Polygonal Surfaces", Computer Graphics, Vol. 26, No. 2, pp. 55-64, July 1992.

[TZ94] D. Taubman, and A. Zakhor, "Multirate 3D Subband Coding of Video", IEEE Trans. Image Processing, vol. 3, pp. 572-588, Jan. 1994.

[VRML] ISO/IEC 14772-1. The Virtual Reality Modeling Language (VRML). 1997.

[VS91] L. Vincent, and P. Soille, "Watersheds in Digital Spaces: an Efficient Algorithm Based on Immersion Simulations", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13(6), pp. 583-598, 1991.

[WL97] K. Wu, and M. D. Levine, "3D Part Segmentation Using Simulated Electrical Charge Distributions", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 19, pp. 1223-1235, Nov. 1997.

[XHM99] X. Xiang, M. Held, and J. Mitchell, "Fast and Efficient Stripification of Polygonal Surface Models", ACM 1999 Symposium on Interactive 3D Graphics, pp. 71–78, 1999.

[YL99] M. Yang, and E. Lee, "Segmentation of Measured Point Data using a Parametric Quadric Surface Approximation", Computer-Aided Design, 31: 449-457, 1999.

[ZMT03] E. Zhang, K. Mischaikow, and G. Turk, "Feature-Based Surface Parameterization and Texture Mapping", Georgia Tech TR 2003-03-29.

[ZPK+02] Y. Zhang, J. K. Paik, A. Koschan, M. A. Abidi, and D. Gorsich, "A Simple and Efficient Algorithm for Part Decomposition of 3D Triangulated Models Based on Curvature Analysis", Proc. Int. Conf. Image Processing, Rochester, New York, Vol. III, pp. 273-276, Sep. 2002.