University of Alberta

R-SIMP: MODEL SIMPLIFICATION IN REVERSE, A VECTOR QUANTIZATION APPROACH

by

Dmitry D. Brodsky ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 1999

0-612-47012-1

University of Alberta

Library Release Form

Name of Author: Dmitry D. Brodsky

Title of Thesis: R-Simp: Model Simplification In Reverse, A Vector Quantization Approach

Degree: Master of Science

Year this Degree Granted: 1999

D Brodsky

Dmitry D. Brodsky
27 Lakebreeze Crescent.
St. Catharines, Ontario
Canada, L2M 7C3

Date: June 7, 1999

But if you keep proving stuff that others have done, getting confidence, increasing the complexities of your solutions - for the fun of it - then one day you'll turn around and discover that nobody actually did that one! And that's the way to become a computer scientist.

– Richard P. Feynman, 'Lectures on Computation'

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **R-Simp: Model Simplification In Reverse, A Vector Quantization Approach** submitted by Dmitry D. Brodsky in partial fulfillment of the requirements for the degree of **Master of Science**.

Ben Watson

Dr. Benjamin Watson (Supervisor)

[illegible]

Dr. Alinda Friedman (External)

[illegible]

Dr. John Buchanan (Examiner)

Date: May 7 1999

To my family, my friends
and to those who went
before their time.

Abstract

Just as colour image quantization can be viewed as simplification of three dimensional colours in a two dimensional image, model simplification can be viewed as quantization of three dimensional normal vectors on a two dimensional surface. Thus many of the approaches used in quantization can be applied to the problem of model simplification.

A model simplification algorithm is presented that is based on the *splitting* algorithm from quantization literature. The algorithm works in reverse by expanding from a coarse to a fine model. Traditionally, curvature is defined in an infinitely small local area. This algorithm measures orientation change in larger patches with techniques inspired by definitions of curvature from differential geometry. With these measures, the algorithm determines which portion of the model to partition next, and how to partition it. The algorithm can accept non-manifold input models and is capable of simplifying topology. It produces good quality simplifications and is faster than most other simplification algorithms.

Acknowledgements

Time stand still
I'm not looking back
But I want to look around me now
See more of the people
And the places that surround me
Now

-Rush

I would like to thank all the people who made this Master's thesis possible and enjoyable. These last two years have been a lot more than just a Master's. It has been a journey far beyond what I could have imagined.

You never know where the road of fate will lead you. I have always been interested in systems and when I came to the University of Alberta I was planning to do my Master's in it. In my first term I took a computer graphics course and met Ben Watson. He started a reading group on current topics in computer graphics and invited me to join; my interest in graphics was sparked. So, I would like to start off by thanking my supervisor Ben Watson for initiating that spark. I am greatful for all his help, his time and the freedom he gave me to pursue my interests.

I am forever indebted to my brother for his invaluable comments and his gargantuan effort in proof reading my papers and thesis. I would like to thank my parents and grandparents for their support and their never ending love and encouragement.

I would like to thank the graphics group for always making me feel welcome, for their help, and for some of the most interesting conversation I have ever had. Thank you, you are a great bunch of people.

And finally, I would like to thank all my friends that made the last two years very special. Thank you Matt and Paul for all your support, encouragement, and those long latenight talks. I can not begin to express my gratitude and thanks to Lourdes, Oscar, Mike, and Lara for their friendship, advice, and much much more. Thank you.

Table of Contents

1 Introduction **1**

1.1 Background 1

1.1.1 Vertex Clustering 2

1.1.2 Decimation 2

1.1.3 Vertex Merge & Edge Collapse 2

1.2 Motivation 2

1.3 Objectives 3

1.3.1 A Quick Overview of R-Simp 4

1.3.2 Evaluation of Simplification Quality 4

1.4 Contributions 5

1.5 What Follows 5

2 Quantization **7**

2.1 Vector Quantization 7

2.1.1 Product Codes 7

2.1.2 Splitting 8

2.1.3 Pruning 8

2.1.4 Pairwise Nearest Neighbour 8

2.1.5 K-Means 9

2.2 Quantization & Model Simplification 9

3 Previous Work **10**

3.1 Polygonal Mesh Simplification 10

3.1.1 Background 10

3.1.2 Algorithm Comparison Criteria 11

3.1.3 Summary of Simplification Algorithms 12

3.1.4 *Product Code* Based Algorithms 13

3.1.5 *Pruning* Based Algorithms 16

3.1.6 *Pairwise Nearest Neighbour* Based Algorithms 22

3.2 Preservation of Colour, Texture, and Other Attributes 30

3.3 Level of Detail 33

3.3.1 Adaptive Display Algorithms 33

3.3.2 Creation of Level of Detail Hierarchies 34

3.4 Conclusions 35

4 Curvature **37**
4.1 Importance of Curvature 37
4.2 Normal Curvature 37
4.3 The Gaussian Map 38
4.4 Approximating Curvature 39

5 The R-Simp Algorithm **40**
5.1 Overview 40
5.1.1 The System 40
5.1.2 Pre-Processing 41
5.1.3 Simplification 41
5.1.4 Post-Processing 42
5.2 Selecting a Cell to Split 42
5.3 Splitting A Cell 46
5.3.1 Overview & Objectives 46
5.3.2 Sampling Normal Curvedness 48
5.3.3 Determining How to Split a Cell 49
5.4 Computing the Centroid 52
5.4.1 Approaches Tested 53
5.4.2 Quadratic Vertex Placement Policy 54
5.4.3 Problems with the quadratic vertex placement policy 56
5.5 Topology Check 57

6 Evaluation & Discussion **59**
6.1 Simplification Speed 59
6.2 Quality of a Simplification 60
6.2.1 Quantitative Measures of Quality 60
6.2.2 Qualitative Measures of Quality 61
6.3 Evaluation 63
6.3.1 Measures Employed 63
6.3.2 The Test Bed 63
6.3.3 Comparisons 64
6.4 Other Criteria 68
6.5 The Pros of Simplifying in Reverse 68
6.6 The Cons 70
6.7 Enhancements 71
6.8 Applications of R-Simp 72
6.9 QSlim Version 2.0 72

7 Conclusion **79**
7.1 Summary of Objectives and Results 79
7.2 Summary 79
7.2.1 Existing Algorithms 80
7.2.2 R-Simp 80
7.3 Summary of Contributions 81

7.4 Future Work 82
7.4.1 Level of Detail Hierarchies 82
7.4.2 Preservation of Colour, Texture, and other Attributes 82
7.4.3 Progressive Transmission 83
7.4.4 Search & Simplification 83
7.4.5 Parallelization 83
7.4.6 Memory Constraint 84
7.4.7 Error Bound 84

Bibliography **85**

List of Figures

3.1 A patch with and without a hole. 18
3.2 Mutual Tessellation 20
3.3 Removal of original vertices. 20

4.1 The Gaussian map. 39

5.1 Selecting a cell: curvedness measure 43
5.2 Selecting a cell: area sensitivity 44
5.3 Selecting a cell: a patch's relative size 44
5.4 Selecting a cell: large and small scale curvedness. 45
5.5 Splitting a cell: reducing the amount of curvedness 46
5.6 Splitting a cell: splitting the wrong way 47
5.7 Splitting a cell: results of good and bad splitting 47
5.8 Normal curvedness: part one of computation 49
5.9 Normal curvedness: part two of computation 50
5.10 Normal curvedness: the result 51
5.11 Splitting a cell: against maximal curvedness 52
5.12 Splitting a cell: with one splitting plane 53
5.13 Splitting a cell: with two splitting planes 54
5.14 Splitting a cell: with three splitting planes 55
5.15 Centroid placement: cause of an ill-conditioned matrix 57

6.1 Models used in comparisons. 64
6.2 Bunny: output model size verses simplification time 65
6.3 Dragon: output model size verses simplification time 66
6.4 The disappearing cow leg. 67
6.5 Input model size verses simplification time 69
6.6 QSlim V2.0: output model size verses simplification time 73
6.7 The bunny model. 74
6.8 The dragon model. 75
6.9 The cow model. 76
6.10 R-Simp: visual quality and polygon count 77
6.11 QSlim: visual quality and polygon count 78

List of Tables

3.1 Summary of existing model simplification algorithms. 13

4.1 Characterization of surfaces 38

6.1 Statistics of models used in the comparisons. 64
6.2 Performance of R-Simp verses QSlim and Vertex Clustering 67
6.3 Comparison of R-Simp, QSlim, and Vertex Clustering 68

Chapter 1

Introduction

Many of todays computer graphics applications, such as virtual reality environments and data visualization, require the ability to display very large and complex models and/or datasets. Both types of environments require the data to be rendered as quickly as possible. While hardware is getting faster the size and complexity of models and data sets continue to grow and surpass the hardware's ability to render them in the required amount of time. No matter how fast the hardware is there will always be a model or dataset that it can not display fast enough.

One way to display these large models and datasets is to reduce their size and complexity, but they have to be reduced in such a way as to resemble the original as much as possible. This is where model simplification algorithms come in. Model simplification algorithms are able to reduce the model's size and complexity such that the hardware is able to display the model in the required amount of time, but still have the simplified model resemble the original.

1.1 Background

Model simplification is the process of reducing the number of polygons in a model while preserving the shape and/or appearance of the original. There are three main approaches to model simplification; each approach makes a tradeoff between simplification speed and the quality of the simplification.

1.1.1 Vertex Clustering

The fastest approach is vertex clustering but it produces the poorest simplifications. Vertex clustering algorithms simplify by subdividing the model's volume into clusters and then selecting a representative vertex from each cluster. The simplified model is formed by retriangulating the representative vertices. The size of the clusters dictate the size of the simplified model; the larger the clusters the smaller the model.

1.1.2 Decimation

Decimation techniques tend to produce better simplifications than vertex clustering but they are slower. These algorithms locate roughly planar regions on the surface of a model and retriangulate them with fewer polygons. The amount of simplification is controlled by setting a coplanarity threshold, which dictates how flat the region has to be before the region can be approximated by fewer polygons more closely approximating a plane.

1.1.3 Vertex Merge & Edge Collapse

Vertex merge and edge collapse approaches produce the best simplifications but they are the slowest of all the approaches. These algorithms simplify by determining the best pair of vertices to merge, they then merge them, compute a representative vertex, and repeat the process until the required model size is reached. Edge collapse algorithms are vertex merge algorithms with the constraint that only vertices that are connected by an edge are merged.

1.2 Motivation

As stated above, model simplification is the process of reducing the number of polygons in a model while preserving the shape and/or appearance of the original. Quantization is the process of taking a set of values and creating a smaller set of values to represent the larger set. These two processes are similar since both find a smaller set of elements to represent a larger set as optimally as possible.

By viewing simplification as a kind of quantization, it is possible to form a new taxonomy of existing model simplification algorithms. This quantization-based taxonomy also suggests possible avenues for the creation of new model simplification algorithms.

Most simplification algorithms trade speed for quality. Algorithms that produce good quality simplifications tend to be very slow and algorithms that produce low quality simplifications tend to be fast. The problem is that the good quality simplification algorithms are very slow with large input models because their complexity is $n \log n$. The fast algorithms have a linear complexity but produce very poor simplifications. Thus, there is a need for an algorithm that is:

1. fast,
2. the simplification speed should be linear with model size, and
3. produces reasonable quality simplifications.

1.3 Objectives

The goal of this research was to develop a simplification algorithm that is fast and produces reasonable quality simplifications. The algorithm had to fulfill the following criteria:

1. The complexity of the algorithm had to be linear or near linear so that it is able to simplify very large input models in a reasonable amount of time. For example, to simplify a model consisting of a million polygons should take a couple of minutes or less.
2. The algorithm will be used to do drastic simplification of large input models; on average the output model size will be one or two percent of the original size. Thus, there was an implicit suggestion that the simplification should be done in reverse; from coarse to fine.
3. Edges that are visually significant needed to be better preserved than in simplifications created by conventional vertex clustering algorithms, especially large features.
4. Disjoint components in a model should not be merged.

5. The computations in the algorithm should be simple and cheap since the emphasis of the algorithm is on simplification speed.

The solution was gleaned from quantization research. The result is a simplification algorithm that uses vertex clustering techniques to track vertex connectivity and uses *splitting* to simplify the model.

The following two sections, 1.3.1 and 1.3.2, briefly describe the algorithm and how its performance is evaluated.

1.3.1 A Quick Overview of R-Simp

The algorithm tracks vertex connectivity (edges) using clustering techniques similar to the ones described in Rossignac and Borrel[Ross93], but unlike [Ross93] it does not use this technique to simplify the model. Instead of uniformly subdividing the model's volume and clustering the vertices in each subdivision, the algorithm varies the size, orientation, and location of clusters based on the curvature of the surface. The algorithm begins with the entire model contained within a single cluster. Clusters are selected and split on the basis of the amount of curvature of the surface in the cluster. This process iterates until the desired number of clusters (vertices) is reached. Since the algorithm begins with the coarsest simplification and gradually increases its complexity, it simplifies in reverse; most other simplification algorithms start with a fine model and go to a coarse approximation.

1.3.2 Evaluation of Simplification Quality

The evaluation of the simplification quality was done qualitatively and quantitatively. To provide reference points for comparison the results from two existing algorithms were compared to R-Simp's simplifications, a vertex clustering algorithm similar to one proposed by [Ross93] and QSlim [Garl97]. The vertex clustering algorithm is extremely fast but produces very poor simplifications. QSlim is a vertex merge algorithm that is considerably slower but produces good simplifications. In terms of quality and speed R-Simp is positioned between these two algorithms, and thus there are reference points for comparison on either side.

Quantitatively, the algorithm was judged on its speed and geometrical quality. The geometrical quality was assessed by *Metro* [Cign97], a tool that geometrically compares two polygonal models and reports the maximum and mean geometrical error (difference). Qualitatively, the simplified models of all three algorithms were examined visually and informally to determine how well major features such as creases and surface protrusions were preserved.

1.4 Contributions

This research makes several important contributions:

- Two measures, called curvedness measures, were developed to approximate curvature at large scales. Most curvature measures from differential geometry work locally around a point on a surface. These measures are sensitive to the orientation change in large patches.
- The algorithm simplifies in reverse: from coarse to fine. This enables the algorithm to find coarse simplifications quickly.
- The simplification process is linear for a fixed output model size. The algorithm's complexity is $n_i \log n_o$, where n_i is the input model size and n_o is the output model size; most vertex merge and edge collapse algorithms have a complexity of $n_i \log n_i$.
- The algorithm is based on *splitting* from quantization literature. It is believed to be the first algorithm to use this quantization approach.

1.5 What Follows

In chapter 2, a brief introduction to quantization is presented and a quantization algorithm taxonomy is described. Chapter 3 presents the previous work in the area of model simplification. Several algorithms from each class of the quantization taxonomy are presented and their strengths and weaknesses analyzed. The chapter also presents several algorithms that preserve attributes on the surface of a model during simplification, several algorithms that create level of detail hierarchies, and several adaptive display algorithms. In chapter 4 a brief overview of curvature is given. Chapter 5 presents R-Simp, the simplification algorithm, and chapter 6 discusses the results obtained from comparing R-Simp to QSlim[Garl97] and to a simple vertex clustering algorithm. Finally, chapter 7 presents conclusions and possible future work.

Chapter 2

Quantization

Quantization is the process of selecting representative values for ranges of input values. The challenge is determining the representative values such that they optimally represent the entire range of input values.

The work in this thesis relies specifically on vector quantization. This chapter reviews the basics of vector quantization and several vector quantization algorithms are presented.

2.1 Vector Quantization

Vector quantization is the process of selecting representative vectors from $\mathcal{C} \subset \mathrm{R}^n$ for ranges of input vectors from $\mathcal{S} \subset \mathrm{R}^n$. A quantizer is a function $Q : \mathcal{S} \rightarrow \mathcal{C}$ where $\mathcal{C} = \{v_i \in \mathrm{R}^n \mid 1 \leq i \leq N\}$. $\mathcal{C}$ is called the *codebook*. The challenge is finding $\mathcal{C}$ such that it optimally represents $\mathcal{S}$. The codebook $\mathcal{C}$ partitions the set $\mathcal{S}$ since each v_i represents multiple vectors from $\mathcal{S}$. A single partition of $\mathcal{S}$ is called a *cell* and v_i is called a *centroid*. The difference between a vector v_i and an input vector u is called the *distortion*. When the distortion for all $u \in \mathcal{S}$ is minimal the codebook is considered optimal.

In [Gers92], Gersho and Gray present several quantization algorithms. Below, five of the algorithms are reviewed.

2.1.1 Product Codes

Product code algorithms can be viewed as multilevel hash tables. They use scalar quantizers that are applied to each element of the input vector; i.e:

$$Q(v_i) = \{q(v_{i_1}), q(v_{i_2}), ..., q(v_{i_n})\}$$

where $q(u)$ is a scalar quantizer and the centroid is usually the weighted mean of the elements in a cell. Unlike the other algorithms, those using *product codes* create codebooks with cells of uniform size. Most other algorithms dynamically vary the size of the cells to reduce distortion. This results in codebooks that are very suboptimal, but because the algorithms do not have to dynamically vary cell size they are quite fast.

2.1.2 Splitting

Splitting algorithms start with the codebook containing a single cell. The cell with the most distortion is located and split. The splitting continues until the required distortion or codebook size is reached. As in *product codes* the centroid is usually the weighted mean of the elements in a cell.

2.1.3 Pruning

Pruning algorithms work by removing unnecessary entries from the codebook or by adding elements to the codebook so that the distortion remains below a given threshold. When the algorithm works by removing unnecessary entries from the codebook, the codebook initially contains all the vectors in the input set $\mathcal{S}$. Then, the cells that increase the distortion least are removed. The number of entries that are removed depends on the required size of the codebook or the distortion threshold. Alternatively, the codebook is initially empty. Each input vector is considered in succession, and if representing a vector v with the current codebook results in distortion greater than a given threshold, the vector v is added to the codebook. *Pruning* algorithms do not need to compute the centroid because the elements in the codebooks after the quantization process are the centroids.

2.1.4 Pairwise Nearest Neighbour

Pairwise nearest neighbour algorithms work in reverse of *splitting* by merging pairs of cells. These algorithms also set the initial codebook to contain all the vectors in

S. All possible cell pairs are considered and the pair that would introduce the least distortion if merged is located. This pair is then merged to become a single cell. A new centroid is computed for the new cell, which is usually the weighted mean of the elements in the cell. Merging continues until the desired codebook size or distortion tolerance is reached.

2.1.5 K-Means

The K*-means* algorithm creates the most optimal codebooks, but due to its complexity it is to slow to be used. Also, there is a one to one mapping between the above quantization techniques and simplification algorithms; no such mapping is known for the K*-means* algorithm. This algorithm iteratively improves the codebook and the centroids. During an iteration a new centroid is computed for each cell and then the elements in the codebook are requantized, based on the new centroids, to minimize the distortion. This process continues until the centroid values converge. The K*-means* algorithm is the slowest of all the algorithms presented. Other algorithms are often used to provide a good initial guess as input to K*-means.*

2.2 Quantization & Model Simplification

Simplification relates to quantization as follows: A centroid equates to a primitive in the simplified model. For most vertex merge algorithms the centroid is the resulting vertex from the merging of two vertices. For vertex clustering algorithms the centroid is the representative vertex in a cluster. A cell equates to a collection of faces or vertices. In this thesis a cell equates to a set of faces, which is called a *patch*, in the original model.

There are a few ways in which model simplification differs from vector quantization. For example, on a three dimensional bunny model, a face from the ear and tail would be a poor cell. In a colour image of the same bunny, two pixels of similar colour, even if located on the ear and tail, make an acceptable cell. Thus, in model simplification, two disjoint faces do not make up an ideal cell, while in image quantization a cell with two widely separated pixels is perfectly acceptable.

Chapter 3

Previous Work

Much research has been done in model simplification. Model simplification deals with the problem of polygonal mesh simplification, with preserving colour and texture on model surfaces, and with creating level of detail hierarchies. This chapter will present work done so far in these areas with a strong focus on polygonal mesh simplification.

Model simplification can be viewed as quantization of three dimensional normal vectors on a two dimensional surface. Casting model simplification as quantization is convenient because there is a strong correlation between the quantization technique and the quality of simplification. Thus, quantization provides a good taxonomy for existing simplification algorithms.

3.1 Polygonal Mesh Simplification

3.1.1 Background

Polygonal mesh simplification is the process of reducing the complexity of a polygonal mesh by reducing the number of vertices, and hence the number of faces that make up the mesh. Any simple polygon may be used to define a surface; most simplification algorithms use triangles. This thesis assumes, without loss of generality, that all surfaces are composed of triangles because any simple polygon can be triangulated in linear time [Chaz91]. There are two main goals of mesh simplification, one, to produce a simplified mesh that best approximates the original, and two, to do this as quickly as possible. Unfortunately, these are opposing goals.

The problem of mesh simplification is cast as a vector quantization problem because quantization provides a good taxonomy for classifying existing algorithms based on the algorithm's simplification quality. Most simplification algorithms use one of three quantization techniques: *product codes*, *pruning*, or *pairwise nearest neighbour*. When applied to model simplification, *Product code* based algorithms are fast but tend to produce low quality simplifications. Model simplification algorithms that are based on *pairwise nearest neighbour* tend to be slow but produce better quality simplifications. Model simplification algorithms that use *pruning* are in between the other two techniques in terms of simplification speed and quality.

3.1.2 Algorithm Comparison Criteria

Several criteria are used to compare model simplification algorithms. The main two criteria are the quality of the simplification and the simplification speed. Simplification speed refers to the time it takes to process an input model and produce the simplified output model.

Quality refers to how similar the simplified model is to the original. Both quantitative and qualitative approaches are used to determine how well a simplified model approximates the original. Most quantitative approaches measure the distance between the original and the simplified surface; the smaller the distance the better the approximation. There are many different measures that can be used to compute this distance. To determine qualitatively how good the simplification is a human visually examines the simplified model. Currently computers are not able to qualitatively measure quality; this is an open research problem. An unbiased qualitative evaluation of several simplification algorithms would require a user study. We are not aware of any such study.

Preservation and/or the simplification of a model's topology are also important criteria to consider when comparing simplification algorithms. Intuitively, the topology of a model is the number of holes it has. In some applications it is desirable to close up small insignificant holes. In other applications, like medical imaging, it is important to maintain the topology and preserve all holes no matter how small.

The ability of an algorithm to accept non-manifold models as input and the ability of an algorithm to preserve the manifoldness of a surface are two additional characteristics that need to be taken into account when comparing simplification algorithms. A surface is considered to be a two dimensional manifold if the local topology is everywhere homeomorphic to a disc [Lueb99, Weis98]. A homeomorphism is a function that transforms a geometric figure into another geometric figure by an elastic deformation[1]. Intuitively, a manifold is a surface that cuts the space into two parts, such as a blanket. In a manifold triangular mesh every edge is shared by exactly two triangles and every triangle has exactly three neighbouring triangles. If the triangular mesh has a border then it is considered manifold if everywhere on the border the local topology is is homeomorphic to a half-disc; some edges will only belong to one triangle [Lueb99]. A border, or a boundary edge as it is sometimes referred to, is an edge that is only shared by one polygon. A surface that does not meet the above criteria for manifoldness is said to be non-manifold. It is advantageous if an algorithm can accept non-manifold surfaces as input because many models, when initially created, are non-manifold. Also, if the surface is manifold then the algorithm should preserve this characteristic because when a manifold surface becomes non-manifold severe visual distortions are usually introduced and some algorithms require manifold surfaces.

The final two criteria are whether the simplification algorithm can propagate surface attributes, like colour, texture, and vertex normals, from the surface of the original model to the new surface and if border edges are preserved. This is the basic set of criteria that is used to compare the model simplification algorithms presented in this chapter.

3.1.3 Summary of Simplification Algorithms

Table 3.1 summaries the algorithms reviewed in this chapter using the criteria presented in section 3.1.2. All the speed and quality comparisons presented in Table 3.1 are relative to the other algorithms listed in the table. Some algorithms are not rated in terms of speed, quality, or both because there was not enough data to rank them. To properly rank these algorithm their implementations need to be obtained and run on a single machine using the same model. This was not feasible in the scope of this thesis, but some rankings were obtained from various papers [Kalv96. Lind98b]. The algorithm by Rossignac and Borrel [Ross93] is considered to be the fastest and produces the worst quality simplifications of all these algorithms. Thus, the algorithms

[1]Webster's Electronic Dictionary

Class	Alg.	Spd.	Qu.	T.S.	T.O.	M.I.	M.O.	Attr.	Bdr.
Product	[Ross93]	> [Low97]	< [Low97]	√		√			
Codes	[Low97]	< [Ross93]	> [Ross93]	√		√			
	[He95]	< [Ross93]	> [Ross93]	√		√			
Pruning	[Kalv96]			√	√		√		
	[Hink93]				√		√		√
	[Turk92]				√		√		
	[Schr92]				√	√	√	√	
Pairwise	[Hopp93]	< [Hopp96]	> [Lind98b]		√		√		
Nearest	[Hopp96]	< [Lind98b]	> [Lind98b]		√		√	√	
Neighbour	[Garl97]	< [Ross93]	< [Lind98b]	√	√		∝	√	√
	[Lind98b]	< [Garl97]	> [Garl97]		√	√	√		√
	[Cohe97]	< [Garl97]			√		√	√	√
	[Ronf96]				≈		√		
	[Algo96]				√		√		

Legend

√	This characteristic is preserved.
∝	The algorithm can preserve or not preserve the characteristic. It is the user's decision.
≈	This characteristic is mostly preserved.
>[x]	The algorithm is faster or produces better simplifications than algorithm [x].
<[x]	The algorithm is slower or produces worse simplifications than algorithm [x].
Class	The class the algorithm belongs to in the quantization taxonomy.
Alg.	The algorithm being compared.
Spd.	The speed of the algorithm relative to the other algorithms.
Qu.	The simplification quality of the algorithm relative to the other algorithms.
T.S.	Can the algorithm simplify the model's topology.
T.O.	Can the algorithm preserve the model's topology.
M.I.	Can the algorithm accept a non-manifold model as input.
M.O.	Can the algorithm preserve the model's manifoldness.
Attr.	Can the algorithm preserve the attributes, such as colour and texture, on the model's surface and transfer them to the simplified model.
Bdr.	Does the algorithm preserve the borders explicitly, if they exist.

Table 3.1: Summary of existing model simplification algorithms.

that are not ranked are implicitly slower than [Ross93] and produce better quality simplifications than [Ross93].

3.1.4 *Product Code* Based Algorithms

Vertex clustering algorithms use *product codes* as the basis for simplification. These algorithms simplify by quantizing the vertices into codebooks whose cells are of predetermined size. The centroid is the representative vertex and it is usually computed as the mean of the vertices in a cell. A representative vertex is a vertex on the simplified model and represents a set of vertices from the original. Once the centroids are computed they are retriangulated to form the simplified model. A cell (cluster) is a collection of vertices from the original model. The quantizer is a simple function that maps the vertex into a cell by quantizing the vertex's coordinates independently; thus, $Q(v) = \{q(x), q(y), q(z)\}$. These algorithms are fast and produce low quality simplifications, because $Q(v)$ is usually simple distance function that largely ignores the surface's geometry and topology.

Rossignac and Borrel

Rossignac and Borrel [Ross93] developed one of the first vertex clustering algorithms. This algorithm is fast because it only has to do a linear pass through the vertices, but produces low quality simplifications. The model is read into memory and a list of vertices and a list of faces are created. The codebook is a three dimensional grid that uniformly subdivides the volume of the model's bounding box. The number of cells depends, proportionately, on the size of the simplified model; i.e. the larger the simplified model the more cells there are in the codebook. A pass is made through the vertices and each vertex is quantized into the cell at that location. Afterwards the centroid is computed for each cell and the model is retriangulated. Faces that end up having one or two vertices are usually thrown out. The centroid can be computed in several ways: by finding the mean of the vertices in a cell, by computing a weighted mean of the vertices in a cell, or by taking the vertex of greatest weight. The weight of a vertex represents its importance and may be computed in several ways; e.g. the surface area of the surrounding faces or the largest angle between two adjacent edges.

This algorithm can accept non-manifold vertices as input, but because it mostly ignores the geometry and topology during simplification it may create non-manifold surfaces which can add large distortions to the simplified model. This algorithm is able to simplify a model's topology, which in certain applications is desirable. It is extremely fast but produces poor quality simplifications. This algorithm does not take into account any additional attributes on the surface and it does not explicitly preserve border edges.

Low and Tan

Low and Tan [Low97] improved on [Ross93] by having vertex clusters form around important vertices rather than around locations in the model's bounding volume. Unlike [Ross93], the algorithm does not uniformly subdivide the model's volume. Initially the vertices of the model are weighted and sorted. The weighting is done using two criteria: the size of faces adjacent to the vertex and the probability of the vertex lying on a silhouette. The face size criterion is determined by finding the longest edge adjacent to the vertex. The probability of a vertex being on a model's silhouette is computed by taking the $\cos\left(\frac{\theta}{2}\right)$, where θ is the maximum angle, between any two edges adjacent to the vertex. 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 that 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 the output model.

This algorithm has several advantages over [Ross93]. First feature edges are better preserved because the cells are centred on vertices that make up the feature edges. A *feature edge* is a ridge or a crease in a surface that contributes a significant visual detail. Second, because the weighting of vertices is always the same, the difference between consecutive levels of detail is smaller since cells will always be positioned at the most important vertices. This algorithm is slower than [Ross93] because it needs to initially weight and sort the vertices, which is an $n \log n$ operation, but it produces better simplifications than [Ross93]. In terms of the other comparison criteria, it is equivalent to [Ross93].

He et al.

He et al. [He95] 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, 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 very 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. To reduce distortion due to the simplification higher detailed translucent meshes are then added on top of the simplified surface.

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 [Ross93] but produces better results. Its abilities with respect to the other comparison criteria are equivalent to [Ross93].

3.1.5 *Pruning* Based Algorithms

The next class of simplification algorithms use the *pruning* quantization technique. When applied to polygonal mesh simplification, *pruning* techniques generally produce better codebooks than *product codes*, and therefore the algorithms that employ them produce better simplifications. Unfortunately the improved quality has a price; because the *pruning* technique is more sophisticated than *product codes*, it is more expensive to compute, and therefore the simplifications take longer to produce than simpler *product codes*. These simplification algorithms work by locating relatively planar regions in a model, approximating the relatively planar regions with a plane, and then retriangulating the area with fewer polygons. Some of these algorithms have trouble simplifying models that contain many sharp edges and few planar areas.

Superfaces

The Superfaces [Kalv96] 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 superfaces are created or the codebook is populated; initially the codebook is empty. Faces are added to the codebook in the following way. A face is added to an existing cell if the face is, 1), 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 it starts a new cell. The process stops when all the faces in the original model have been added to the codebook.

Several criteria are used to determine relative coplanarity. The first criterion is the distance of the superface to the original surface. This distance is determined by computing the maximum distance between the centroid and the vertices of the faces in the cell. The centroid is the average plane of all the faces in a cell. If adding the face to the cell causes the distance threshold to be exceeded then the face is not added. Another criterion is the face-axis rule that states that the angle between the normal of the centroid and the normal of the face can not exceed a certain threshold angle. The last criterion is the no fold-over rule. To be able to retriangulate the superface no faces are allowed to fold onto the superface (in general this is an expensive check and so it is not used when a superface is initially constructed). The terms *fold* or *fold-over* describe the change in orientation of two planes; if the angle between planes p_1 and p_2 was greater than 90 degrees and then the angle became less then 90 degrees then it is said that p_1 or p_2 has *folded* onto p_2 or p_1. Once the superface is completed it is checked for fold-over faces. If a fold-over face is found the patch is regrown with the third criterion enforced. An optional criterion that can be used is called the Gerrymandering check that ensures that the superfaces do not get too thin or too long.

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 if all the vertices in the original boundary are within a certain distance of the superedge.

In the third stage the superfaces are retriangulated. The final codebook is a set of planar patches that completely cover the original surface.

An advantage of this algorithm is that it is on average linear in the number of faces in the model. Although the average performance of the algorithm is linear, the processing that needs to be done for each face gives it $n \log n$ complexity in the worst case. This algorithm is significantly slower than [Ross93] but produces significantly better simplifications. It can not accept non-manifold surfaces as input but it preserves both the topology and the manifoldness of a model. Also, it does not handle additional surface attributes, such as colour and texture, and it does not explicitly preserve boundary edges.

Hinker and Hansen

An algorithm similar to [Kalv96] was proposed by Hinker and Hansen [Hink93]. The algorithm constructs near-coplanar sets of faces and then retriangulates them with fewer polygons. As in [Kalv96] the codebook is initially empty. A face, $\mathcal{F}_o$, with normal $\mathcal{N}_o$, is selected as a starting element for a cell $\mathcal{C}_o$. Faces are added to $\mathcal{C}_o$ as long as they are adjacent to the patch in the cell and do not cause the mean normal of the cell to exceed a user specified angular threshold between it and $\mathcal{N}_o$. The process iterates until all the faces have been placed into the codebook.

Once all the faces have been added, the patch in each cell is examined to determine if it has any holes (see Figure 3.1). If a hole is detected the faces that make up the patch are adjusted so that the hole remains undisturbed. The patch is then retriangulated.

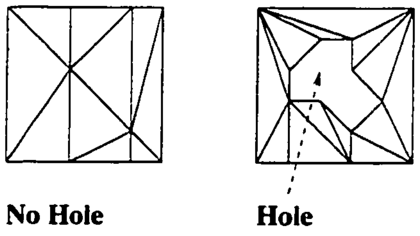


Figure 3.1: A patch with and without a hole.

This algorithm is faster than [Kalv96] because the decision metric used to decide if a face is to be added to a cell is simpler compared to the one used in [Kalv96]. This algorithm works well on models that contain many flat surfaces, but it does not work well on surfaces that have a great deal of high frequency curvature such as terrains of mountain ranges. This algorithm is easily parallelizable because once a patch has been located no external information to the patch is required to perform the other two stages. Its quality is comparable to that of [Kalv96]. It does not simplify a model's topology and explicitly preserves a model's borders. It can not accept non-manifold surfaces, but it preserves the manifoldness of a surface during simplification. This algorithm does not preserve additional surface attributes.

Turk

Turk [Turk92] 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. The curvature measure is the smallest angle between the vertex normal and an adjacent edge. The area is the area of a face. A new vertex is more likely to be positioned in faces that are bigger and/or have high curvature at their vertices; it is possible that a face may receive several new vertices. A global relaxation technique is then applied to the new vertices. Each vertex repels the other vertices in its local area forcing the vertices to be evenly distributed on the surface. To speed up the process the repulsion forces drop of linearly with respect to distance and become zero at a fixed radius. The repulsion force in areas of high curvature is scaled by the inverse amount of curvature so that the vertices repel less and therefore areas of high curvature will contain more vertices. Next, a mutual tessellation is performed on the surface. Mutual tessellation involves the retriangulation of faces to include the new vertices (see Figure 3.2).

Lastly, the original vertices are removed. This involves taking an original vertex R and all vertices that share a triangle with R, call this set T, and projecting them onto a plane that is tangent to the surface at R (see Figure 3.3).

Several checks are performed to ensure that if R is removed the new surface will remain valid. The first check ensures that the resulting surface around R will not contain any folds because otherwise unwanted artifacts are created. The second check, ensures that two surfaces will not be joined (preserving manifoldness). For example,

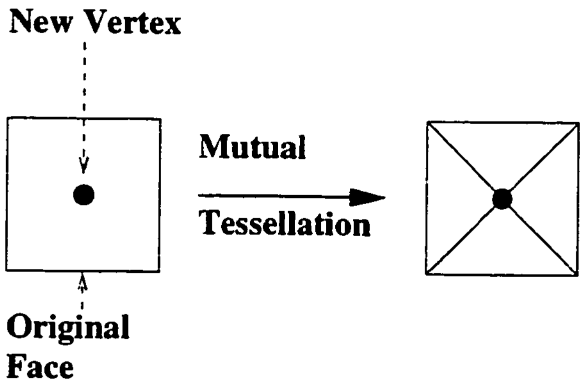


Figure 3.2: Mutual Tessellation

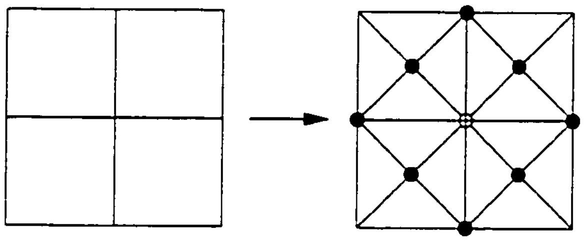


Figure 3.3: Removal of original vertices. Key: Vertex ● $\in T$ and Vertex ○ $\in R$.

removing the top vertex of a tetrahedron will collapse it, joining two surfaces. If these two checks succeed then R is removed and the vertices in T are retriangulated with the restriction that no new edges are to be created. When all possible original vertices are removed the result is a simplified model. Initially the codebook starts off with just the new vertices. In the original vertex removal stage any vertex that can not be removed is added to the codebook. This algorithm can also create multiple levels of detail and interpolate between them, an aspect that is discussed later.

This algorithm performs well on surfaces that have smooth curves, and which are mostly continuous. According to Turk, this algorithm does not work as well on surfaces that contain many edges and sharp corners, e.g. man made objects. One aspect of model simplification that [Turk92] does not handle is additional vertex information such as colour and texture. Since [Turk92] introduces a complete new set of vertices it becomes difficult to propagate additional vertex information from the original to the simplified model. No simplification speed results were presented and therefore it is difficult to say how fast this algorithm is, but its quality is comparable to [Kalv96]. This algorithm requires that the input models be manifold and it also preserves topology. The borders are not explicitly preserved.

Schroeder et al.

Schroeder et al. [Schr92] developed an algorithm that simplifies the model by pruning away the "least important" vertices. Initially the codebook is initialized with the vertices from the original model. In the first pass, the algorithm classifies all the vertices based on their local geometry and topology. A vertex is classified into one of five classes: A vertex is classified as simple if it is manifold and not a boundary edge, and complex if it is non-manifold. If a vertex is part of a boundary then it is considered a boundary edge. Furthermore, if a simple vertex has two adjacent faces that have a dihedral angle smaller than some specified threshold angle then the vertex is further classified as part of an interior edge. If there are three adjacent faces that have a dihedral angle smaller than some specified threshold angle then the vertex is classified as a 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 and the area of all adjacent faces is retriangulated. The mean plane is constructed using the vertex's adjacent face normals weighted by the face area and the mean of all the face centres. If the vertex is a boundary vertex or an edge vertex, then the threshold is compared to the distance from the vertex to the the new edge that would be created if this vertex were to be removed. Usually, corner vertices are not eliminated, but if the corner is not a significant visual detail then the distance to plane criteria is used to decide whether to remove it. The decision to remove corner vertices is left up to the user. For each vertex removed, additional checks are made to ensure that topology is preserved. The process stops when either no more vertices can be removed, according to the given distance threshold, or the number of vertices for the simplified model has been attained.

The simplification quality of this algorithm is comparable to that of [Turk92], but it is unclear how fast it is relative to the other algorithms in its class, because no speed data was published. This algorithm can accept non-manifold models as input and it preserves topology. Since the new model consists of vertices from the original, all the vertex information is preserved. It is not evident that additional face information is preserved or can be preserved. This algorithm also does not explicitly preserve borders.

3.1.6 *Pairwise Nearest Neighbour* Based Algorithms

The last class of polygonal mesh simplification algorithms use the *Pairwise nearest neighbour* approach of quantization.

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 a polygonal mesh. These algorithms start with the codebook 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. A simple distortion measure is the distance between the vertices, but it is not often used because it does not take into account such things as the curvature of the surface. These algorithms usually have a complexity of $n \log n$.

Many algorithms add an additional constraint that only vertices that are connected by an edge are merged. This constraint is useful because it helps to preserve feature edges by guiding the simplification along these edges rather then across them. The constraint also helps to preserve topology.

The algorithms discussed below all simplify in the same way. The primary differences between them are how they compute the distortion between cells (the cost to merge the cells) and how they determine the centroid. In all these algorithms a centroid is the representative vertex. A representative vertex is a vertex in the simplified model and represents several vertices from the original.

Hoppe et al.

One of the first algorithms that used this simplification technique was developed by Hoppe et al. [Hopp93]. This algorithm minimizes an energy function to compute the distortion and the centroid. 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 constrains 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.

The energy function used to compute the distortion and the centroid takes into account several characteristics of the surface and consists of three parts:

$$\mathcal{E}(K, V) = \mathcal{E}_{dist}(K, V) + \mathcal{E}_{rep}(K) + \mathcal{E}_{spring}(K, V) \tag{3.1}$$

In equation 3.1 K represents the topology, the connectivity of the surface, and V represents the geometry, the location of the vertices. $\mathcal{E}_{dist}(K, V)$ is equal to the sum of squared distances from the vertices of the new surface to the original surface. $\mathcal{E}_{rep}(K)$ is a penalty function that penalizes meshes with large number of vertices. Since the algorithm allows cells to be added and removed, there needs to be a term that forces the cell count to decrease, otherwise cells would be added indefinitely. It is not enough to minimize $\mathcal{E}_{dist}(K, V)$ and $\mathcal{E}_{rep}(K)$ because a minimum might not exist, so a third term, $\mathcal{E}_{spring}(K, V)$, is added to the equation to help guide the simplification and to reach a desired local minimum. The term $\mathcal{E}_{spring}(K, V)$ is a spring energy term that places a spring of rest length zero on every edge.

The algorithm randomly selects an edge from the set R and applies one of the three transformations. 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 lower 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 location of the centroid is computed by minimizing the $\mathcal{E}_{dist}(K, V)$ and $\mathcal{E}_{spring}(K, V)$ terms of the energy function.

The simplifications produced by [Hopp93] are better than the simplifications produced by the algorithms already presented, but it is one of the slowest algorithms. The algorithm preserves the model's topology but no care is taken to preserve additional surface information. This algorithm requires that the input model be manifold and it preserves the manifoldness of the surface. In the algorithm described in the next section the same author(s) simplify and extend this algorithm to provide the ability to create levels of detail hierarchies and preserve additional vertex information.

Progressive Meshes

The approach taken by Hoppe in [Hopp96] is similar to [Hopp93] but has several important differences. The algorithm still uses and minimizes an energy function to compute the distortion, but it contains several new terms to account for additional values at the vertices. Instead of random selection a priority queue is used to order edge transformations, which has the benefit that all edges that cause the least amount of distortion are collapsed first. The number of edge transformations has also been reduce to one: edge collapse. Simplification quality is not appreciably changed. The computation of the distortion becomes simpler because only one transformation needs to be considered. The new energy function changes slightly from equation 3.1 to:

$$\mathcal{E}(M) = \mathcal{E}_{dist}(M) + \mathcal{E}_{spring}(M) + \mathcal{E}_{scalar}(M) + \mathcal{E}_{disc}(M) \tag{3.2}$$

The first two terms, $\mathcal{E}_{dist}(M)$ and $\mathcal{E}_{spring}(M)$, are identical to those in equation 3.1. The last two terms are added to preserve vertex attributes associated with mesh M.

This algorithm produces comparable quality to [Hopp93] and is faster than [Hopp93], but it is still considerably slower than many other simplification algorithms. This algorithm also tries to preserve attributes associated with each vertex and it tries to correct any discontinuities that may happen as the surface is simplified. This algorithm preserves topology and requires the input model to be a manifold surface. It does not explicitly preserve boundary edges.

QSlim

The algorithm developed by Garland and Heckbert [Garl97] is considerably faster than those described in [Hopp96, Hopp93]. The algorithm merges vertices that are connected by an edge or are within a certain threshold distance. The goal of [Garl97] is to find and merge a pair of cells such that the resulting centroid (representative vertex) will be as close to the original surface as possible. Two cells will not be merged if the merge will cause a face to fold over onto another face.

Distortion in [Garl97] is a distance measure that measures the sum of squared distances from a vertex v to a set of planes (similar to [Ronf96]). 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. When two vertices are merged their associated plane sets are also combined.

Equation 3.3 computes the sum of the squared distances from v to its associated set of planes.

$$\Delta(v) = \Delta([v_x, v_y, v_z]^T) = \sum_{p \in planes(v)} (p^T v)^2 \tag{3.3}$$

where $p = [a, b, c, d]^T$ represents the plane defined by the equation $ax + by + cz + d = 0$.

The above equation can be rewritten as follows:

$$\begin{aligned} \Delta(v) &= \sum_{p \in planes(v)} (v^T p)(p^T v) & (3.4) \\ &= \sum_{p \in planes(v)} v^T (pp^T) v & (3.5) \\ &= v^T \left(\sum_{p \in planes(v)} pp^T \right) v & (3.6) \end{aligned}$$

Thus, the distortion measure is:

$$\Delta(v) = v^T Q v \tag{3.7}$$

where

$$Q = \left(\sum_{p \in planes(v)} pp^T \right) \tag{3.8}$$

Q represents the set of planes for every cell. This is convenient because when cells i and j are merged their respective Q's are added, $Q_{new} = Q_i + Q_j$. The distortion measure favours cells that, when merged, produce a centroid 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.

The computation of the centroid in [Garl97] is cast as a minimization problem. The algorithm finds the centroid that is closest to all the planes in Q. This happens when $\partial\Delta/\partial x = \partial\Delta/\partial y = \partial\Delta/\partial z = 0$; this is a linear problem and is quick to solve.

Boundary edges and vertices are handled by placing a perpendicular plane at the edge and weighting the distortion with a large penalty factor; fold-overs are handled in a similar fashion.

This algorithm is faster, by several orders of magnitude, than [Hopp96, Hopp93]. Its simplifications are usually not as good, but are considerably better than that of [Ross93]. It can simplify well those surfaces that are simplified poorly by *pruning* based algorithms. A big advantage of this algorithm 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 topology will be preserved, or by setting it to a value greater than zero topology will not be preserved. This algorithm explicitly preserves borders but does not preserve additional surface information. Also, it expects a manifold model as input.

Turk and Lindstrom

The algorithm by Turk and Lindstrom [Lind98b] is fast and memory efficient. Like [Hopp96, Hopp93] it constrains the vertex merges to vertices with edges between them. Unlike many simplification algorithms, 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 then the border may change in length and the local area it surrounds may change. This algorithm tries to minimize both of these changes. The centroid is an optimization problem that tries to minimize the change in volume and area. The distortion computation is based on the change in volume and area computed during the centroid computation.

The optimization problem combines and solves several linear equality constraints. Ideally only three constraints are needed, in circumstances when two or more constraints are linearly dependent additional constraints are added in a predetermined order of importance. The first constraint is volume. The centroid should be placed such that the change in volume is minimal because otherwise the surface will usually deform. This constraint may not exist if the surface is not manifold in the local area of the vertex merge. The next two constraints are used to maintain boundaries (borders), if they exist. This is accomplished by preserving the local area enclosed by the boundaries. If these constraints are not enough, volume and a boundary optimizations are added that involve the minimization of an objective function. Finally, if the constraints still do not provide a single solution, a constraint is added that optimizes the triangle shape. The volume and boundary optimizations are always computed because they are used in the distortion computation.

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 algorithms, simplification stops when the required size of the model is reached or no more edges can be collapsed.

This algorithm is faster than [Hopp96] but slower than [Garl97]. Also, the simplification quality is in between [Garl97] and [Hopp96]. The algorithm preserves topology and is able to accept non-manifold models as input. It explicitly preserves borders, but it ignores any additional attributes. such as colour and texture, that are on the original surface.

Cohen et al.

Cohen et al. [Cohe97] present an algorithm that bounds surface deviation between a simplified surface and the original, and it also minimizes the surface deviation between consecutive levels of detail. A simplified model of size n is considered to be a specific *level of detail* of an original model. This algorithm also restricts the vertex merges to those vertices that are connected by an edge. The algorithm selects the edge that introduces the least amount of distortion (surface deviation), if collapsed, and collapses it. The process iterates until the required model size or maximal surface deviation is reached.

The distortion is based on the surface deviation, which is computed by enclosing every simplified triangle, and its matching face(s) on the original model, with an axis aligned bounding box. The larger the bounding box, the greater the deviation, and hence the distortion. Initially the bounding boxes have no volume.

To determine the centroid location, the algorithm analyzes the cells to be merged, and the triangles that are adjacent to the centroids of the cells. The centroids and the adjacent triangles in the cells to be merged are projected onto a plane. A check is made to make sure that no faces have folded over onto the plane. Then the centroids are merged and a new vertex position is found using a linear programming approach. The location of the new vertex cannot make any faces fold over onto the plane. The new vertex is then moved in the direction of the projection to find the position where this new surface deviates least from the surface created by the two cells that are being merged. Once the position is located the new vertex becomes the centroid of a new cell.

The algorithm maps texture coordinates from the original model to the simplified model by taking a vertex v on the simplified model, finding the corresponding texture value on the original surface, and mapping this value onto v.

This algorithm is slower than [Garl97] but produces better results. The input model has to be manifold, but the algorithm preserves topology and texture information that is on the original model. This algorithm does not do any explicit preservation of border edges.

Ronfard and Rossignac

Ronfard and Rossignac [Ronf96] developed an edge collapse algorithm that uses an interesting approach for computing the centroid and the distortion. The distortion is the maximum of two measures. The first measure measures the amount of orientation change a face has gone through after an edge collapse. The second measure computes the maximum distance between the centroid of a new cell and the set of planes associated with the cell. This latter measure was later used in QSlim [Garl97, Garl98].

The first distortion measure is a penalty function f_p that ensures that the mesh remains unfolded and smooth. When an edge is collapsed, the faces in the neighbourhood of the collapsed edge change orientation; f_p measures this change: $f_p(V_1, V_2) = K \max_i A_i$, where K is a constant that is sufficiently large so face flips are heavily penalized, and A_i is the angle between the face normal i before a merge and the face normal i after a merge.

The second distortion measure is a is a geometric error function f_e that measures the distance of vertices as they move perpendicular to the original surface. Every vertex in the original model has a set of adjacent faces. Initially the distance from a vertex to its planes is zero. When cells are merged their plane sets are combined and the distance from the centroid to each of the planes is computed. The geometric error function f_e is the maximum of these distances.

Initially the location of the centroid is selected as one of the positions of the two vertices being merged; e.g. $V_c' = V_2' \leftarrow (V_1', V_2')$. V_c' is used by f_e. After the merge, the location of the centroid V_c' is optimized by applying the quadratic vertex placement technique already described for [Garl97]. Ronfard and Rossignac [Ronf96] acknowledge that it would be more elegant to include this optimization into the distortion computation.

This algorithm is slower than [Ross93], but it is unclear what is its speed relative to the algorithms in its class since speed results were not published. Its quality seems to be comparable to the other edge collapse algorithms. This algorithm only accepts manifold surfaces as input. It is able to simplify topology and it is unclear if it preserves the manifoldness of a surface since there seems to be a possibility of faces flipping, even if K is set sufficiently high. This algorithm does not explicitly preserve borders and it does not preserve additional surface attributes.

Algorri and Schmitt

The algorithm developed by Algorri and Schmitt [Algo96] combines the *pruning* with the *pairwise nearest neighbour* approaches. This algorithm simplifies an object in four steps. In the first step, the entire model is scanned and edges are swapped if the result will flatten and smooth the mesh. The second step classifies all the vertices. A vertex is classified by the number of feature edges adjacent to it. A vertex in a planar region has a classification value of zero. A vertex lying on a feature edge has a classification value of two, and a corner has a classification value of three. Next, these classification numbers are used to create clusters. Clusters with classification value of zero are simplified first because these are roughly planar surfaces. Within each cluster the simplification is accomplished by applying a *pairwise nearest neighbour* based algorithm. The distortion is the edge length and the centroid is the mean of the centroids of the cells being merged. This is a fast and easy approach but it has drawbacks. Potential edge collapses are not performed because the mean position might cause a face to fold over, this is not permitted and other possible locations are not examined. When all the zero valued clusters have been simplified the edge (two valued) clusters are analyzed. If the angle between two feature edges that join at an edge vertex is less than a certain threshold then the vertex is slid along the feature edge and placed on one of the existing vertices on the feature edge; i.e. $V_2 \leftarrow (V_1, V_2)$. The last step involves removing isolated vertices. These vertices are left in the middle of a planar region because cell merges always leave one vertex in a planar cluster. These vertices are removed and the cluster is retriangulated.

It is difficult to tell how fast this algorithm is relative to the other algorithms that use the *pairwise nearest neighbour* approach because speed results were not published. This algorithm is slower than [Ross93] and produces comparable results to the rest of the algorithms in the *pairwise nearest neighbour* class. The algorithm expects to have a manifold model as input and it preserves the topology of the model. It does not explicitly preserve borders and it ignores all additional surface attributes.

3.2 Preservation of Colour, Texture, and Other Attributes

In section 3.1 several algorithms were presented that reduce the complexity of a polygonal mesh. Many of these algorithms simplify the geometry and ignore the distortion caused to attributes associated with the surface such as colour, texture, and vertex normals. Some of the algorithms described above try to solve this problem, but generally this is difficult because model geometry and attributes, usually do not coincide. These algorithms use linear interpolation to determine the correct attribute value on the new surface [Cohe97, Hopp93]. It is not always feasible to use linear interpolation because certain values, like colour and texture coordinates, are not linear. That is, if vertex V_1 has attribute a_1 and vertex V_2 has attribute a_2, it is not generally correct to compute the attribute for merged vertex V_{12} as $a_{12} = \frac{a_1+a_2}{2}$.

Garland and Heckbert

Garland and Heckbert [Garl98] proposed an extension to their original algorithm [Garl97] that takes into account vertex attributes. This algorithm uses the exact same approach to simplification but extends the dimension of the quadratic error measure to include colour, texture, and vertex normal attributes. For example, for colour attributes, centroids would become (x, y, z, r, g, b) vectors. This algorithm makes the assumption that all attributes are linear and that their distance metrics are Euclidean. This is not true for colour and other attributes, but it simplifies the problem and allows the attribute distortion measure to be integrated into the overall distortion computation. If each face were to have a different texture, this algorithm would break down, but for sufficiently uniform surfaces it works well. Moreover, the algorithm is extensible to any number of attributes at a vertex, because additional attributes only increase the dimension of the quadratic error matrix.

Cohen et al.

An algorithm developed by Cohen et al. [Cohe98] extended their earlier work [Cohe97] to preserve vertex normal vectors, colour, or texture attributes. Although [Cohe97] preserved texture attributes, it did it in a very simple way that allowed a strong possibility of texture distortion. This algorithm does a better job of preventing texture distortion by measuring the amount the texture attributes have slid across the surface it then tries to minimize the sliding.

The algorithm begins by creating a (u, v) parameterization of the surface and an attribute map $\mathcal{A}$, such as a normal map, colour map, or texture map. If the surface already has an attribute map the first step is skipped. The distortion is the Euclidean distance between a point on the simplified surface and a point on the original surface that have the same coordinates (values) in map $\mathcal{A}$. It seems that the algorithm can only preserve one attribute at a time. This deviation measure is sufficient because the attributes are bound to the surface and so the movement of these properties corresponds to the movement of the surface and thus the surface deviation is also minimized.

Since vertex normals, colour, and texture attributes are not assumed to be linear, and the differences are not measured with a Euclidean distance measure, the problem of attribute distortion is greatly reduced.

Hoppe

Hoppe [Hopp96], as part of his algorithm, developed a way to preserve other mesh attributes. As part of the energy function used to compute distortion, there is a term that computes distortion for mesh attributes, called $\mathcal{E}_{scalar}$. The $\mathcal{E}_{scalar}$ function treats these attributes as linear quantities and tries to minimize the distortion due to a merge. As in [Garl98], problems can arise due to the fact that these quantities are treated as if they were linear. The algorithm will work fine on surfaces that have a sufficiently continuous texture or colour spread, but not on surfaces that have many high frequency texture or colour changes.

Cignoni et al.

Cignoni et al. [Cign98] proposed an approach to mesh attribute preservation that separates the attribute preservation step from the simplification step, rather than tackling these two steps at the same time.

Instead of remapping attribute values from the original to the new surface, this algorithm creates a texture that contains the attribute values and maps it onto the new surface. The input consists of two models, a full resolution model M with all the mesh attributes, and the simplified model S containing only the simplified mesh of M. The output is the simplified mesh S with a (u, v) parameterization and an attribute map.

The procedure to create the attribute map is straight forward: The user sets a sampling rate which is used to sample the faces in the simplified surface S. For each sample point $p \in S$ a point $p_m \in M$ is located. The point p_m is the closest point to p in terms of Euclidean distance. Once point p_m is obtained its corresponding face in M is found and the attributes are computed by interpolating the values found at the vertices. Once all the attribute values have been computed for a face in the simplified model an attribute map is created for that face. The attribute maps for all the faces are then merged into one large attribute map for the entire model.

This algorithm has several advantages; one being that it can be extended to convert procedural textures (defined as functions, not images). The user can also control the amount of mesh attribute detail by setting the sampling rate. Linear interpolation is minimized to a face on the original model rather than across multiple faces; this reduces distortion. The algorithm is fast and because it is decoupled from the simplification stage, the simplification component can be chosen independently.

3.3 Level of Detail

Interactive environments require several levels of detail for the objects they are displaying for maximum display efficiency. For example, a user who sees a car one meter away should see the door handles while a user who is 50 meters away does not necessarily need to see the handles to recognize that the object is a car. Hence, there is a need for two things. First, level of detail hierarchies need to be created to enable moving from one level of detail to another without jarring visual discontinuities; and second, a method is needed to determine the best level of detail for an object for a particular viewing distance and angle. These two tasks are usually handled by different applications. The creation of the level of detail hierarchies usually falls to the simplification algorithms because many can create them as the simplification progresses. The control of the level of detail is handled by adaptive display algorithms.

3.3.1 Adaptive Display Algorithms

In an interactive environment, frame rates must be high to maintain the illusion of reality. Each environment might have several objects that need to be displayed in each frame, and the detail of each object should vary depending on the user's viewpoint. The job of an adaptive display algorithm is to manage this variation of detail. Adaptive display algorithms usually do not perform mesh simplification, rather they are given the objects at different discrete levels of detail as input. This is known as "static" level of detail control.

Funkhouser and Sequin

Most virtual environments vary wildly in model complexity: one portion may be very detailed, another portion quite simple. The goal of Funkhouser and Sequin's [Funk93] algorithm is to keep the frame rate constant in such discontinuous environments. Other algorithms [Hopp97, Lueb97] have trouble in these environments because they determine the level of detail they can display by the time taken to display the previous frame, assuming that if N polygons were displayed in the previous frame, then they will display $N \pm \epsilon$ in the upcoming frame. This is not always true. Instead of relying solely on the previous frame, Funkhouser also computes the maximum allowable level detail for each object for each frame, using various heuristics.

Luebke and Erikson

Luebke and Erikson [Lueb97] designed an algorithm that controls the level of detail of entire scenes, rather than the level of detail of individual objects in the scene. This algorithm uses a vertex tree to do the dynamic level of detail selection; this is known as "dynamic" level of detail control because much of the displayed detail is determined at runtime. A vertex tree is a hierarchical structure that contains the entire scene and the graph of the simplification process. The leaves contain the entire scene at full detail and moving up in the tree simplifies the scene.

Hoppe

Hoppe [Hopp97] developed an algorithm that uses the output of progressive meshes [Hopp96] to create an adaptive display algorithm. The algorithm is similar to that of [Lueb97].

3.3.2 Creation of Level of Detail Hierarchies

Level of detail hierarchies are most often created during the model simplification process. A level of detail hierarchy is a log of all the local simplifications that were performed during the simplification process. The key to creating levels of detail hierarchies is that there needs to be a sufficiently fine granularity between levels of detail so that a switch from one level to the next does not cause visible discontinuity. Level of detail hierarchies have been created by all types of simplification algorithms and a stand-alone algorithm [Cohe96] has also been developed.

Any simplification algorithm can be run n times to create n levels of detail. However, this approach is inefficient, and in some instance the levels of detail contain too many discontinuities between them.

Algorithms that are based on *product codes* do not inherently build level of detail hierarchies. However, these algorithms can easily be extended to do so by adding an oct-tree and simplifying the model at each level of the tree. This type of level of detail hierarchy corresponds to a vertex tree and can be used with minor modifications with adaptive display algorithms that use vertex trees.

Algorithms that are based on *pruning* do not lend themselves explicitly to the creation of level of detail hierarchies except the algorithm by Turk [Turk92]. The [Turk92] algorithm builds level of detail hierarchy by incrementally adding vertices at the retiling phase and saving the result. The results of this algorithm could be used in the adaptive display algorithm described in [Funk93].

Algorithms that use *pairwise nearest neighbour* are well suited for creating level of detail hierarchies. To create a level of detail hierarchy all that is required is to record the order of merges.

Cohen et al.

Cohen et al. [Cohe96] developed a framework to create static levels of detail. The main focus of the work is to have the ability to use any algorithm and to ensure that the distortion between levels of detail is smaller than a specified threshold. Level of detail is controlled by the amount of deviation that is allowed from the original surface. The framework drops an envelope (interior/exterior "images" of the surface at a distance ϵ) around the original surface and the simplification algorithm must work within this envelope. A legal move is one that keeps the surface within the envelope and keeps the surface manifold. Although this scheme slows down the simplification algorithm, feature areas are better preserved, and if the error is finely controlled then the difference between two closely adjacent levels of detail is small.

3.4 Conclusions

This chapter has presented and compared several simplification algorithms. Several conclusions can be reached from the comparison of these algorithms.

Product codes based simplification algorithms are the fastest but usually produce the worst simplifications. Vertex clustering algorithms are well suited for creating previews of very large models. These algorithms are not well suited for simplifying models that have many high frequency details. Applications that require medium to high quality simplifications and do not require high speed should use simplification algorithms that are based on *pruning* or *pairwise nearest neighbour.*

Simplification algorithms that are based on *pruning* usually produce considerably better results than vertex clustering algorithms but they are slower. These algorithms produce medium to high quality simplifications, but they are limited in the types of surfaces that they can simplify well. They do not work well on surfaces that contain a great amount of high frequency noise such as terrain maps.

Algorithms that are based on the *pairwise nearest neighbour* approach tend to produce the best simplifications. Although these algorithms are the slowest, they work well for all types of surfaces. Any application that requires high quality simplifications should use this sort of algorithm.

Chapter 4

Curvature

Curvature describes the manner in which surface orientation changes across a surface. Most of the algorithms discussed in chapter 3 use approximations of curvature to guide simplification. In effect, these algorithms are quantizing surface orientation. This thesis will explicitly adopted an orientation quantization approach. In this chapter, the basic elements of curvature are reviewed.

4.1 Importance of Curvature

Determining curvature is important because it provides information about the shape of a surface. Given a simple polytope, like a tetrahedron, it is possible to determine its shape by the location of vertices. It is much more difficult to determine the shape of a more complex surface from vertex location alone. By determining the curvature of a surface, it becomes easier to characterize the surface's shape.

4.2 Normal Curvature

One common measure of surface curvature is normal curvature [O'ne72]. Normal curvature is based on the rate of change of the normal vector field U on a surface S in direction u, where u is a unit vector tangent to the surface S at point p. If the normal vectors are pointing towards p then normal curvature is negative, otherwise it is positive. Thus, normal curvature is sensitive to concavities and convexities. There are two important normal curvature extrema called *principle curvatures*; these are the maximum (k_1) and minimum (k_2) values of normal curvature. The directions corresponding to these principle curvatures are called *principle directions*. k_1 and k_2 are used to define two additional curvature measures that describe how the surface is behaving in the local area surrounding p.

The first measure, called *mean curvature* (K_a), is computed by taking the mean of k_1 and k_2, $K_a = \frac{k_1+k_2}{2}$. If $K_a > 0$ then the surface is an elliptical bump and if $K_a < 0$ then the surface is an elliptical hollow. If $k_1 \approx k_2$ and $K_a \approx 0$ then the surface is relatively planar. The magnitude of K_a indicates the sharpness of the elliptical bump or elliptical hollow.

The second measure, called *Gaussian curvature* (K_g), is the product of k_1 and k_2; $K_g = k_1 k_2$. If k_1 and k_2 have the same sign, then $K_g > 0$ and the surface is an elliptical bump or an elliptical hollow. If k_1 and k_2 have opposite signs, then $K_g < 0$ and the surface is hyperbolic, a saddle point. If $K_g = 0$ then the surface is planar or it is a cylindrical ridge or hollow.

By combining these two measures it is possible to characterize the shape of the surface around point p (see Table 4.1).

Surface	**k_1 and k_2**	**K_a**	**K_g**
Elliptical Bump	same sign	> 0	> 0
Elliptical Hollow	same sign	< 0	> 0
Hyperbolic	different signs	≈ 0	< 0
Cylindrical Ridge	$\approx 0, < 0$	> 0	≈ 0
Cylindrical Hollow	$\approx 0, > 0$	< 0	≈ 0
Plane	$\approx 0, \approx 0$	≈ 0	≈ 0

Table 4.1: Characterization of surfaces using mean and Gaussian curvature [Dill81].

4.3 The Gaussian Map

"A *Gaussian map* is a function from an orientable surface in Euclidean space to a sphere. It associates to every point on the surface its oriented normal vector" [Weis98]. An *orientable surface* is a surface that has a top and a bottom side; a mobius strip is not orientable. Intuitively, a *Gaussian map* is created by taking all the unit normals at every point on surface S and translating them to a point p that is the centre of a sphere (see Figure 4.1). The convex hull created on the surface of the sphere by the normals is a measure of the change in orientation of the surface and is called the coverage patch. The larger the change in orientation, the larger the coverage patch.

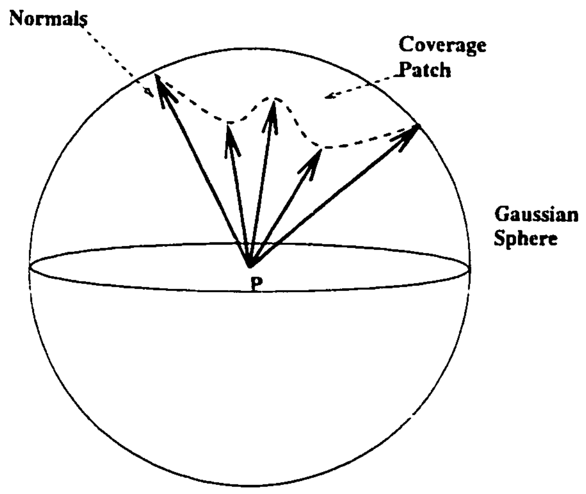


Figure 4.1: The Gaussian map.

4.4 Approximating Curvature

The above curvature measures are defined for infinitely small patches and thus provide a good description of the local surface around a point. However, they do not work well for larger surface patches with multiple scales of curvature; e.g. a carpet from far away looks flat but at close quarters it has a great deal of curvature. Hence there is a need for a measure to approximate curvature at large scales, on large surface patches.

Two measures that approximate curvature at large scales were developed, one to determine the "amount" of curvature, and the other to determine the direction of maximal curvature. Both of these measures will be discussed in chapter 5, but it is important to note that in both cases neither measure is computed in the differential geometry sense. Instead of measuring the curvature in an infinitely small area around a point p on a surface, the orientation change is measured on the entire surface. The term *curvedness* will be used to refer to summaries of orientation change and the term *normal curvedness*, similar to normal curvature, will be used to refer to summaries of orientation change in a given direction from a specific point.

Chapter 5

The R-Simp Algorithm

The R-Simp algorithm tracks vertex connectivity using the clustering technique of [Ross93] and simplifies using the *splitting* algorithm. The *vertex connectivity* of a model describes how the vertices in a model are interconnected; i.e. the edges. Using the *splitting* algorithm to quantize the surface enables R-Simp to quickly simplify large models and to vary detail based on the amount of curvedness. The ability to vary detail with curvedness ensures a minimum level of quality.

The R-Simp algorithm consists of three stages: pre-processing, simplification, and post-processing. This chapter describes the R-Simp algorithm, with focus on the simplification stage. Section 5.1 describes the system, the pre-processing and post-processing stages and gives an overview of the simplification stage.

5.1 Overview

5.1.1 The System

R-Simp can be integrated into an application or used as a stand-alone simplification tool. As a stand-alone simplification tool R-Simp's usage is:

```
rsimp -n # [-o <output file>] input file
```

The parameter `-n #` is required and specifies the required number of vertices in the simplified model. R-Simp will accept the input file on standard in, so it may be used with the standard unix shell pipes. The parameter `-o <output filename>` is optional and specifies the name of the output file. If it is not given then the output file is written to standard out.

Currently a matrix condition threshold, described in Section 5.4.3, is hardcoded. Ideally it should be made into a parameter controlled by the user because the ability to modify this threshold may improve the simplification quality for certain models. Currently it is hardcoded to reduce the complexity of the user interface.

The input file must be in the *PLY* format. The *PLY* file format was developed at Stanford University by Greg Turk[1].

5.1.2 Pre-Processing

The pre-processing stage is responsible for setting up the required data structures. The preprocessing stage does five important tasks.

1. Computes the unit normal for each face in the model.
2. Computes the area of each face in the model.
3. Computes the total surface area of the model.
4. Creates a vertex adjacency list for each vertex v in the model. A vertex adjacency list consists of all vertices that are connected by an edge to v.
5. Creates a face adjacency list for each vertex v in the model. A face adjacency list consists of all faces that contain the vertex v.

5.1.3 Simplification

The simplification stage is responsible for simplifying the model to the desired size.

The algorithm begins with the model in a single cell. A cell contains a collection of faces from the original model called a patch. The centroid of a cell is the representative vertex that is used in the post-processing stage to triangulate the model. The cell with the greatest curvedness is selected and subdivided. The cell is subdivided based on its patch's curvedness resulting in new cells containing less curvedness. This process is iterated until the target number of cells (vertices) is reached.

The algorithm consists of four main steps:

1. Select the cell containing the greatest distortion (curvedness).

[1]http://www-graphics.stanford.edu/data/3Dscanrep/

2. Split the selected cell.
    (a) Compute the cell's normal curvedness.
    (b) Split the cell; produce new cells.
3. Compute the centroid of each new cell.
4. Iterate, until the required codebook (model) size is reached.

In practice it is possible to speed up the simplification stage by subdividing the first cell (the model's bounding box) into eight uniform cells without taking curvedness into account. These eight cells become the current codebook and the algorithm proceeds as outlined above. This step does not reduce output quality but improves execution time considerably.

Even if the entire model is topologically connected, a cell may contain two or more disconnected components. Approximating these disjoint components with a single centroid will usually introduce severe distortion. To solve this problem, a simple topology check that places disjoint components in separate cells is used.

5.1.4 Post-Processing

The post-processing stage is responsible for retriangulating the model. The end result of the simplification stage is a set of cells. Each cell contains a centroid and a number of vertices that reference the centroid. The triangulation algorithm iterates through all the faces in the model and for each face does the following:

1. For each vertex in the face determine the corresponding centroid.
2. If two or more of the vertices point to the same centroid then the face has degenerated into a line or a point and is not included in the new set of faces.

The final result is a simplified polygonal model.

5.2 Selecting a Cell to Split

The first step in a simplification iteration is to select a cell to split. R-Simp selects the cell with the greatest distortion. R-Simp's distortion measure should to be sensitive to the amount of curvedness of the patch in a cell, the patch's size, the scale of curvedness (e.g. carpet looks flat from a distance but is quite curved close up), and be inexpensive to compute.

Most curvature measures are only valid in a infinitely small local area around a point p. What is needed is a measure that is applicable to large surface patches. An inexpensive way to approximate the curvedness of a patch is to measure its coplanarity by examining the magnitude of the area weighted mean normal of its faces. The smaller the magnitude, the greater curvedness of the patch. Figure 5.1 shows a two dimensional example of three different patches with different amounts of curvedness.

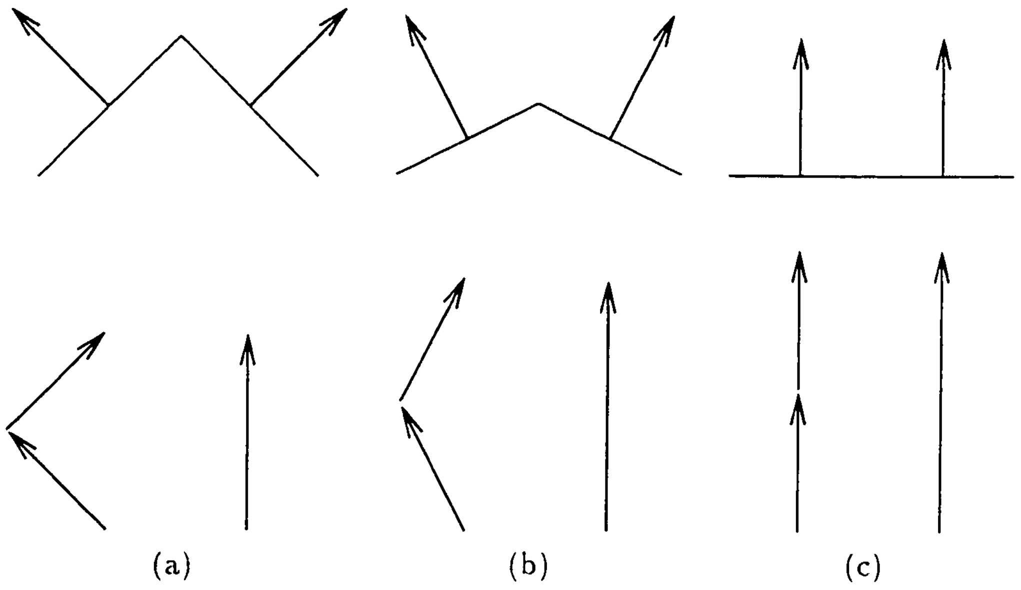


Figure 5.1: Measuring the curvedness of a surface using the magnitude of the area weighted mean normal.

The patch in Figure 5.1a has the most curvedness and has the shortest mean normal, while the patch in Figure 5.1c is completely coplanar and has the longest mean normal. If all the faces are coplanar, the magnitude of the mean normal will equal the area of the patch. The curvedness (coplanarity) component $\mathcal{D}_c$ of the distortion measure is defined as:

$$\mathcal{D}_c = \left\| \frac{\sum_i^N N_i A_i}{\sum_i^N A_i} \right\| \tag{5.1}$$

where N is the number of faces in the patch, N_i is the normal of face i, and A_i is the area of face i.

One could compute the mean normal without area weighting but that would make the measure less sensitive. Figure 5.2 shows this problem in two dimensions.

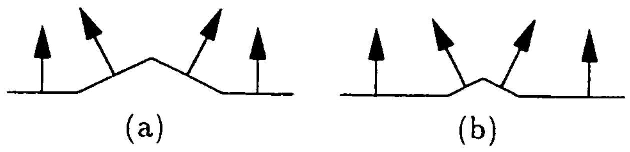


Figure 5.2: Patches (a) and (b) have the same amount of area but the area is distributed differently in the two patches. The sum of unit normals in patch a equals the sum of unit normals in patch (b) without area weighting. Thus, this implies that the curvedness in both patches is the same. This is incorrect because patch (a) has more curvedness.

The patch in Figure 5.2a has considerably more curvedness (less coplanarity) than the patch in Figure 5.2b. The measure in equation 5.1, without area weighting, would return equal values for both patches because the measure would just be summing the unit normals and computing their magnitude, which in this case is the same in both patches. Thus, area weighting is necessary to correctly measure the amount of curvedness in a patch.

Even if a cell is extremely small it can contain a large amount of curvedness. In order to prevent small, highly curved details (e.g. a small spring in an engine) from dominating the simplification, it is necessary to make the distortion measure sensitive to the size of the patch relative to the surface area of the entire model. Figure 5.3 demonstrates this in two dimensions.

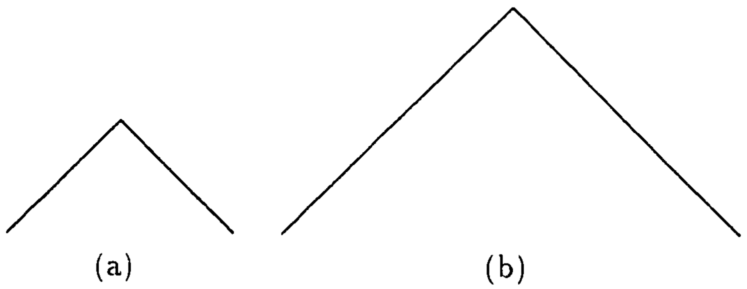


Figure 5.3: Patches (a) and (b) have the same amount of curvedness, but focus should be put on patch (b) since it is bigger than patch (a).

Figure 5.3 shows two patches with identical curvedness, except that the patch in Figure 5.3a is smaller than the patch in Figure 5.3b. In these circumstances, the focus should be placed on the larger patch in Figure 5.3b. If just the curvedness component $\mathcal{D}_c$ is used, then the focus could be placed on either patch. To make the measure sensitive to the patch's size, $\mathcal{D}_c$ is scaled by the ratio of the patch's area to the model's surface area:

$$Distortion = \frac{\sum_i^N A_i}{\sum_i^M A_i}(1 - \mathcal{D}_c) \tag{5.2}$$

where M is the number of faces in the model. $\mathcal{D}_c$ is complemented so that the distortion increases as curvedness increases.

The distortion measure should also be sensitive to the scale of curvedness. Scale of curvedness relates to the visual importance of the curvedness. Figure 5.4 illustrates the differences between small and large scale curvedness. Small scale curvedness is curvedness that may be approximated by a plane without loss of significant detail; e.g. a carpet. Large scale curvedness is curvedness that contributes significant visual detail. It is desirable to be sensitive to the scale of curvedness because small scale curvedness regions could be detected and approximated with a plane, and regions with large scale curvedness could be given more focus. Unfortunately this distortion measure is not sensitive to the scale of curvedness. In Figure 5.4 both patches are equivalent in area and have the same amount of curvedness. However, focus should be put on the patch in Figure 5.4a because the curvedness is visually more significant.

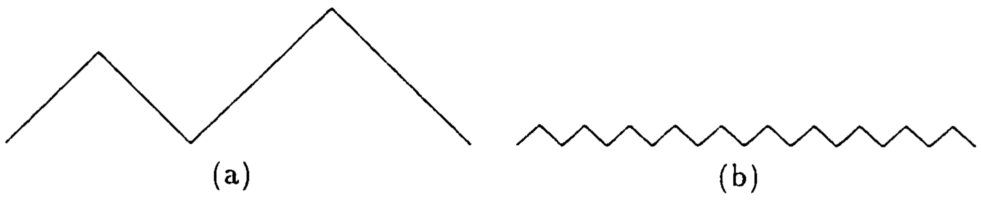


Figure 5.4: Selecting a cell: large and small scale curvedness.

The use of normal curvedness, discussed in section 5.3, was experimented with to add sensitivity to curvedness scale; it resulted in no significant improvement in quality and a 20 percent decrease in speed. In theory, an improvement in quality should have resulted. It is believed that no improvement was seen because the models used to test R-Simp did not contain surfaces like those in Figure 5.4; i.e. the models did not contain surfaces that had small and large scale curvedness of the same magnitude.

5.3 Splitting A Cell

Once a cell is selected, it must be partitioned. The goal is to partition it such that the features in the patch are preserved well. This is accomplished by computing the normal curvedness of the patch and using this curvedness information to split the cell.

5.3.1 Overview & Objectives

The objective when splitting a cell is to create cells that contain patches that are roughly planar, that is, to reduce their curvedness as much as possible (see Figure 5.5).

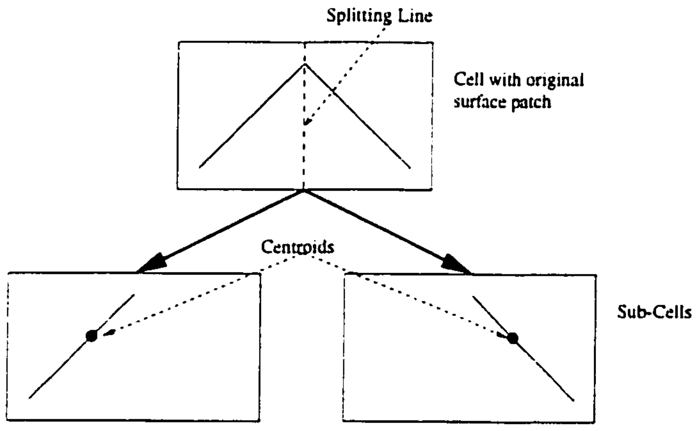


Figure 5.5: Splitting a cell to reduce the amount of curvedness in a patch.

There are several reason why creating planar patches and doing it with as few splitting planes as possible is desirable. Creating planar patches is beneficial because the features on the surface are better preserved. Consider Figure 5.5. Assume that the centroid (the representative vertex) is the mean vertex of the cell. The centroid for each sub-cell would be the mean of the line segments in the sub-cells. After triangulation the result will be that of Figure 5.7a. If the cell is cut the other way, like in Figure 5.6, the sub-cells will contain the top and the bottom of the line segments. The centroids will collapse the area under the original line and the result will be that of Figure 5.7b.

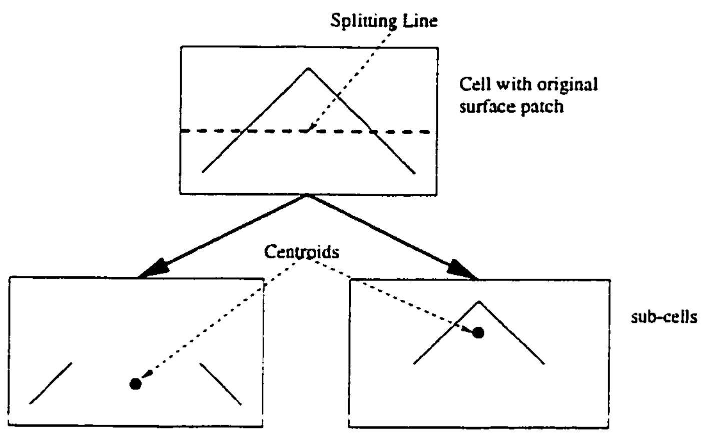


Figure 5.6: Splitting a cell the wrong way.

Clearly the surface patch is better preserved by the approximation in Figure 5.7a. Figure 5.7b has lost almost all its curvedness.

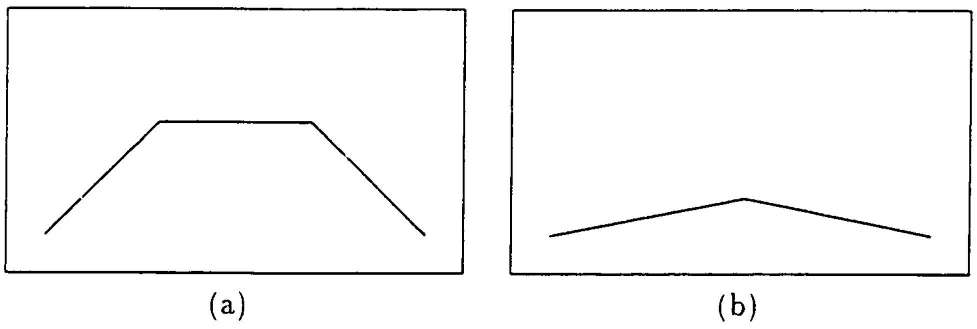


Figure 5.7: Results of good (a) and bad (b) splitting.

The curvedness measure also minimizes the number of splitting planes necessary to split a cell into roughly planar sub-cells. This is quite efficient. Areas of high curvedness are partitioned into more cells, while planar areas are partitioned into fewer cells.

The information necessary to determine the orientation and the number of sub-cells to create is found by sampling normal curvedness in various directions around the mean vertex V_c; V_c is the unweighted mean of vertices in a cell. From the normal curvedness samples, the minimum and maximum directions of normal curvedness and their associated magnitudes are then obtained. These values are used to determine how to split the cell.

The next section (5.3.2) describes the method used for computing the normal curvedness in a patch. The following section (5.3.3) describes how the normal curvedness information is used to split a cell.

5.3.2 Sampling Normal Curvedness

Normal curvedness is an approximation of normal curvature. Normal curvature measures the rate of change of normal vectors in an infinitely small area at a point on a surface in a specific direction. Normal curvedness measures the orientation change at a point P on a surface in a specific direction k. This is accomplished by computing the area-weighted mean angle, plus one standard deviation, between the mean normal of a patch and all the normals of faces that lie in direction k.

The normal curvedness for a patch is computed around the mean vertex, $\mathcal{V}_c$, the unweighted average of all the vertices in a cell. A cell's patch can be quite complex. This complexity is reduced by projecting each face in the cell onto a reference plane, $\mathcal{P}_c$, located at $\mathcal{V}_c$ and perpendicular to the area-weighted mean normal $\mathcal{N}_c$ (see Figure 5.8). The normal curvedness is sampled in j directions represented by vectors. The vectors point away from $\mathcal{V}_c$ and are embedded in $\mathcal{P}_c$. It was found sufficient to sample in five degree increments around $\mathcal{V}_c$, thus $j = 72$.

The goal is to compute the normal curvedness for each of the j normal curvedness directions. For each direction, the L faces are collected whose midpoint falls within 2.5 degrees of the direction vector k. The contribution to normal curvedness in direction k of each face i is computed in two steps. First, twist (sometimes called *geodesic torsion*) is removed by performing a projection of the face normal $\mathcal{N}_i$ onto a plane, $\mathcal{PP}_i$, defined by $\mathcal{N}_c$ and the vector from $\mathcal{V}_c$ to the midpoint of face i (see Figure 5.9). The result is called $\mathcal{N}_{PP_i}$. Second, the contribution for face i is computed by finding the angle, θ_i, between $\mathcal{N}_{PP_i}$ and $\mathcal{N}_c$. If $\mathcal{N}_{PP_i}$ points towards $\mathcal{V}_c$ then θ_i is negated since it represents negative curvedness. This negation step adds sensitivity to curvedness scale, allowing small scale ripples in the patch to cancel.

The normal curvedness computation $\mathcal{K}$ for a direction k is the area weighted mean of θ_i of the L faces associated with the direction plus one standard deviation:

$$\mathcal{K} = \left\| \mathcal{K}_{mean} + \sqrt{\frac{\sum_i^L (\mathcal{K}_{mean} - \theta_i)^2 A_i}{\sum_i^L A_i}} \right\| \tag{5.3}$$

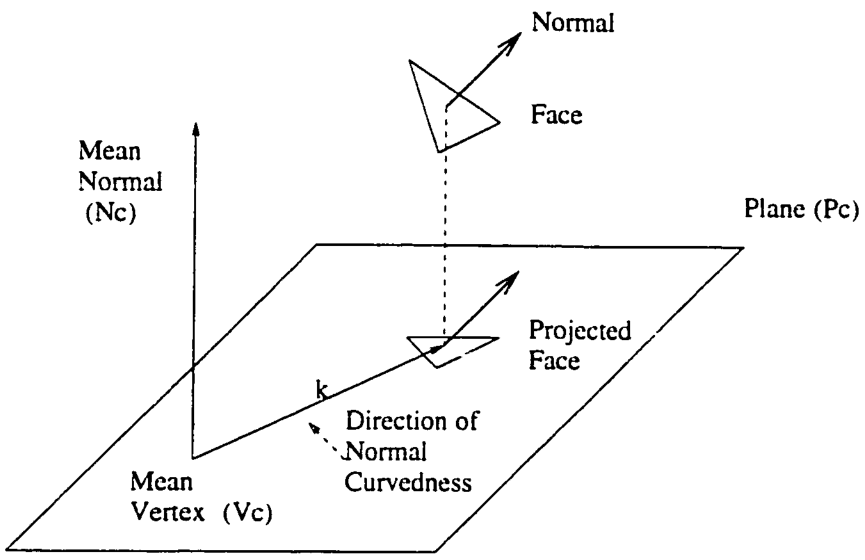


Figure 5.8: Computing normal curvedness in direction k.

where

$$\mathcal{K}_{mean} = \frac{\sum_i^L \theta_i A_i}{\sum_i^L A_i} \tag{5.4}$$

In differential geometry, normal curvature is the rate of orientation change in the chosen direction. However, in practice, it was found that $\mathcal{K}_{mean}$ does not capture the full extent of the normal curvedness; the addition of one standard deviation increases sensitivity to the orientation change.

A representation of the final result is depicted in Figure 5.10. The vectors on the plane represent the direction of sampled normal curvedness and their magnitude represents the amount of curvedness in that direction.

5.3.3 Determining How to Split a Cell

Splitting a cell involves three steps. First, determine how many splitting planes are needed. Second, orient the planes, and finally position them in the patch.

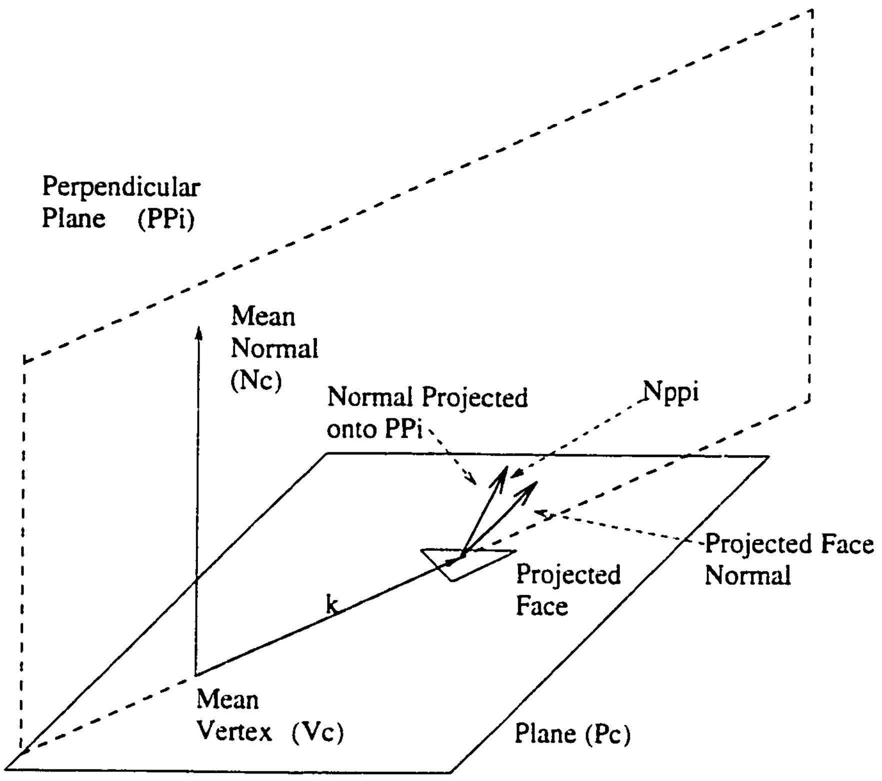


Figure 5.9: Removing twist from face i

Splitting depends on the pattern of curvedness in the patch. To determine the pattern of curvedness the direction of maximum curvedness ($\mathcal{K}_{max}$) and the direction of minimum curvedness ($\mathcal{K}_{min}$) are computed. The computation involves summing the $|\mathcal{K}|$s for opposing normal curvedness sample directions and selecting the minimum and maximum of the sums. The directions of maximum and minimum curvedness are called the principle axes of curvedness, $\mathcal{A}_{max}$ and $\mathcal{A}_{min}$.

In order to determine how many splitting planes are needed, the coverage on the Gaussian sphere is estimated. If $|\mathcal{K}_{max}\mathcal{K}_{min}| > \frac{\pi^2}{4}$ then more than half the sphere is covered; this implies that there is high curvedness in the direction of the mean normal $\mathcal{N}_c$ as well as in the $\mathcal{A}_{min}$ and $\mathcal{A}_{max}$ directions. In this case the patch is split by three planes. Otherwise, the ratio $\frac{\mathcal{K}_{min}}{\mathcal{K}_{max}}$ is computed. If this ratio is greater than 0.5 then the curvedness in the $\mathcal{A}_{min}$ and $\mathcal{A}_{max}$ directions are roughly equal and the patch is split with two planes. If the ratio is less than 0.5 then the curvedness in the $\mathcal{A}_{max}$ direction is dominant and the patch is split with one plane. Recall that the distortion measure includes not just the amount of curvedness, but also the size of the patch.

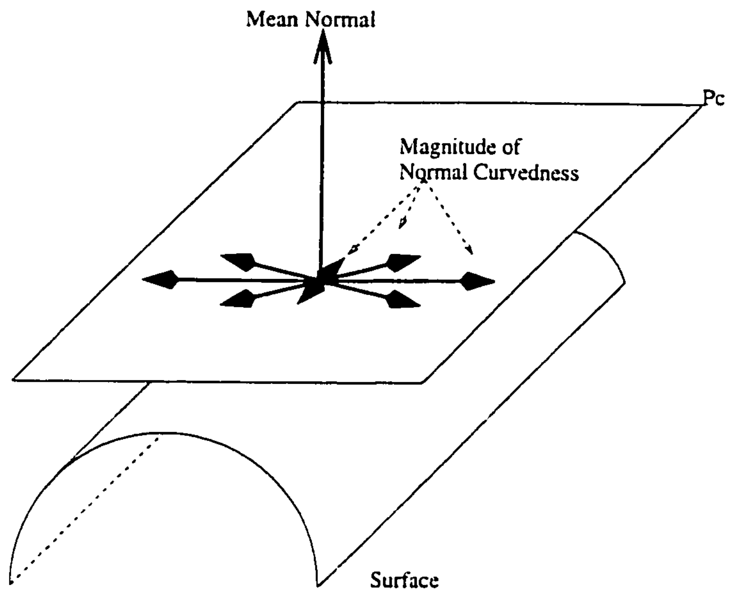


Figure 5.10: The end result of sampling normal curvedness on a cylindrical ridge. As expected there is a great deal of curvedness from side to side but almost none along the ridge of the cylindrical surface.

In some cases, even if a patch is relatively planar, it may still be desirable to split it because it may have some important smaller scale curvedness that is not evident at the patch's current size. In such cases, patches with less then one degree of normal curvedness are split with two planes.

In all cases one splitting plane is orthogonal to A_{max}. It is desirable to split orthogonal to A_{max} because it reduces the curvedness in the surface by the greatest amount. In Figure 5.11a the splitting planes divide the surface parallel to A_{max}. The surface area of the patch becomes smaller but the amount of curvedness in each sub-patch remains the same. In Figure 5.11b the splitting planes split the patch orthogonal to A_{max} and the patch size and the amount of curvedness in each patch is reduced. Figure 5.12 shows a cylindrical patch that is split by a single plane.

In the cases with two and three splitting planes, one plane is parallel to $\mathcal{N}_c$ and perpendicular to the plane orthogonal to A_{max}. Since there is curvedness in the A_{max} direction and in the A_{min} direction, the patch is probably an elliptical bump, an elliptical hollow, or a saddle surface. To maximize the creation of planar regions, these patches are split into four sub-cells. Figure 5.13 shows an elliptical hollow being partitioned with two splitting planes.

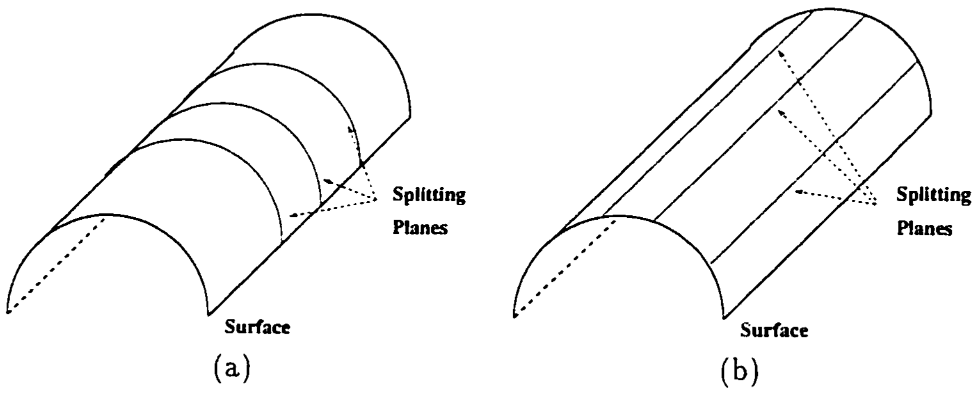


Figure 5.11: Splitting parallel and orthogonal to $\mathcal{A}_{max}$.

In the case with three splitting planes, one plane is perpendicular to $\mathcal{N}_c$. Since there is high curvedness in all directions it is not clear how the patch is behaving except that more than half of the Gaussian sphere is covered. By splitting with three planes it is more probable that the sub-cells will contain patches on which the curvedness can be computed more accurately, and thus determining the behaviour of the surface; Figure 5.14 shows a sphere that is split by three planes.

Finally, it is necessary to position the splitting planes in the patch. Ideally the surface should be split along any ridges (hollows) or through any elliptical bumps (hollows). Locating such features would require finer sampling of the patch. One technique that works well is to compute the mean midpoint of the faces associated with $\mathcal{A}_{max}$ and ensuring that all splitting planes contain this point. In degenerate cases it is possible that this positioning technique will not split the cell (meaning that all vertices would be placed into single sub-cell). If this occurs, the mean vertex $\mathcal{V}_c$ is used to position the splitting planes.

5.4 Computing the Centroid

One of the drawbacks of clustering algorithms is that the centroid computation is often poor. In [Ross93] the representative vertex for a cluster is the mean or the weighted mean of the vertices in a cluster. This approach can poorly approximate curvature within a cluster; e.g. smoothing sharp edges and corners.

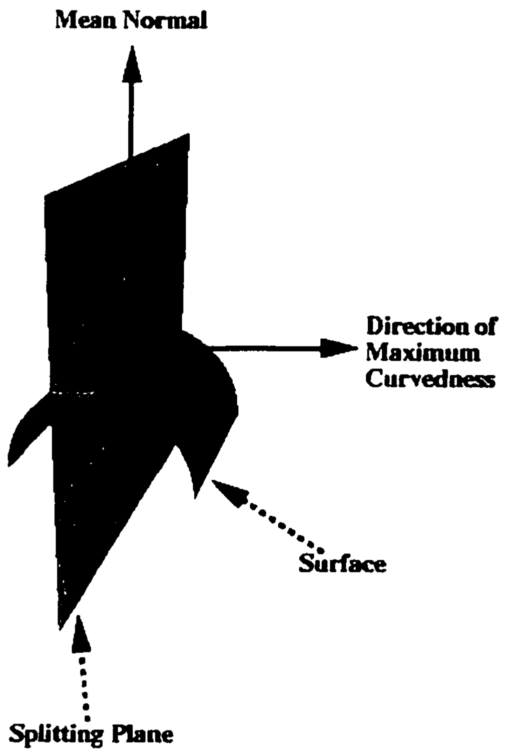


Figure 5.12: Splitting a cell with one plane.

5.4.1 Approaches Tested

Initially, for simplicity, the mean vertex was used as in [Ross93]. The overall simplification was good but creases and small features disappeared quickly. To solve this problem two different approaches were tried. The first proved too expensive, so the second was used.

The first approach was to try to align the mean normal of the simplified patch with $\mathcal{N}_c$. However, since changing the mean normal in one patch affects the mean normal in a neighbouring patch, such a technique would require the solution to a set of non-linear simultaneous equations. This approach was rejected as incompatible with the objective of high speed simplification.

The second approach, which was implemented, is the quadratic vertex placement policy described in [Garl97, Garl98]. This approach proved to be fast and improved quality.

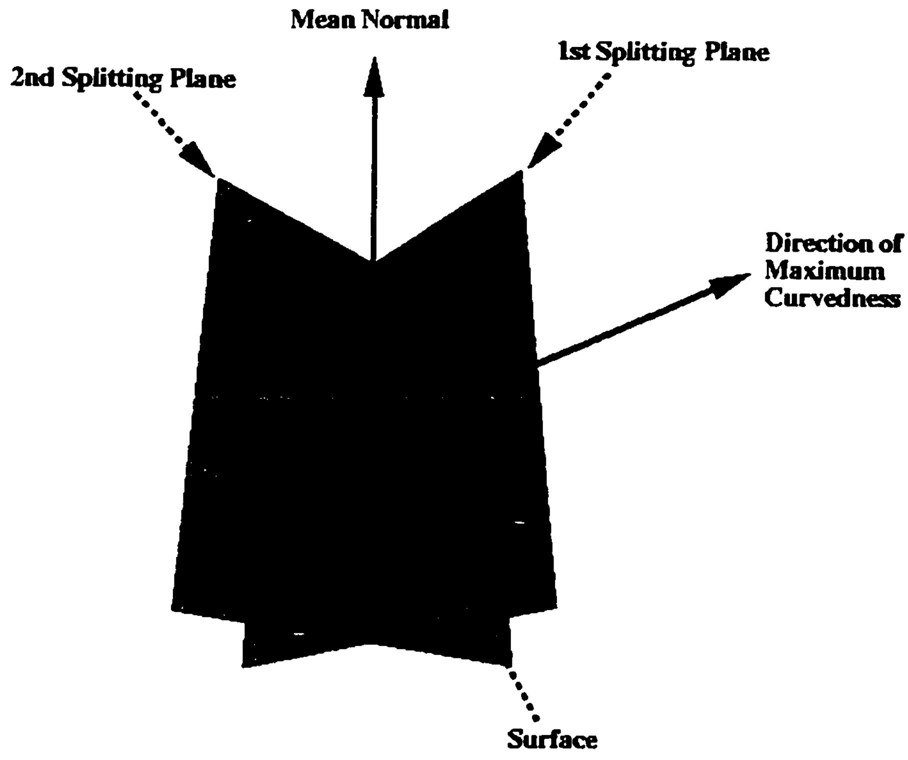


Figure 5.13: Splitting a cell with two planes.

5.4.2 Quadratic Vertex Placement Policy

The quadratic vertex placement policy is a minimization process that optimizes the location of the centroid with respect to the faces in a cell. The idea is to place the centroid as close as possible to all the faces in a cell. Intuitively, placing the centroid close to all the faces in a cell will result in edges and corners being better preserved.

This process minimizes the sum of squared distance between a vertex v and all the faces (planes) in a cell. Each face in a cell defines a plane that satisfies the equation[2] $n^T v + d = 0$, where $n = [n_x, n_y, n_z]^T$ is the unit face normal and d is a constant. The squared distance between a vertex and a planes is:

$$\begin{aligned} D^2 &= (n^T v + d)^2 && (5.5)\\ &= (n^T v + d)(n^T v + d) && (5.6)\\ &= v^T(nn^T)v + 2dn^T v + d^2 && (5.7) \end{aligned}$$

[2]By convention, all vectors are column vectors. The inner product of two vectors is $n \cdot v = n^T v$.

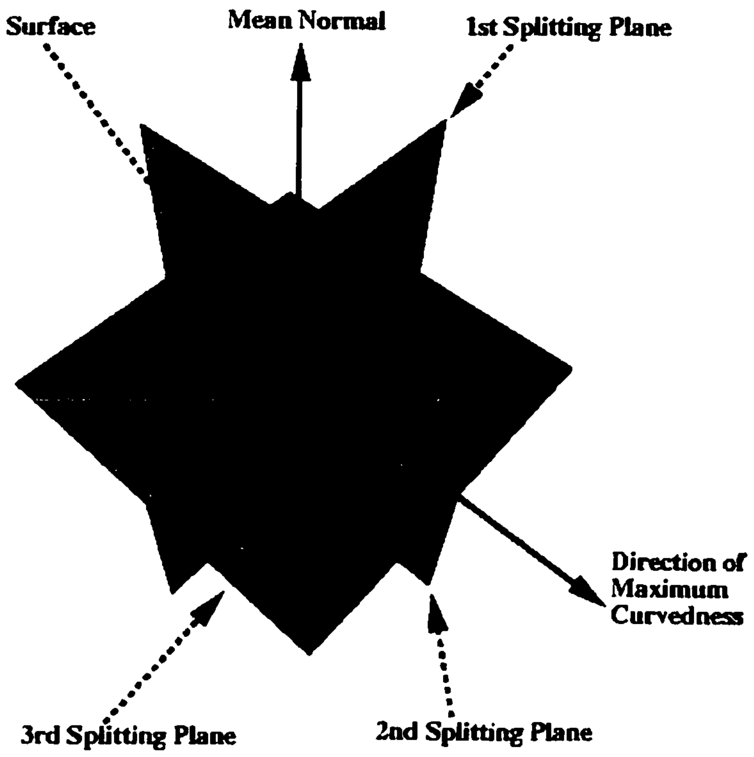


Figure 5.14: Splitting a cell with three planes.

Equation 5.7 is a quadratic plus a linear term plus a constant that can be represented by Q:

$$Q(v) = v^T \mathcal{A} v + 2b^T v + c \tag{5.8}$$

where

$$\mathcal{A} = nn^T, b = dn, c = d^2$$

Representing the quadratic in the form of Q is convenient because component wise addition can be defined as:

$$Q_1(v) + Q_2(v) = (Q_1 + Q_2)(v) \tag{5.9}$$

where

$$(Q_1 + Q_2)(v) = v^T(\mathcal{A}_1 + \mathcal{A}_2)v + 2(b_1 + b_2)^T + c_1 + c_2 \tag{5.10}$$

Thus, all that is needed is one quadratic Q_t to compute the sum of squared distances between a vertex v and all the faces in the cell. Where Q_t:

$$Q_t = \sum_{i \in Faces}^{N} Q_i \tag{5.11}$$

The next step is to determine v such that $Q_t(v)$ is minimum. Since Q_t is a quadratic, the minimum occurs where the partial derivatives are equal to zero. Thus, a solution is found to:

$$\begin{bmatrix} \frac{\partial Q}{\partial x} \\ \frac{\partial Q}{\partial y} \\ \frac{\partial Q}{\partial z} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.12}$$

which is equivalent to:

$$v = -\mathcal{A}^{-1} b \tag{5.13}$$

where

$$\mathcal{A} = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{bmatrix} \tag{5.14}$$

Equation 5.13 is just a simple system of linear equations for which a solution can be found in constant time. The solution represents the optimal vertex position by the squared distance criteria.

5.4.3 Problems with the quadratic vertex placement policy

In some instances the system (Equation 5.13) may be unconstrained or ill-conditioned. In these circumstances the optimal vertex position is undetermined and the mean vertex is used.

The system becomes unconstrained when the number of non-coplanar planes is less than three. The matrix $\mathcal{A}$ becomes ill-conditioned when the system is unconstrained, but due to floating point error the system becomes constrained. This can happen in two cases. In the first case, the surface is planar except there is a point that is elevated a small distance from the plane, in the second case, the surface is a cylindrical ridge (hollow) and a point on the ridge lies a small distance above or below the ridge. These distances are extremely small, almost equal to the system's floating point resolution,

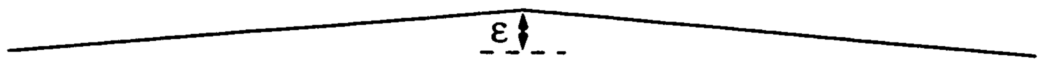


Figure 5.15: The cause of an ill-conditioned matrix $\mathcal{A}$; $\epsilon \approx 0$.

and when they are incorporated into the matrix $\mathcal{A}$, they cause the matrix to become ill-conditioned. Figure 5.15 is a two dimensional example.

Such surfaces are difficult to detect, so instead the condition of matrix $\mathcal{A}$ is determined using a standard approach from numerical analysis [Buch92]. The matrix condition value is computed by:

$$Cond = \|\mathcal{A}\|_\infty \left\|\mathcal{A}^{-1}\right\|_\infty \tag{5.15}$$

where $\|\mathcal{A}\|_\infty$ is the infinity norm of $\mathcal{A}$.

The closer the *Cond* value is to unity the better conditioned the matrix is. Matrix $\mathcal{A}$ is usable if its condition value is less than a certain threshold. Ideally this threshold should be adjusted by the user since the threshold can vary due to the geometry and the topology of the model. To reduce the complexity of the user interface to R-Simp the threshold was fixed at 1000 which worked well for the majority of the models tested. It was found though that for models of man made objects that contained many planar regions and sharp corners, such as a table, the threshold value should be set lower.

5.5 Topology Check

Even if the entire model is topologically connected, a given cell may contain two or more disconnected components. Approximating these components with a single centroid can introduce severe distortion. The algorithm uses a topology check to determine if a cell contains topologically disjoint components.

The topology check is done as a breadth-first traversal on the model's original vertices in a cell. Initially, the algorithm has a list $\mathcal{L}$ of all vertices in a cell. It removes a vertex from $\mathcal{L}$ and inserts it into a queue. The algorithm is:

1. Remove a vertex v_q from the queue.

2. For each vertex in the adjacency list of v_q:

   (a) Insert it into the queue if the vertex is in $\mathcal{L}$.

   (b) Remove it from $\mathcal{L}$.

3. Iterate until the queue is empty.

If $\mathcal{L}$ is empty when the queue is empty, then the cell contains only one component and the simplification algorithm proceeds normally. If $\mathcal{L}$ is not empty, then the cell has disjoint components. The component that was just removed from $\mathcal{L}$ is put into a separate cell and the topology check algorithm is rerun on the rest of $\mathcal{L}$. The process continues until $\mathcal{L}$ is empty. Each component is put into a separate cell. Although this topology check increases the simplification time, the increase in simplification quality is considerable.

Chapter 6

Evaluation & Discussion

Every simplification algorithm makes a tradeoff between quality and speed. R-Simp emphasizes speed over quality. At the same time, it ensures a minimum level of quality that is much higher than many existing vertex clustering algorithms.

One has to be careful when discussing the quality of simplifications because quality is more difficult to measure than speed. An author of an algorithm may claim that the algorithm produces better quality simplifications than other algorithms, but that claim depends on the models simplified and the quality measure used for the comparison. As was discussed in chapter 3, different algorithms are suited for different surfaces. In the same manner, the suitability of a quality measure may depend on the simplification algorithm. In most cases, the quality of a simplification should be qualitatively judged by the end user of the simplified model.

This chapter evaluates and discusses the performance of R-Simp. To remove some bias, several different models were used for comparisons and quantitative and informal qualitative analyses of the simplifications were performed. The chapter first introduces the concepts of simplification speed and simplification quality and, how to measure them. Second, simplification results are presented along with comparisons to two existing simplification techniques. Finally, the chapter discusses the limitations of R-Simp, and the applications that suit its strengths.

6.1 Simplification Speed

Simplification speed is a measure of how quickly a simplification algorithm can produce the output model, given an input model. Normally, speed is not difficult to measure, but certain conditions need to be present for the results to be valid. The first condition that should be satisfied is that all the speed trials be run on a dedicated machine with a large amount of memory. The speed measure should only include the processing time required to simplify a model. If the machine being used for the speed trials is used for other purposes at the same time, then the time measures will include time when the simplification algorithm was suspended. If the machine has a limited amount of memory, then the timing result will also include system paging time. The next condition that should be met relates to the timing of the actual simplification algorithms. The implementations that are run usually contain code to read into memory the input file and write the output file. It is desirable not to measure the time taken to do these tasks, because these tasks are irrelevant to the performance of the simplification algorithm (and all algorithms must perform this step). Only the routine that performs the simplification should be timed. If both conditions are met, then the speed results obtained should be relatively meaningful.

6.2 Quality of a Simplification

The quality of a simplified model is a measure of similarity between the simplified and the original model; i.e. how much does the simplified model resemble the original. This comparison can be done quantitatively and qualitatively.

6.2.1 Quantitative Measures of Quality

Quantitatively, the quality of a simplified model can be assessed in several different ways. The most common way is to compare the simplified model's geometry and topology to that of the original. Usually, this comparison is accomplished by measuring the distance between the original and the simplified surface; the smaller the distance the better the simplification. If the original model is an enclosed surface then comparing the volumes of the simplified model and that of the original may provide an indication on the accuracy of the simplification. This measure alone can be misleading because two objects may have the same volume, but have completely different shapes. If the model has boundaries, than an analysis of how well the boundaries are preserved may also provide an indication as to the quality of the simplification.

One such quality measuring tool is *Metro* [Cign97]. *Metro* computes the maximum and mean geometric errors of the simplified surface with respect to the original. The error is computed. first, by point sampling both the simplified and the original surface, and second, by finding the two-sided Hausdorff distance between the sample point and the other surface. If the surface is a closed manifold, then *Metro* computes and compares the volume of the simplified surface with the original.

All these measures are good at quantifying the quality of the geometrical and topological simplification. However, if the surface is textured or coloured, these measures can not quantify how well the texture or colour was preserved.

One possible way of measuring the preservation quality of colours and textures on a surface is to do image based comparison. Image based comparison involves making an image of the simplified and the original model from the same viewing position and then comparing the differences at the pixel level. One way of doing this is by computing the *root mean square* error:

$$\mathcal{E}_{RMS} = \sqrt{\frac{\sum_{i=1}^{n}(p_o - p_s)^2}{n}} \tag{6.1}$$

where n is the number of pixels in the image. p_o is a pixel in the image of the original model. and p_s is a pixel in the image of the simplified model. This quality measure is not able to measure the quality of the whole model because it compares images of the model and not the models themselves. This measure does not only depend on the model itself but also on the model's surroundings such as lighting and rendering techniques. Hence. an evaluation of only the model is not possible. and surrounding factors can contribute to the quality computation of the model. Moreover, there is little evidence that *root mean square* error has a strong correlation to image quality. Substitutes for this measure are an active area of research.

6.2.2 Qualitative Measures of Quality

Currently machines are not able to do qualitative assessments and therefore this task is left to humans. Given that humans are all different there does not exist a single set of criteria that may be used to asses the quality of a simplification. But with a review of visual perception literature it is possible to determine several criteria that may be used to judge the quality of a simplification. Humans use *contrast* and the *spatial arrangement* of the contrast to perceive the form of an object [Seku94]. Informally, contrast is change of colour across the visual field. In polygonal models, the spatial arrangement of contrast is produced by the model's edges. Since edges play an important role in object recognition, the criteria used to assess the quality of a simplification should be based on how well edges are preserved in the simplified model.

The first criterion is how well the silhouette is preserved. The silhouette is one of the most prominent edges in a model and therefore it is important that it be preserved since silhouettes depend on viewpoint, models should be viewed from many viewpoints. The second criterion is how well significant model features, such as ears on a bunny or the legs on a cow, are preserved. Such features are important because they tend to contribute large edges to the interior of the model and to the silhouette. The third and least important criterion is how well smaller edges and creases are preserved. By employing these criteria one could informally evaluate the quality of a simplification algorithm. Biederman [Seku94] proposed that object recognition is based on *geons* (for **geo**metrical icon**s**). A *geon* is a basic shape such as a cylinder or a tetrahedron. He proposed that the human visual system initially breaks down an object into *geons* to quickly do object recognition. Large features, such as a bunny's ears or a cow's leg, would be considered *geons*. Thus, how well large features in a model are preserved is important.

Since every person is different and each has their opinion of what is good and bad, a good way to evaluate the quality of a simplification algorithm is to perform a user study. One possible user study would involve the simplifying of several different models with various simplification algorithms and asking individuals to rank the models based on their perception of quality. Another would ask them to name the displayed objects, and the naming time would be recorded. A final and a more psychophysical approach would ask individuals to adjust low pass filters until the filtered view of an original and a simplified model were not distinguishable.

6.3 Evaluation

Simplification algorithms are usually judged by two criteria. The first criterion is speed, the time required to simplify a model. The second and more difficult to measure criteria is quality.

To assess the performance of R-Simp its runtime and simplification quality was compared to two other algorithms. The first is a simple vertex clustering algorithm, similar to the one developed by [Ross93]. The second is QSlim [Garl97], an algorithm that uses the *pairwise nearest neighbour* technique to simplify the surface. These algorithms were used to simplify seven different models to several levels of detail. Seven different models were used to remove bias due to the suitability of the algorithm for a particular surface. The simplification times and the geometrical error between the simplified surface and the original were recorded.

6.3.1 Measures Employed

To measure the geometrical error between a simplified surface and the original a tool called *Metro* [Cign97] was used.

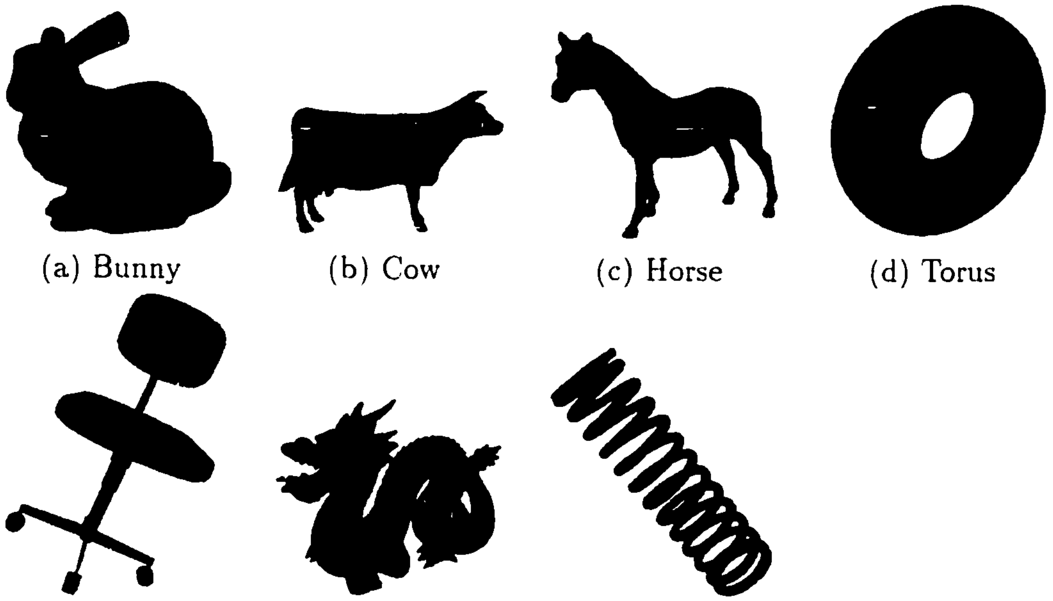
The *Metro* tool was used because it was written by a third party, and thus possibly eliminating one source of bias. *Metro* is not without its faults. Running *Metro* on two identical models produced results that showed that the models were slightly different. This is probably due to floating point error. Also, *Metro's* results are only meaningful when the models being compared are fairly similar. Yet in general, informal qualitative assessment of quality correlated well with the results produced by *Metro*.

6.3.2 The Test Bed

The models used for the comparisons are listed in Table 6.1 and are shown in Figure 6.1. All the simplifications were performed on a 195Mhz R10000 Onyx2 with 512MB of memory. Simplification run-times reflect the time to simplify the model; they do not include the time to read in and write out the model.

Model	Number of Polygons	Number of Vertices
Bunny	69451	34834
Cow	5782	2903
Horse	96966	48485
Torus	20000	10000
Chair	2481	1318
Dragon	871306	435545
Spring	9386	4695

Table 6.1: Statistics of models used in the comparisons.



(a) Bunny (b) Cow (c) Horse (d) Torus

(e) Chair (f) Dragon (g) Spring

Figure 6.1: Models used in comparisons.

6.3.3 Comparisons

The R-Simp algorithm was assessed on its speed and the quality of its simplifications. The first task was to assess R-Simp's speed. Two speed assessments were performed. A primary speed assessment compared the run-times of all three algorithms on the seven models. The results are summarized in Table 6.2. A more in-depth assessment was done using only QSlim for comparison and two of the seven models.

The second speed assessment determined the effect of output model size on simplification time. For this assessment only QSlim was used, because it is comparable in speed and quality to R-Simp. The bunny and the dragon were used for this comparison because they provided the required range in input model size and surface types. Figures 6.2 and 6.3 show the speed of the R-Simp algorithm with respect to QSlim on the bunny and the dragon models.

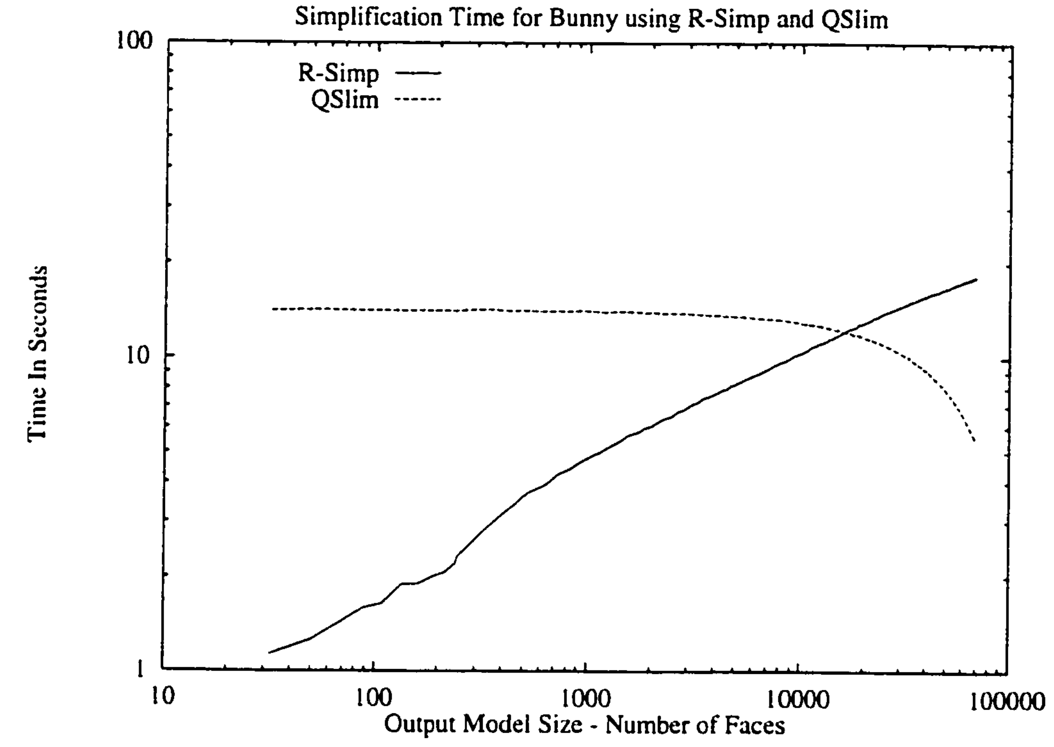


Figure 6.2: The effect of output model size on simplification time for the bunny.

Figures 6.2 and 6.3 also illustrate the advantages of simplifying from coarse to fine. Even before QSlim removes a single polygon, R-Simp is able to produce a simplified model of a bunny containing between zero and 1200 polygons, and a simplified model of the dragon containing between zero and 10000 polygons. However, as the output model size increases R-Simp, slows down in a sub-linear fashion. When output model size is relatively large, R-Simp becomes slower than QSlim. Clearly, R-Simp should be used for drastic simplification, when the required output model size is a small fraction of input model size.

Table 6.2 the shows speed results with other models, and using the vertex clustering algorithm. It is evident that the vertex clustering algorithm is considerably faster than R-Simp or QSlim, and that R-Simp is on average two to four time faster than QSlim for an output model size of about 1500 to 2000 polygons.

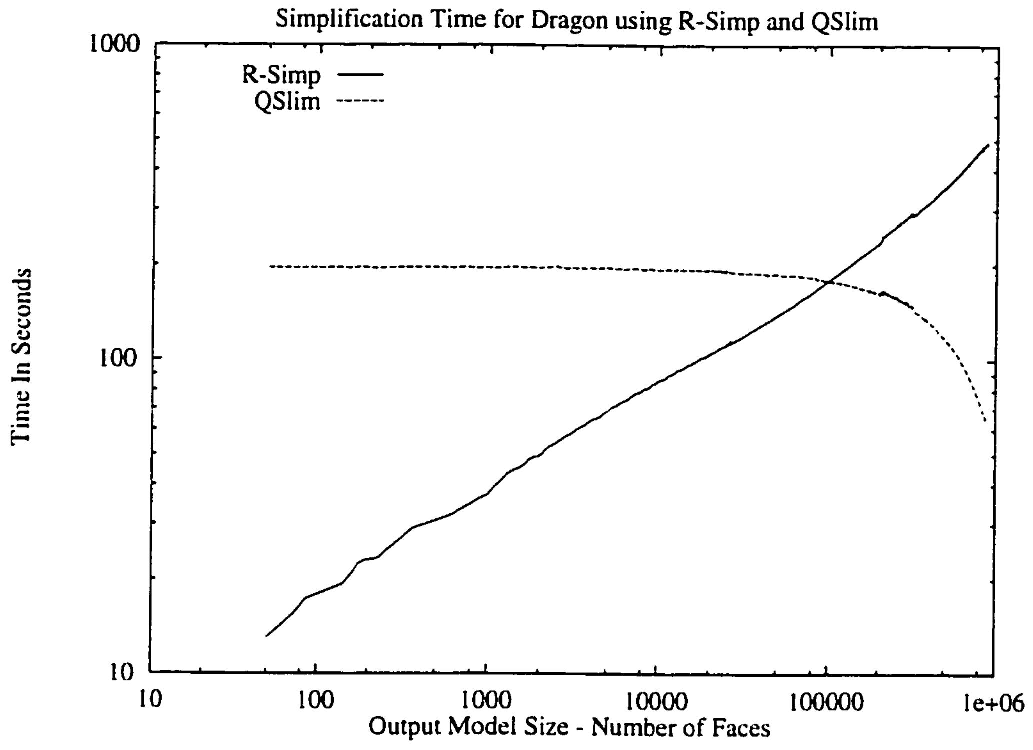


Figure 6.3: The effect of output model size on simplification time for the dragon.

The quality of the simplifications was assessed in two parts. First, *Metro* was used to compare geometrical error. These results are presented in Table 6.2. As expected, the simplifications produced by the vertex clustering algorithm tend to have the largest geometrical error, because the algorithm emphasizes speed. Simplifications produced by QSlim tend to have the least amount of geometrical error; R-Simp has slightly more error than QSlim. There are two anomalies in Table 6.2. First, *Metro* judged R-Simp's simplified chair better than QSlim's. The possible reason for this is that R-Simp's matrix condition check is more accurate than that of QSlim's (see Section 5.4.3). Second, *Metro* could not evaluate the simplified spring produced by QSlim and the vertex clustering algorithm. When *Metro* was run on these models it crashed. It is unclear why this happened.

Next, an informal qualitative assessment was performed, from multiple views, of the simplifications. Figures 6.7, 6.8, and 6.9 show the simplifications done with all three algorithms along with the originals; the three models shown are the bunny, the dragon, and the cow.

Model	Vertex Cluster			R-Simp			QSlim		
	Polys	Time (s)	Error	Polys	Time (s)	Error	Polys	Time (s)	Error
Bunny	1602	0.09	0.7659	1601	5.86	0.3615	1600	14.09	0.2042
Cow	1598	0.03	23.787	1600	0.72	15.382	1600	0.76	7.6146
Horse	1598	0.29	0.0072	1602	6.79	0.0020	1600	19.41	0.0014
Torus 1	1616	0.05	0.0063	1602	2.10	0.0029	1600	3.26	0.0017
Torus 2	400	0.05	0.0228	400	1.33	0.0094	400	3.35	0.0073
Chair	796	0.00	0.2974	815	0.21	0.0830	800	0.32	0.1616
Dragon	2122	2.65	0.0011	2068	51.10	0.0007	2068	195.60	0.0003
Spring	1634	0.04	N/A	1600	0.78	0.0171	1600	1.48	N/A
Mean		0.40	3.5554		8.61	1.9823		29.78	1.1416

Table 6.2: Comparison of R-Simp with QSlim and vertex clustering on a variety of models. The simplified size, execution time, and geometric error are shown. The error is the mean total error returned by *Metro* [Cign97]. On the spring model *Metro* was not able to compute the error on the QSlim and vertex clustering simplifications.

In all three models the vertex clustering simplification looks the worst. The regular subdivision is noticeable, especially on the bunny, and hence small details are not preserved. Finer details on the dragon have disappeared, like the horns and the claws on the feet. The hind legs on the cow are in two parts. This is due to the regular subdivision used by vertex clustering algorithms like [Ross93]. Figure 6.4 shows how features like the legs can disappear when regular subdivision is used.

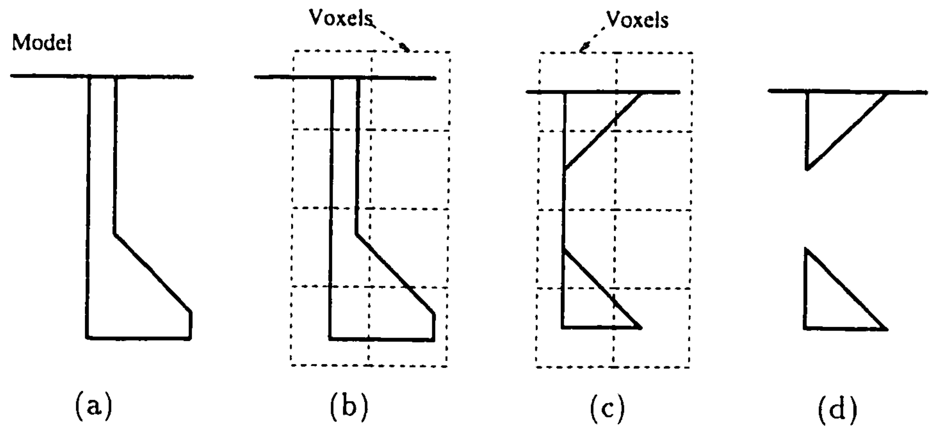


Figure 6.4: The disappearing cow leg. The leg is regularly subdivided in (b) and simplified as shown in (c). Since the middle portion is just a single line it is deleted and a break is created as shown in (d).

QSlim's results look the best. The creases are well defined and much of the detail has been preserved from the original models. It is possible to make out the eyes on all three models and the claws and horns on the dragon have been preserved.

R-Simp's results are comparable to QSlim's. More of the smaller details have disappeared and the creases are not as well defined. QSlim is better able to preserve the creases and small details, because it merges vertices and hence can simplify along the feature edges. R-Simp does not have problems with large features such as the bunny's ears. Figure 6.7c shows the simplified version of the bunny done by R-Simp. The ears are long, smooth, and almost identical to the ears in the original model. The dragon in Figure 6.8 has many small details, and therefore R-Simp did not do as good of a job simplifying it as it did the bunny.

6.4 Other Criteria

Table 6.3 shows how R-Simp compares to the other two algorithms in terms of the comparison criteria presented in Section 3.1.2. All three algorithms are able to simplify topology. QSlim has the overall advantage because it has the most control on what topology to simplify. R-Simp does not have as much control but is still able to prevent disjoint components from being represented by one vertex. Only QSlim can guarantee the preservation of topology. Both the vertex clustering algorithm and R-Simp are able to accept non-manifold input. but QSlim is able to preserve the manifoldness of a surface.

Algorithm	T.S.	T.O.	M.I.	M.O.	Attr.	Brdr.
Vertex Clustering	√		√			
R-Simp	√		√			
QSlim	√	√		$\propto$	√V2.0	√

Table 6.3: R-Simp in relation to QSlim and the vertex clustering algorithm in terms of the other comparison criteria. See Table 3.1 for legend.

QSlim version 2.0 can preserve additional surface attributes, such as colour and texture, and both versions of QSlim explicitly preserve borders.

6.5 The Pros of Simplifying in Reverse

The R-Simp algorithm simplifies in reverse; from coarse to fine. Simplifying in reverse has several advantages.

One advantage is that as the input model size increases, simplification time increases linearly. R-Simp has complexity $n_i \log n_o$, where n_i is the model input size and n_o is the model output size; most other algorithms are $n_i \log n_i$. Figure 6.5 shows how the size of the input model affects the simplification time. To determine the affect of input model size on simplification time, a large model was obtained and simplified down to various levels of detail using a good quality simplification algorithm. The dragon was used as the test model and QSlim was used to simplify it down to various levels of detail. The levels of detail of the dragon were then simplified by R-Simp and QSlim to 2100 polygons. As the graph shows, the bigger the input model, the longer it takes to simplify. However, QSlim's curve is significantly steeper than R-Simp's.

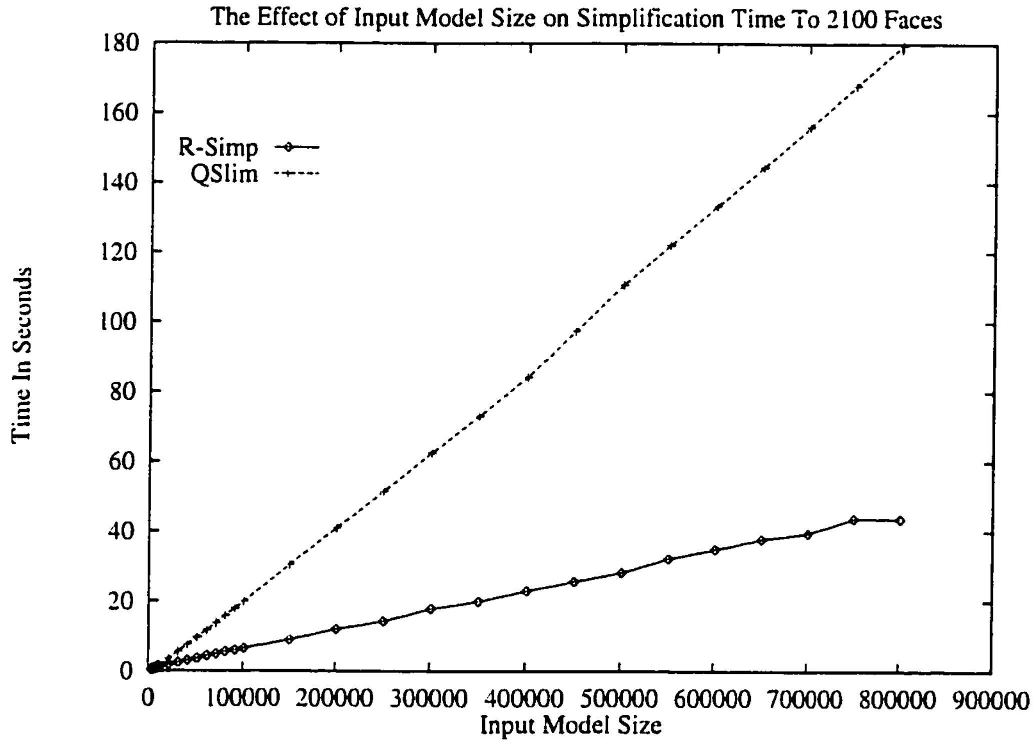


Figure 6.5: The effects of input model size on simplification time. Output model size is 2100 polygons.

Simplifying in reverse enables R-Simp to arrive more quickly at the final result. The reason for this is that R-Simp has to do less work. If the original model is 100000 vertices, then an algorithm like QSlim would have to do approximately 95000 iterations to reduce the model to 5000 vertices. R-Simp would only have to do 5000 iterations.

Simplifying in reverse allows for R-Simp's simplification process to be limited by several different bounds. Like many other algorithms, R-Simp can be given a vertex budget; i.e. it can create a simplified model with a given number of vertices. R-Simp can also be bounded by a time constraint: given a time limit in which it has to produce a simplified model, R-Simp can return a simplified model of unspecified size. The size of the output model will depend on the size of the input model; the larger the input model the smaller the output model will be. This is a big plus, because algorithms that simplify from fine to coarse can not be bound by this type of a constraint because after the time limit has expired there is no guarantee that any size of model will be produced. The vertex bound and the time limit constraints can also be combined. For example, R-Simp can be directed to produce a model that is a given number of polygons or less in a given number of seconds.

6.6 The Cons

R-Simp has two main limitations: simplifications to low polygon counts and optimally placing the centroid in certain cases. R-Simp emphasizes speed over quality and thus the quality of the simplifications suffers at low polygon counts. At low polygon counts, (and after only a small number of splits) the curvedness measure is not accurate enough to determine the optimal way to split a cell and to position the representative vertex. Figure 6.10 shows the bunny at ten different levels of detail, ranging from 50 to 1600 polygons. At approximately 600 polygons most of the creases have disappeared and at 200 polygons the bunny no longer looks like a bunny. Figure 6.11 shows models with the same number of faces created with QSlim. The bunny at 200 polygons still looks like a bunny. This can also be seen with models that contain many small, highly curved features, like the horns on the dragon. For example, the dragon in Figure 6.8c is composed of 2068 polygons, yet it does not look as good as the model simplified by QSlim (Figure 6.8d) whereas the bunny and the cow models are more comparable.

R-Simp has also problems when placing the representative vertex in certain patches. These include fairly planar patches and patches that have regular, sharp edges, both of which are usually found in models of man-made objects. The problem arises when the condition of the system of linear equations used to determine the optimal position of the representative vertex (Section 5.4.2) is ill-conditioned, but the threshold says that it is not. This results in a vertex being placed somewhere off at infinity, and the simplification containing a long spike. There are several solutions to this problem. One possible approach is to vary the condition threshold, but that is not desirable because the end user should tweak as few knobs as possible, and if the algorithm is integrated into an application there might not be a possibility for the necessary user input.

6.7 Enhancements

There are several small enhancements that could improve the algorithm's quality and speed.

The first enhancement would improve R-Simp's speed. Currently the optimal position for the vertex is being computed for every cell. This procedure should to be moved to the end of the algorithm and the optimal vertex should be computed only for the vertices that are in the simplified model.

The second enhancement would improve R-Simp's quality and solve the problem with the ill-conditioned matrix. Currently all cells are put through the optimal centroid placement procedure. What should be done is to classify each patch in the cell depending on its curvedness. There would be three classifications; a planar patch, a cylindrical patch, and everything else. By only applying the optimal representative vertex placement routine to the everything else category the problem should be solved. Planar patches do not need the optimal placement routine because a set of polygons can meet at any position on a planar patch to approximate it. Patches that are cylindrically shaped can use the optimal placement routine but it is more likely that the computation matrix will be ill-conditioned. A possible solution would be to set the condition threshold low and check the condition of the computation matrix. If the result is an ill-conditioned matrix then the representative vertex could be positioned at the location of the splitting plane, since the position represents the ridge of the cylindrical surface.

An optimization that should be made is to make R-Simp more memory efficient. Inefficient memory usage limits the size of the models R-Simp can effectively simplify, and slows the algorithm down. Currently R-Simp needs 4-5 times the input model size in memory. This memory requirement can be reduced significantly.

Portions of the code can also be optimized to improve the overall efficiency and speed.

6.8 Applications of R-Simp

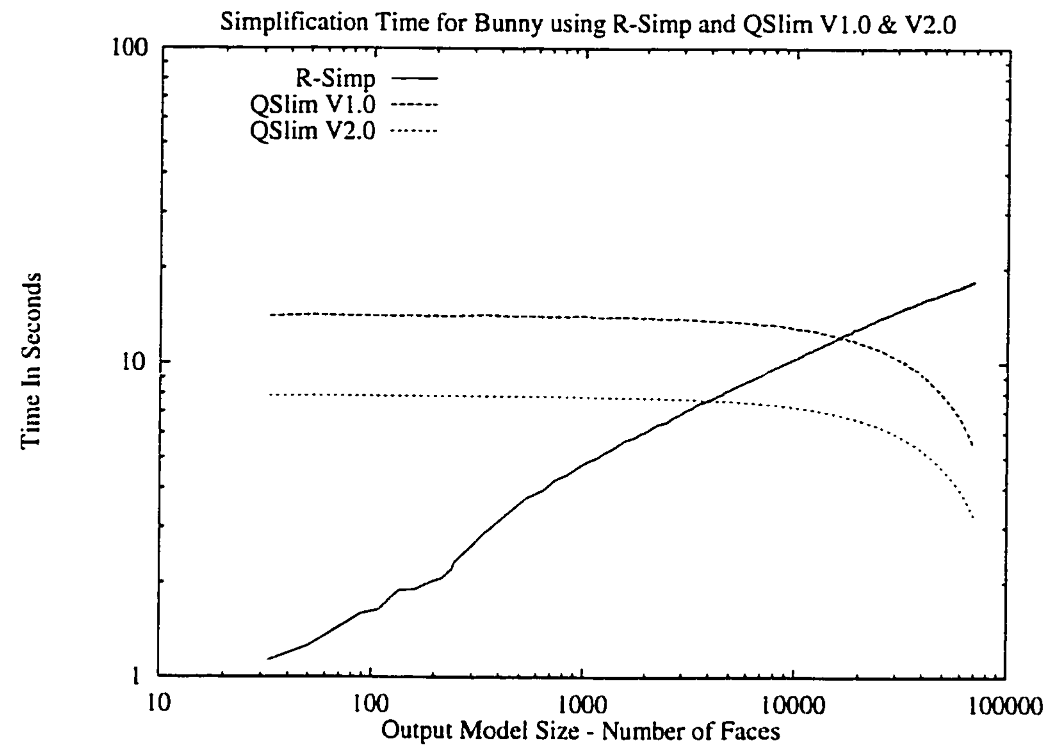
R-Simp is a fast simplification algorithm that ensures a minimum level of quality that is higher then many vertex clustering algorithms. R-Simp can be bounded at the same time by model size and simplification time because it simplifies in reverse. These two attributes make R-Simp an ideal tool for previewing very large models in modeling, visualization, and CAD/CAM applications, because such previews need to be produced quickly. Another application of R-Simp is in a two step simplification process. In the first stage, R-Simp would simplify a very large model down to a moderate size. In the second stage, an algorithm that emphasizes quality could be used to simplify the model further down to a very low polygon count.

6.9 QSlim Version 2.0

All the above comparisons were done with QSlim version 1.0. Shortly before this thesis was finished, QSlim version 2.0 was released. The main difference between versions 1.0 and 2.0 is that version 2.0 was recoded and optimized for speed; algorithm complexity remained the same.

Figure 6.6 shows the results of version 2.0 as compared to version 1.0 and R-Simp. On average there was a two times increase in speed as the graphs show, but the complexity of the algorithm remains unchanged.

The comparisons presented in this chapter are still valid. The quality of QSlim is still the same, but R-Simp is now only 1.25 to 2.0 times faster on the models that were used for the comparisons. The graphs still show the same relationships; R-Simp is still considerably faster for creating low polygon count simplifications. As noted above, further optimizations of R-Simp for speed are still possible.



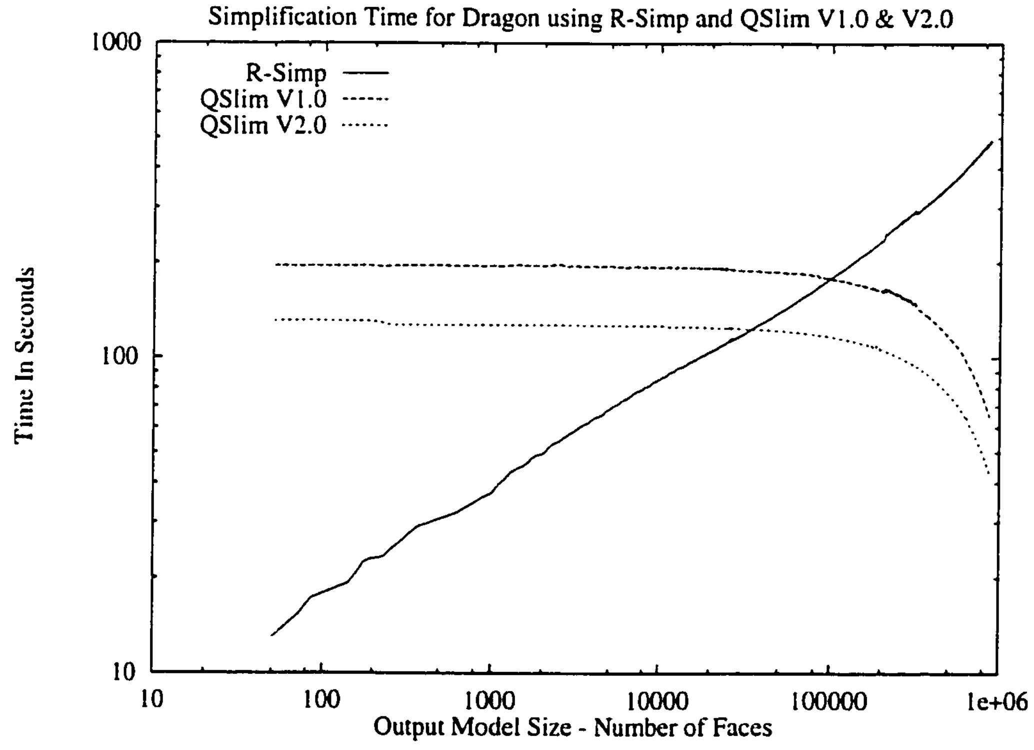
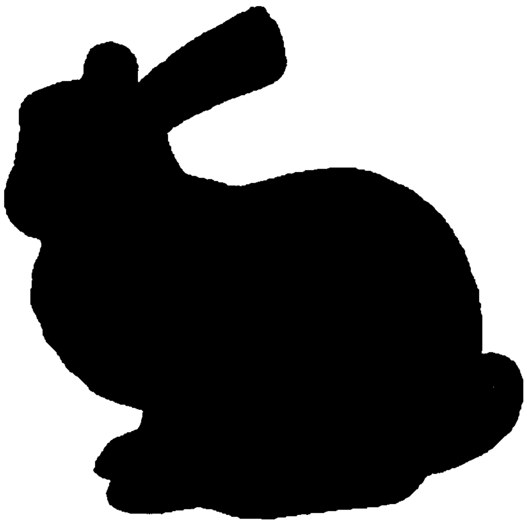
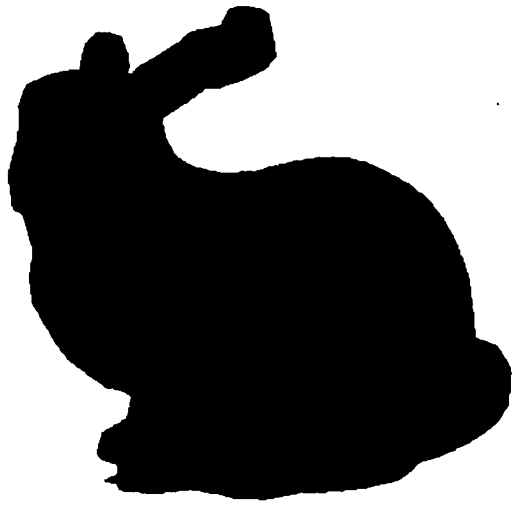


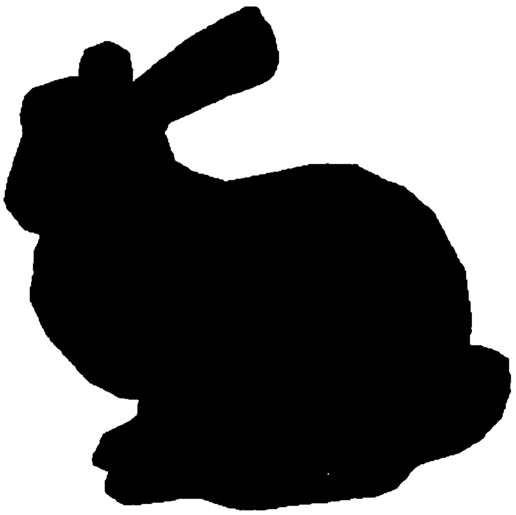
Figure 6.6: The effect of output model size on simplification time for the bunny and the dragon for QSlim version 2.0.



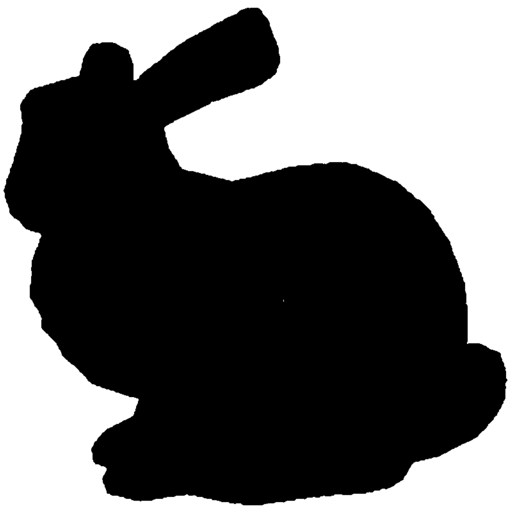
(a) Original: 69451 Polygons



(b) Vertex Clustering: 1602 Polygons

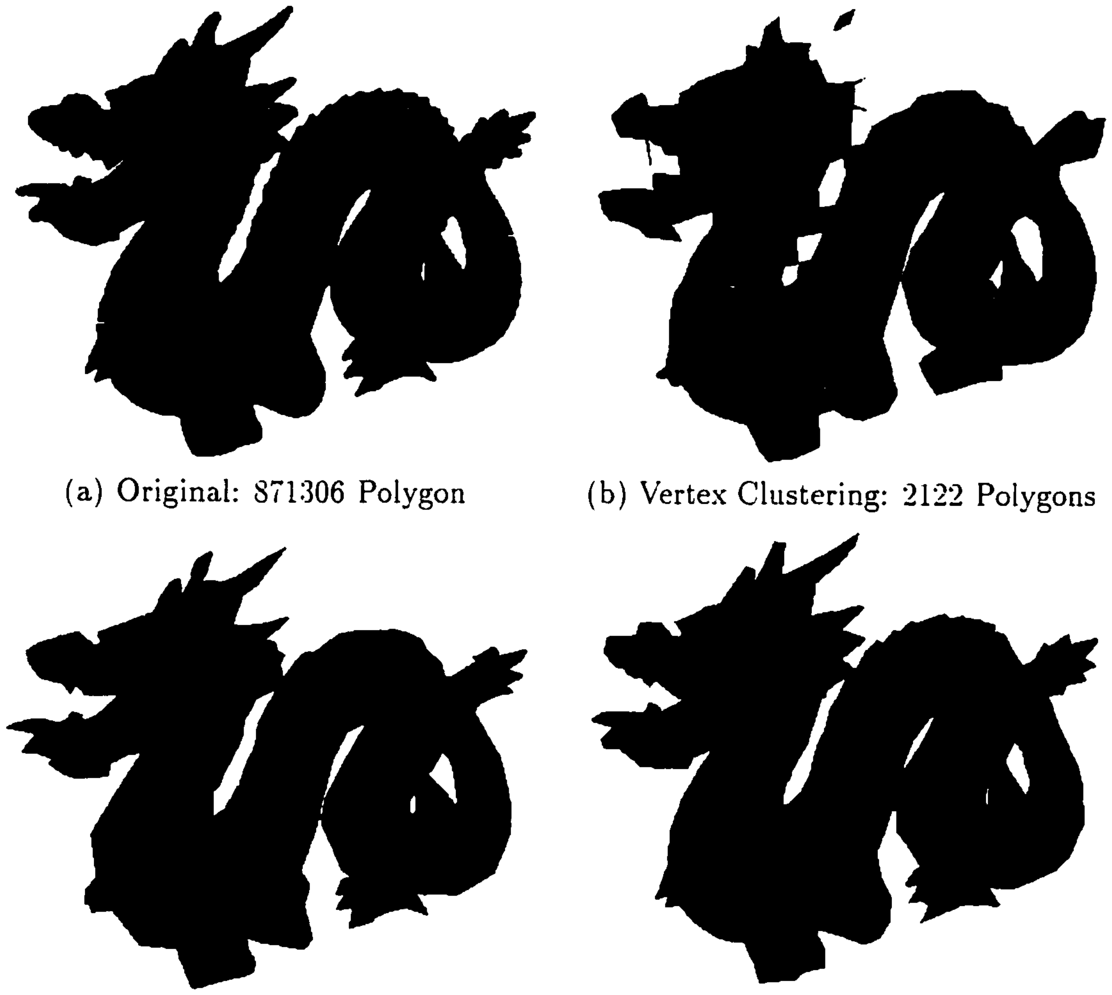


(c) R-Simp: 1601 Polygons



(d) QSlim: 1600 Polygons

Figure 6.7: The bunny model.



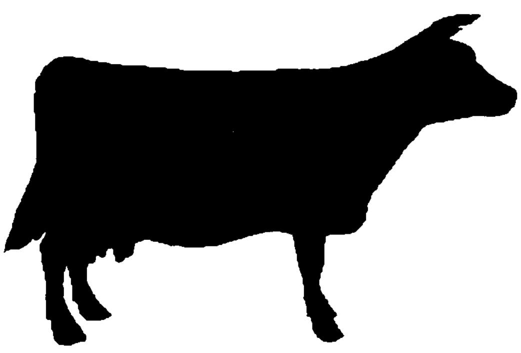
(a) Original: 871306 Polygon

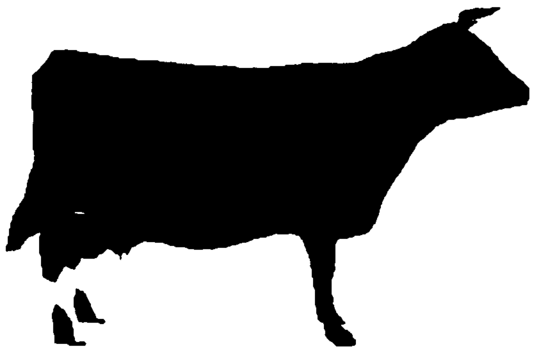
(b) Vertex Clustering: 2122 Polygons

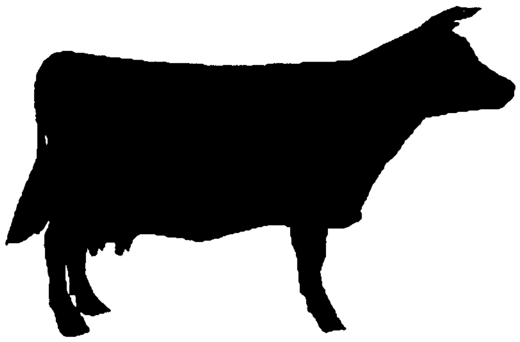
(c) R-Simp: 2068 Polygons

(d) QSlim: 2068 Polygons

Figure 6.8: The dragon model.


(a) Original: 5782 Polygons


(b) Vertex Clustering: 1598 Polygons


(c) R-Simp: 1600 Polygons

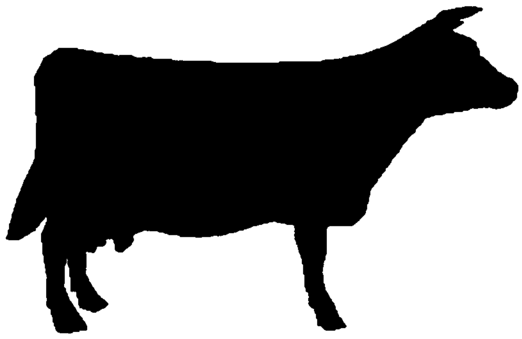
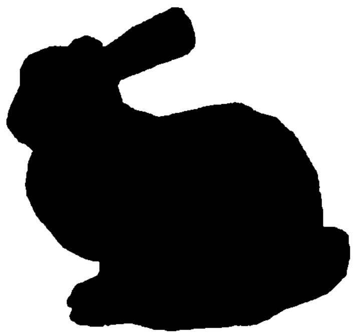

(d) QSlim: 1600 Polygons

Figure 6.9: The cow model.


(a) 1600 Polygons

(b) 1400 Polygons

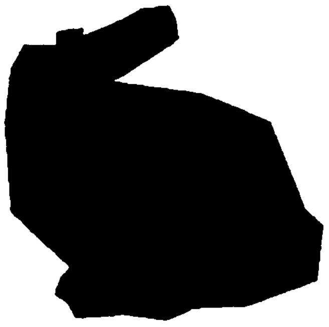
(c) 1200 Polygons

(d) 1000 Polygons

(e) 800 Polygons

(f) 600 Polygons

(g) 400 Polygons


(h) 200 Polygons

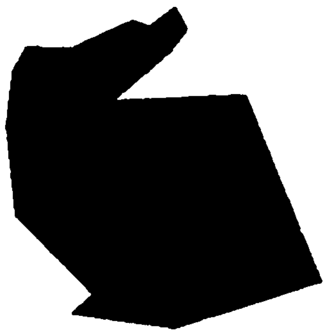
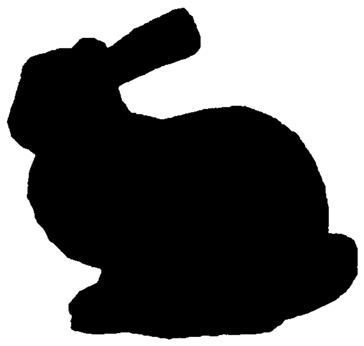
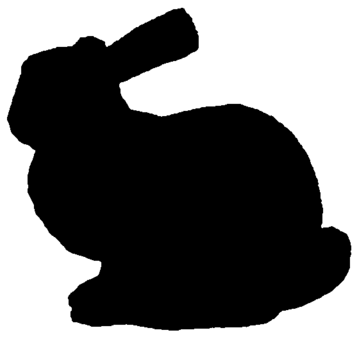
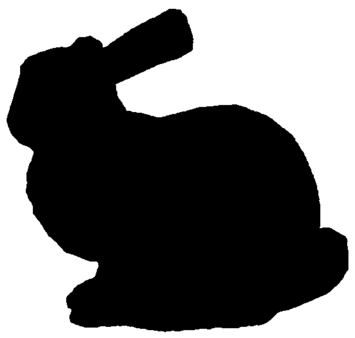
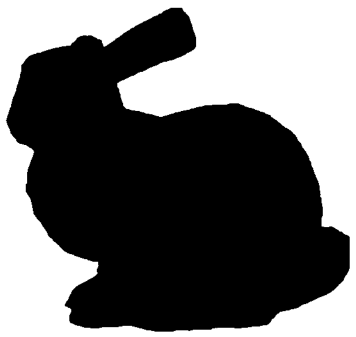

(i) 50 Polygons

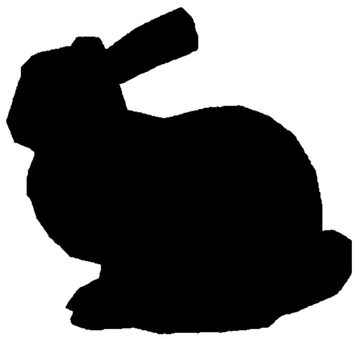
Figure 6.10: The visual quality of R-Simp generated simplifications decreases as polygon count decreases.

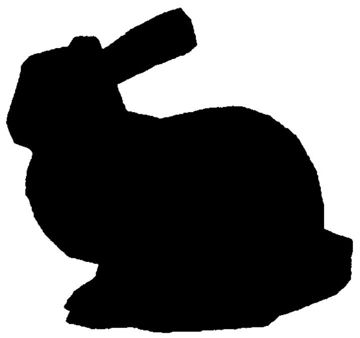

(a) 1600 Polygons

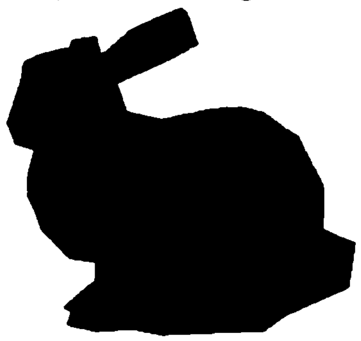

(b) 1400 Polygons

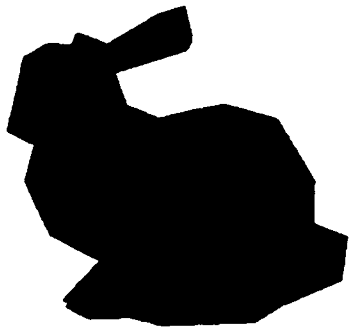

(c) 1200 Polygons


(d) 1000 Polygons


(e) 800 Polygons


(f) 600 Polygons


(g) 400 Polygons


(h) 200 Polygons

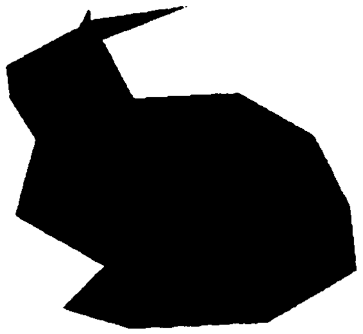

(i) 50 Polygons

Figure 6.11: The visual quality of QSlim generated simplifications decreases as polygon count decreases.

Chapter 7

Conclusion

This thesis presented a fast polygonal model simplification algorithm. This algorithm is considerably faster than many of the vertex merge and edge collapse algorithms and produces significantly better simplifications than many of the vertex clustering algorithms. This thesis also reviewed existing simplification algorithms and analyzed their strengths and weaknesses using a quantization taxonomy.

7.1 Summary of Objectives and Results

The main objective of this research was to create a fast simplification algorithm that produced reasonable quality simplifications. The other objective was to create an algorithm that was linear or nearly linear.

The result was an algorithm that was fast but did not sacrifice quality. For models of 2000 polygons or less it was two to four times faster then one of the fastest vertex merge algorithms [Garl97] and produced simplifications that were comparable in quality. The algorithm had a complexity of $n_{input} \log n_{output}$, which made it linear for constant output size. And since it simplified in reverse, from coarse to fine, it is able to adhere to both a vertex count constraint and a time constraint.

7.2 Summary

Throughout this thesis model simplification is viewed as quantization. Quantization provides an elegant taxonomy by which to classify existing algorithms and provides a different approach for finding solutions for problems in model simplification.

7.2.1 Existing Algorithms

Most existing algorithms fall into three classes of quantization algorithms.

- Vertex clustering algorithms are based on the *product codes* approach to quantization. They tend to be the fastest of all the algorithms but also tend to produce the worst simplifications.
- R-Simp is based on the *Splitting* algorithm. It produces better results but is slower than some vertex clustering algorithms.
- Algorithms such as [Kalv96, Schr92, Turk92] are based on the *pruning* approach to quantization. They are slower than R-Simp and vertex clustering algorithms but tend to produce better simplifications.
- *Pairwise nearest neighbour* based algorithms tend to produce the best simplifications, but also tend to be the slowest. These are mostly vertex merge and edge collapse algorithms.

7.2.2 R-Simp

The R-Simp algorithm uses vertex clustering techniques to track vertex connectivity and it employs *splitting* to simplify the model. *Splitting* works by iteratively subdividing cells that contain the largest distortion. Initially the codebook contains a single cell. The cell with the largest distortion is split. The process continues until the required distortion or codebook size is reached. The basic algorithm is outlined below:

1. Initially the model is contained in a single cell, its bounding box.
2. Select the cell containing the most distortion. The distortion measure is a measure of orientation change in a large surface patch. Intuitively, this measure gauges the coplanarity of a patch.
3. Split the selected cell. Splitting is based on a second measure of orientation change in a large surface patch that is similar to the measure of normal curvature.

(a) Compute the cell's normal curvedness. Determine the magnitudes and the directions of maximum and minimum curvedness.

(b) Split the cell; produce new cells. Depending on the magnitude of the minimum and maximum curvedness split the cell into two, four, or eight new cells. The positioning of the splitting planes is based on the direction of maximum curvedness.

4. Find the centroid of each new cell. This is a technique that minimizes the distance between the centroid and the associated planes in a cell. This approach is similar to the one used in [Garl97, Garl98].

5. Iterate, until the required codebook (model) size is reached. Each cell represents a vertex in the new model. After the required number of vertices is reached the vertices of the simplified model are triangulated to form the new model.

In practice, it was found that the initial cell could be subdivided into eight uniform sub-cells without any reduction of quality in the output model, which greatly sped up the simplification process. A simple topology check was used to separate disjoint components in a cell into separate cells. It was found that this topology check greatly improved the quality of the simplification.

7.3 Summary of Contributions

This thesis makes several contributions in the area of polygonal model simplification.

1. Most curvature measures measure the curvature in an infinitely small area around a point on a surface. It is difficult to determine the overall orientation change of a large surface patch using these curvature measures. Several large scale curvature measures, called here curvedness measures, were developed to measure orientation change in large surface patches. These measures do not measure curvature in the differential geometry sense.

   The first measure computes the orientation change on large surface patches. This measure is analogous to measuring the coplanarity of a surface.

The second measure approximates the normal curvature on a surface patch around a point; including the minimum and maximum direction of curvature.

2. Coarse simplifications are found more quickly than fine simplifications because R-Simp simplifies in reverse.

3. Simplifications are found in linear time for a fixed output size. The algorithm is $n_i \log n_o$, where n_i is the input model size and n_o is the output model size; most other algorithms are $n_i \log n_i$.

4. R-Simp's simplification algorithm is based on *splitting* from quantization literature. *Splitting* allows R-Simp to control the size and placement of clusters, thus enabling R-Simp to use vertex clustering to track vertex connectivity, which results in a fast algorithm that does not sacrifice quality.

7.4 Future Work

The research and development of R-Simp has uncovered several possible avenues for future research.

7.4.1 Level of Detail Hierarchies

With a minor modification R-Simp is able to create level of detail hierarchies. R-Simp already creates an implicit hierarchy, but the hierarchy is not kept due to memory constraints. By keeping the internal nodes and linking the children to the parents (and vice versa), a vertex tree could be created. This vertex tree could be used to produce discrete levels of detail, or with a few more modifications the vertex tree could be used by an adaptive display algorithm like [Lueb97].

7.4.2 Preservation of Colour, Texture, and other Attributes

A possible extension to R-Simp would be to make the simplification sensitive to attributes on the surface. Currently R-Simp bases its simplification decisions on the orientation change of normal vectors on a surface. Since colour and textures are also vector quantities, it would be possible to include the change of these quantities in the simplification decision. Care must be taken to treat these attributes in an appropriate manner since they are not linear and usually do not coincide with the geometrical change of the surface. Both the optimal vertex placement policy and the procedure that decides how to partition a cell are easily expandable to include other vector quantities.

7.4.3 Progressive Transmission

As the web becomes more popular and models become larger there is a need to do progressive transmission of polygonal models. Two algorithms that do progressive transmission of polygonal meshes are [Hopp96, Taub98]. The problem with [Hopp96, Taub98] is that they create a fine to coarse mesh progression, which implies that the models need to be put through the algorithms beforehand. Because R-Simp simplifies from coarse to fine, it is ideally suited for creating simplifications for progressive transmission on the fly. The big hurdle that needs to be cleared is how to track vertices from one mesh to another. The problem is that the vertices of a coarser mesh are not a subset of the vertices in the finer mesh. Ideally if mesh m_1 has N vertices and a finer mesh m_2 has M vertices then to create mesh m_2 from m_1 only $M - N$ vertices should be sent. Currently all M vertices would have to be sent.

7.4.4 Search & Simplification

The quality of the simplification could be improved by adding a look ahead feature to the subdivision decision. Currently the cell with the largest distortion is selected to be subdivided. Quality might be improved by selecting a cell such that the reduction in distortion achieved by splitting it is greatest. This potential improvement in quality will come at some cost, since all leaves in the cell hierarchy would have to be split. In the end the increase in quality may not justify the decrease in speed.

7.4.5 Parallelization

Many of todays systems contain more than one processor. For R-Simp to take advantage of these systems it should be parallelized; i.e. modified to use many processors at once. Currently, R-Simp does not lend itself naturally to parallelization. R-Simp always splits the cell with the greatest distortion first, implying that the selection and the splitting have to be done serially. For example, cell C_a is selected to be split and cell C_b is next in line. If the sub-cells of C_a have less distortion than C_b then C_a and C_b can be split in parallel, but if the distortion of C_a's sub-cells is greater than C_b's then C_a and C_b cannot be split in parallel. Once sub-cells are created the computation that is done for each sub-cell could be parallelized, but the computation is minimal and so parallelization may not bring any benefit.

If the look ahead feature is implemented, then parallelizing R-Simp will definitely be beneficial because each sub-cell created will need to be split and that can be done in parallel.

7.4.6 Memory Constraint

A memory constraint could be implemented that would work in the same way as the time constraint. The memory constraint would limit the amount of memory R-Simp could use when simplifying. If R-Simp runs out of memory it will return a simplified model that is smaller than the requested size. This constraint is useful on systems that have a small amount of memory.

7.4.7 Error Bound

Currently R-Simp uses a vertex bound as its stopping criterion. Instead, a geometrical error bound could used for a stopping criterion. A simple error bound would be the maximum distance from a vertex in the original model to the centroid of the cell it is in. This type of error bound would be useful in several instances. If only the error bound is used then R-Simp could be very easily and effectively parallelized because each branch of the tree can be expanded by a separate thread. The error bound could also aid in the traversal of a vertex tree for adaptive display.

Bibliography

[Algo96] Maria-Elena Algorri and Francis Schmitt. "Mesh Simplification". *Computer Graphics Forum*, Vol. 15, No. 3, pp. C77–C86, September 1996.

[Buch92] James L. Buchanan and Peter R. Turner. *Numerical Methods and Analysis.* McGraw-Hill, Inc., Highstown, NJ, 1992.

[Chaz91] B. Chazelle. "Triangulating a simple polygon in linear time". *Disc. and Comp. Geometry*, Vol. 6, pp. 485–524, 1991.

[Cign97] P. Cignoni, C. Rocchini, and R. Scopigno. "Metro: measuring error on simplified surfaces". Technical report, Istituto per l'Elaborazione dell'Infomazione - Consiglio Nazionale delle Ricerche, 1997.

[Cign98] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. "A general method for preserving attribute values on simplified meshes". *Proceedings IEEE Visualization'98*, pp. 59-66, 1998.

[Cohe96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick P. Brooks, Jr., and William Wright. "Simplification Envelopes". *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pp. 119–128, August 1996.

[Cohe97] Jonathan Cohen, Dinesh Manocha, and Marc Olano. "Simplifying Polygonal Models Using Successive Mappings". *Proceedings IEEE Visualization'97*, pp. 395–402, 1997.

[Cohe98] Jonathan Cohen, Marc Olano, and Dinesh Manocha. "Appearance-Preserving Simplification". *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pp. 115–122, August 1998.

[Corm90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 1990.

[Davi95] Harry F. Davis and Arthur David Snider. *Introduction to Vector Analysis.* Wm. C. Brown Publishers, Toronto, Ontario, 1995.

[Dill81] J. C. Dill. "An application of color graphics to the display of surface curvature". *Computer Graphics*, Vol. 15, No. 3, pp. 153-161, August 1981.

[El-S] Jihad El-Sana and Amitabh Varshney. "Controlled Simplification of Genus for Polygonal Models". *Proceedings IEEE Visualization'97*, pp. 403–412, 1997.

[Fole93] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Systems Programming Series, Don Mills, Ontario, 1993.

[Funk93] Thomas A. Funkhouser and Carlo H. Séquin. "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments". *SIGGRAPH 97 Conference Proceedings*, Vol. 27 of *Annual Conference Series*, pp. 247–254, August 1993.

[Garl97] Michael Garland and Paul S. Heckbert. "Surface Simplification Using Quadric Error Metrics". *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 209–216, August 1997.

[Garl98] Michael Garland and Paul S. Heckbert. "Simplifying Surfaces with Color and Texture using Quadric Error Metrics". *Proceedings IEEE Visualization'98*, pp. 263–269, 1998.

[Gers92] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[Gopi98] M. Gopi and D. Manocha. "A Unified Approach for Simplifying Polygonal and Spline Models". *IEEE Visualization Proceedings*, pp. 271–278, 1998.

[Guez95] Andre Gueziec. "Surface Simplification with Variable Tolerance". *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pp. 132–139, November 1995.

[He95] Taosong He, L. Hong, A. Kaufman, A. Varshney, , and S. Wang. "Voxel-Based Object Simplification". *Proceedings IEEE Visualization'95*, pp. 296–303, 1995.

[Heck97] Paul S. Heckbert and Michael Garland. "Survey of Polygonal Surface Simplification Algorithms". Technical report, Carnegie Mellon University, Draft Version, 1997.

[Hink93] P. Hinker and C. Hansen. "Geometric Optimization". *Proceedings IEEE Visualization'93*, pp. 189–195, 1993.

[Hopp93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. "Mesh Optimization". *SIGGRAPH 93 Conference Proceedings*, Vol. 27 of *Annual Conference Series*, pp. 19–26, August 1993.

[Hopp96] Hugues Hoppe. "Progressive Meshes". *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pp. 99–108, August 1996.

[Hopp97] Hugues Hoppe. "View-Dependent Refinement of Progressive Meshes". *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 189–198, August 1997.

[Hopp98] Hugues Hoppe. "Smooth View-Dependent Level-Of-Detail Control and its Application to Terrain Rendering". *Proceedings IEEE Visualization'98*, pp. 35–42, 1998.

[Isle96] Veysi Isler, Rynson W.H. Lau, and Mark Green. "Real-time Multi-Resolution Modeling for Complex Virtual Environments". *Symposium on Virtual Reality Software and Technology*, pp. 11–19, July 1996.

[Kalv96] Alan D. Kalvin and Russell H. Taylor. "Superfaces: Polygonal Mesh Simplification with Bounded Error". *IEEE Computer Graphics and Applications*, Vol. 16, No. 3, pp. 64–77, May 1996.

[Lewi91] Harry R. Lewis and Larry Denenberg. *Data Structures & Their Algorithms.* Harper Collins Publishers, New York, New York, 1991.

[Lind98a] Peter Lindstrom. "A Survey of Multiresolution Techniques for Arbitrary Polygonal Meshes". Technical report, Georgia Institute of Technology, 1998.

[Lind98b] Peter Lindstrom and Greg Turk. "Fast and Memory Efficient Polygonal Simplification". *Proceedings IEEE Visualization'98*, pp. 279–286, 1998.

[Lore87] W. E. Lorensen and H. E. Cline. "Marching Cubes: a High Resolution 3D Surface Construction Algorithm". *SIGGRAPH 87 Conference Proceedings*, Annual Conference Series, pp. 163–170, July 1987.

[Low97] Kok-Lim Low and Tiow-Seng Tan. "Model Simplification Using Vertex-Clustering". *1997 Symposium on Interactive 3D Graphics*, pp. 75–82, April 1997.

[Lueb97] David Luebke and Carl Erikson. "View-Dependent Simplification of Arbitrary Polygonal Environments". *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pp. 199–208, August 1997.

[Lueb99] David P. Luebke. "A Developer's Survey of Polygonal Simplification Algorithms". Technical report, University of Virginia, (Unpublished draft) - CS-99-07, 1999.

[O'ne72] Barret O'Neill. *Elementary Differential Geometry.* Academic Press Inc., New York, New York, 1972.

[Ronf96] Remi Ronfard and Jarek Rossignac. "Full-range Approximation of Triangulated Polyhedra". *Computer Graphics Forum*, Vol. 15, No. 3, pp. C67–C76, C462, September 1996.

[Ross93] Jarek Rossignac and Paul Borrel. "Multi-resolution 3D approximations for rendering complex scenes". *Modeling in Computer Graphics: Methods and Applications*, pp. 455–465, 1993.

[Schr92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. "Decimation of triangle meshes". *Computer Graphics*, Vol. 26, No. 2, pp. 65–70, July 1992.

[Seku94] Robert Sekuler and Randolf Blake. *Perception.* McGraw-Hill, Inc., Toronto, Ontario, 1994.

[Taub98] Gabreil Taubin, Andre Gueziec, William Horn, and Francis Lazarus. "Progressive Forest Split Compression". *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pp. 123–132, August 1998.

[Turk92] Greg Turk. "Re-tiling polygonal surfaces". *Computer Graphics*, Vol. 26, No. 2, pp. 55–64, July 1992.

[Vere95] Oleg Verevka. "The Local K-means Algorithm for Colour Image Quantization". M.Sc. thesis, University of Alberta, 1995.

[Weis98] Eric W. Weisstein. *CRC Concise Encyclopedia of Mathematics.* CRC Press LLC., 1998.

[Xia97] Julie C. Xia, Jihad El-Sana, and Amitabh Varshney. "Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 2, pp. 171–183, April 1997.