SIMPLIFICATION OF POLYGONAL SURFACE MODELS

A Dissertation Presented

by

Tsung-Chin Ho

to the graduate school in partial fulfillment

of the requirements for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

State University of New York at Stony Brook

August 1999

State University of New York at Stony Brook

The Graduate School

Tsung-Chin Ho

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Joseph S. B. Mitchell, Dissertation Advisor
Professor, Applied Mathematics and Statistics

Alan Tucker, Committee Chair
Professor, Applied Mathematics and Statistics

Cláudio T. Silva
IBM T. J. Watson Research Center

Amitabh Varshney
Assistant Professor, Computer Science

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies & Research

Abstract

In this dissertation, I present the following surface simplification algorithms.

- Greedy-Cuts Algorithm: a surface simplification algorithm satisfying global error bounds, based on the "greedy" heuristic, growing a polygonal surface according to a scheme that attempts to add the largest possible triangle at each stage, subject to the given error bounds (which may vary over the surface of the input model).

- Simplified Patch Boundary Merging (SPBM) Algorithm: a simple, fast, and memory-efficient surface simplification scheme based on a three-step process: (1) patchification (partitioning of the surface into patches corresponding to similar normal vectors), (2) simplifying patch boundary curves, according to a global error bound, and (3) retriangulating the resulting patches, according to a simple ear-clipping algorithm, with heuristics designed to produce good quality triangulations. SPBM is found to be especially fast and memory-efficient, while producing simplifications that generally have high visual fidelity. Extensive experimentation shows that the algorithm compares favorable to the prior state-of-the-art.

- Topology Simplification Algorithm: a fast and memory-efficient genus reduction technique that allows simplification of topology and removal of small "features" while simplifying geometry. In contrast with many previous approaches, our method accurately preserves surface profile curves.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In computer graphics and geometric modeling, polygonal meshes remain a popular rendering primitive for solid objects in a wide variety of applications. Besides the major benefit of the simplicity of this representation, they are widely supported by commercial graphics hardware and software tools. In recent years, very detailed and complex surface models, with millions of triangles, can be easily produced with either hardware or software in various graphics applications. For example, the CAD models in computer-aided geometric design, the range data obtained from automatic acquisition devices in computer vision, the isosurfaces extracted from volume datasets through the marching cubes algorithm in scientific visualization. Unfortunately, this rapid growth in the complexity of surface models has overwhelmed the performance of current high-end computer graphics hardware. There is however hope since a highly detailed object representation is not always required for graphics applications. Many of the datasets generated from the above applications, such as from the marching cubes algorithm, maintain redundant primitives while

accurately describing highly complex geometry. *Surface simplification* is the replacement of a complex surface model, having many triangles, by a mesh having substantially fewer triangles that "approximates" the original model.

In the last few years, virtual reality walkthroughs and flythroughs of extremely complex environments have become of increasing importance in computer graphics. Virtual scenes contain meshes having many millions of triangles. With the limitation of rendering capacities of graphics engines, the reduction of geometric primitives rendered in each frame is crucial for efficient performance. It is common to define several versions of a model at various *levels of detail* (LOD) for accelerated rendering. The idea is to construct a hierarchy of approximations of a model and then select the appropriate level of representation from within the hierarchy, depending upon its perceptual importance in the virtual scene. For instance, a detailed mesh should be presented as the object moves closer to the viewer, and coarser representations should be used when it moved further away from the viewer. As model complexities grow at a rate beyond the phenomenal advances in graphics hardware, it becomes essential for software developers to discover better methods of utilizing multi-resolution hierarchies in order to speed up rendering. This explains the wealth of recent research in the areas of model simplification, level-of-detail control, visibility culling, and image-based rendering.

In addition to the advantage for increased rendering performance, surface simplification provides other benefits. When a large surface model cannot fit in the local memory, the level-of-detail generations can help speed up the transmission of this large model over networks. To enhance visualization performance, only the parts of an object close to the viewer need to be detailed.

The *view-independent* levels of detail producing uniform approximations are also not appropriate for these practical applications. *Adaptive approximations* or *selective-refinement-based* algorithms offer an elegant solution to the problem of handling varying of resolutions of the representation of an object. However, for the visualization and transmission of a highly detailed object, such as the surface of a molecule or a human head, one would like to display progressively better approximations as the object moves close to the viewer. A *discrete* level-of-detail representation cannot easily achieve this goal without producing cracks or artifacts. *Progressive-based* simplification algorithms, constructing a continuous level of detail for an object, provide a method to appropriately display those highly detailed objects. Simplification can also improve the efficiency of computationally intensive problems in collision detection, ray tracing, and radiosity calculation.

There is an abundance of recent work on polygonal model simplification. These prior methods are based on various approaches, including energy function optimization, controlled vertex/edge/face decimation, vertex clustering, refinement, greedy facets, and wavelets. Some of the most recent have introduced appearance-preserving simplification [11], progressive forest split compression [66], memory-efficient edge collapses [42], and methods that attempt to preserve other important surface attributes (e.g., texture, color, shape detail) during simplification; see, e.g., [9, 24]. These prior simplification algorithms have attempted to optimize various criteria, including feature preservation, visual fidelity, global error bounds, algorithmic speed, and memory consumption.

The goal of our research is to investigate many aspects of the simplification

of polygonal surface models, with emphasis on the tradeoffs between algorithmic speed, memory consumption, and visual fidelity. In this dissertation, we first focused on a simplification technique with bounded error by extending the "greedy cuts" approach used by Silva et al. [61, 62] to approximate dense terrain grids with Triangular Irregular Networks (TINs). After decomposing a 3D surface into several terrain-like patches, we then adopt a novel triangle feasibility test and a modified greedy-cuts algorithm to triangulate every terrain-like patch.

To accelerate further the triangulation process and preserve the sharp features of the original surface, we also propose a simple, fast, and memory-efficient surface simplification technique, the "Simplified Patch Boundary Merging" (SPBM) method. Our algorithm can be decomposed into two main phases. The first phase consists of a "patchification" algorithm, which partitions a given polygonal surface into multiple "patches", based on a user-defined parameter (which bounds the allowable variation in the surface normal over each patch). In contrast with the prior "SuperFaces" approach [39] each patch is represented only by its boundary curve. Actual level-of-detail generation is then performed in three phases: (1) simplification of the polygonal curves bounding the patches, (2) selective patch merging, and (3) patch triangulation. We show through experimentation that SPBM results in a substantial speed-up, averaging a factor of two and a half times over a leading publicly available code (QSlim2.0), while consuming less memory and still producing very good visual fidelity (in some cases better, in some cases slightly worse).

Most simplification algorithms are designed to preserve the topology of the

original models. While preservation of genus is important in some applications such as molecular modeling, it is not required in many applications that demand only a visually close approximation of the original model; preserving topology may impose undue restraints on the degree of simplification. For example, during an interactive flythrough of CAD models that have a large number of small holes or tunnels, most or all topology-preserving simplification schemes are incapable of achieving desired reduction levels.

As an extension of our SPBM algorithm, we propose a simple, fast, and memory-efficient method for identifying holes, bumps, and cavities, followed by a controlled patch simplification scheme. Our method is able to remove "small" holes, bumps and cavities, subject to a user-specified error tolerance, while maintaining a high level of visual fidelity. Our method is based on a "patchification" process that partitions a given polygonal surface into multiple sets of contiguous polygons, again with bounded error, in conjunction with a set of simple heuristics designed to facilitate the production of "well-shaped" patches, with boundaries that respect surface discontinuities. Afterwards, we search the set of "sharp" edges that bound patches to identify a set of candidate loops that might bound holes, bumps, or cavities. By investigating the local geometry and the global topology, we are able to identify holes, bumps, and cavities that are candidates for removal (subject to a size condition, determined by an error tolerance). After removing loops that bound holes, bumps, and cavities, we merge the resulting patches, and output a new model based on a triangulation of the final set of patches.

The main benefits of our new method include *speed*, low memory consumption, one-step simplification, and visual fidelity. In contrast with many

previous approaches, our method is able to maintain accurately the profiles that contribute most strongly to surface discontinuities, while eliminating small features.

This dissertation is organized as follows: In Chapter 2 we introduce the terminologies used in later chapter and overview the previous simplification works. In Chapter 3 we introduce our novel Greedy-Cuts algorithm for bounded error surface simplification. The SPBM scheme is then presented in Chapter 4. Our preliminary results on the topology simplification technique are discussed in Chapter 5. Finally, we summarize our work and indicate a path for some future research in Chapter 6.

Chapter 2

An Overview of Surface Simplification

In the previous chapter we have introduced the importance of surface simplification for several different graphics applications. In this chapter we intend to overview the techniques that have been explored on the topic of surface simplification. In the last few years, there has been extensive research on simplification of general 3D objects. However, no single algorithm can satisfy all kinds of models under all conditions. Although each simplification algorithm has its own specialty, we classify, based upon the procedure of generating reduced meshes, these algorithms into roughly two categories: *iteration methods* and *surface-partition methods*. The iteration methods locally reduce *primitive elements*, such as vertices, edges, and faces, one by one. The basic steps are specified as follows:

- Evaluate each primitive element by some decimation criteria and build a

candidate list. The various decimation criteria include a global or a local error evaluation and the preservation of geometric or attribute features.

- If the candidate list is empty, then exit; no further simplification is possible. Otherwise, remove from the surface the first element in the candidate list; this leaves a hole.
- Retriangulate the hole.
- Reevaluate those primitive elements that have been affected by this change, and update the candidate list.

Surface-partition methods deal with the whole surface at one time. These methods perform, roughly, the following elementary steps:

- Partition the whole surface into several regions by satisfying some decimation criteria.
- Simplify or refine the borders of each region.
- Retriangulate each region.

2.1 Iteration Methods

Based on different primitives, we further classify the iteration techniques.

Vertex decimation

Schroeder et al. [58] use vertices as the primitive elements. The decimation criterion for each interior vertex is the distance to the average plane determined

by its adjacent faces. A vertex is removed only if this operation preserves local topology and the value at this vertex is smaller than a specified threshold. The hole created by vertex removal is retriangulated by a recursive loop-splitting procedure, which splits the border polygon by a split plane and produces triangles with good aspect ratios. The border vertices are reevaluated. If this splitting procedure fails, this vertex is not removed. The resulting vertices of the simplified mesh are a subset of the original mesh. Furthermore, by setting different thresholds, levels of detail are created and the vertices of the simplified model are also a subset of the previous simplified model. Unfortunately, this algorithm estimates the error between the current iteration and the previous iteration, not the original mesh. With this concern, Sourcy and Laurendeau [64] [65] developed a vertex decimation algorithm with strict error bounds. This method evaluates the error of a simplified approximation with the original mesh by storing a set of deleted vertices with each triangle, and adopts a constrained Delaunay triangulation to retriangulate the resulting hole. Klein et al. [40] described a similar approach to this method.

"Simplification envelopes" were introduced by Mitchell and Suri [49] for the purposes of simplifying surfaces to within an error bound ϵ, based on fitting the approximating surface within the ϵ-offset of the input surface, obtained by convolution with a disk of radius ϵ. In special cases (e.g., convex surfaces), Mitchell and Suri were able to prove approximation bounds for a "greedy" method by modeling the problem as a set cover problem. Cohen et al. [13] also used inside and outside envelopes, constructed using vertex normals, and have applied them to general surfaces using heuristic methods. The envelopes provide an accurate error estimation of the simplified mesh to guarantee that

all vertices of the reduced mesh are within a user-specified distance from the original mesh and that all original vertices are within the same distance from the reduced mesh. They design two algorithms, one local and one global, to generate reduced meshes. In their local algorithm, vertices are removed one by one. A vertex is removed if the hole created can be filled by some feasible triangles, which reside between these two envelopes. By setting various error values on vertices based, e.g., on a function of distance from the viewer, this method can perform an adaptive approximation of the original model. In their global algorithm, they were able to produce a list of ordered candidate triangles by comparing the number of vertices each feasible triangle covered, and then reconstructing the model by a greedy hole-filling process. Triangle overlapping can be avoided by explicit testing. This greedy approach can generate very good approximations, but, unfortunately, is time-consuming.

Edge decimation

Hoppe et al. [37] use edges as primitive elements. They create an energy function through the whole surface of the model and define three edge-based moves: edge collapse, edge split, and edge swap. An embedding function, used to simulate the moves, is established from the topological realization of the simplicial complex to the geometric realization of the initial model. Based upon legal moves, the algorithm produces a reduced mesh to optimize the energy function by iterative execution of two nested loops. The energy function includes three terms: a distance term, a vertex term, and a spring term. The distance term and spring term measure the geometric error between the simplified mesh and the original. The vertex term is to penalize a mesh

containing too many vertices. This idealized algorithm is theoretically elegant; however, for practical reasons, their implementation is based on a heuristic local approach, which randomly removes edges one by one and optimizes the local topology. A heuristic test, based on an angular check, is applied to the new triangulation to help to avoid mesh self-intersections.

André Guźiec [25] develops a novel error approximation called "error volume," based upon the edge collapse operation. The *error volume*, computed from the union of spheres of varying radii, is constructed dynamically as the border vertices are reevaluated. An iteration of this algorithm collapses the shortest edge in the current mesh. Collapsing an edge creates a hole and then an internal vertex close to this removed edge is found by solving a linear program, for volume preservation. This algorithm uses a *compactness value* of a triangle defined as the following formula:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2},$$

where a is the area of the triangle and (l_0, l_1, l_2) are the lengths of the three sides. An edge is collapsed only if the compactness values of all triangles, formed by the border edges of the hole and the inner vertex, are larger than a user-specified threshold. Error volume is updated locally by accumulating distance errors among border vertices, and is minimized using linear programming. When the error volume is reduced below a user-specified tolerance volume, the simplification process stops. Mesh self-intersection has yet to be addressed in this approach.

By defining the *zone* of a vertex as the planes of the adjoining faces of this vertex, Ronfard and Rossignac [55] described another error estimation metric

for a collapsed edge, based on computing the maximum distance between the resulting vertex and its *zone* (the union of the zones of the edge's two endpoints). A similar approach introduced by Garland and Heckbert [23] approximates the set of planes using a *quadric error* metric. Although this method does not provide strict error bounds, it has been shown experimentally to be very fast while still producing approximated representations with high quality.

Instead of collapsing edges one by one, Algorri and Schmitt [4] adopted a mesh smoothing process by swapping edges based on a G^1-continuity criterion to identify edges in the mostly planar regions. All such edges are then collapsed simultaneously.

Triangle decimation

Hamann [27] proposes a data reduction scheme to generate reduced approximations for 3D objects. Their algorithm uses triangles as primitive elements. A candidate triangle T has to pass some local constraints: The vertices adjacent to T are projected onto the plane P containing the triangle T; then, the centroid of T must lie in the region formed by those projected vertices on the plane P. The candidate triangles are put in a priority queue, ordered by a weight that is based on the local curvatures at their three vertices and their interior angles. At each iteration of the algorithm, the candidate triangle with lowest weight at the current list is removed from the queue of candidates, and it is replaced by a single point, which lies at the center of the approximate surface. (The approximate surface is estimated by the distances of the adjacent vertices to the plane P.) The triangles adjacent to the deleted triangle

are removed and the resulting hole is retriangulated using the single point as the center vertex. Edge swapping is used to try to optimize the aspect ratios of the triangles after the retriangulation. The algorithm terminates when a specified percentage of reduction is reached or no more candidate triangles exist. Error estimation is not provided in this algorithm.

2.2 Surface-Partition Methods

Patch decimation

The algorithm of Hinker et al. [31] partitions the whole surface into groups of nearly coplanar polygons. Each group has an adjustable representative normal. A polygon is added into an adjacent group only if the normal of the polygon and the representative normal are within a user-specified tolerance. After the polygon is grouped, the average of these two normals is the new representative normal of this group. The boundary polygon of each group is found by sorting all edges in a group and then removing duplicate edges. The boundary polygons inside another boundary polygon must be carefully resolved. The boundary polygons are straightened by removing the co-linear vertices on the basis of the rate of change in their gradient. Finally, this algorithm triangulates each boundary polygon by a heuristic greedy triangulation procedure, which traverses the vertices of a boundary polygon in order. Although triangles with poor aspect ratios and self-intersected edges could occur, this heuristic method is very efficient.

Kalvin et al. [39] decompose the surface into the union of superfaces. They

attempt to group faces which can be projected onto an approximating plane without self-intersection. The grouping process starts from a seed face and adds adjacent faces, satisfying topology and error bound criteria, into this surface. If there exist ungrouped faces, then one such face is selected as a new seed face and the grouping process is repeated until all faces are assigned to a superface. Then, the border shared by any two superfaces is replaced by one line which connects two end points of this border. If this violates the error bound criteria, this line is recursively subdivided at the most deviant point. Since all faces in a surface can be projected onto an approximating plane, superfaces are retriangulated by searching a "star point" surrounded by a star polygon on this plane. If a star point does not exist, the superface is decomposed repeatedly to guarantee the existence of star points. This algorithm uses a distance error bound to measure the degree of simplification. A superface with holes is still a problem to be resolved.

Subdivision refinement

Lounsbery et al. [47] proposed a technique using wavelets to create multiresolution representations of arbitrary topological surfaces. This method requires that the input mesh have subdivision connectivity. Building on this work, Eck et al. [17] removed this constraint and developed an adaptive subdivision algorithm for smooth parameterizations of the original surface over a base mesh consisting of a small number of triangles. Their algorithm begins by partitioning the original surface into Voronoi-like patches and then generating a variant of Delaunay triangulation as the base mesh. Then, the algorithm creates a parameterization of the local surface for each base triangle. Finally, the

level-of-detail generations are achieved by recursively subdividing these base triangles subject to a user-specified error tolerance. This approach can spend a long time for the generation of the Voronoi-like patches and may also need to create many extra levels of the global subdivision in order to resolve small local features, because of the uniformity of the mesh subdivision.

To overcome the drawbacks of this previous approach, Lee et al. [41] designed a fast coarsification strategy to build the base mesh and allow for user intervention in the form of fixing vertices or edge paths in the original surface. Their algorithm adopted a hierarchical simplification based on vertex removal to create a parameterization of the original surface over the base mesh. This initial parameterization was further improved by a parameter space smoothing procedure based on a "loop subdivision" scheme.

Re-tiling

Turk [68] presents a surface sampling technique that is different from the procedures of the surface-partition method. The sampling algorithm randomly places new points over the whole surface. The regions of high curvature, which are estimated by heuristic radii of curvature, contain a high density of vertices. Next, those randomly placed points are refined by a relaxation procedure which moves each point away from all other nearby points by a heuristic curvature-adjusted radius of point-repulsion. To maintain the topology of the original surface, a method called *mutual tessellation* is applied to triangulate each polygon of the original surface along with those new points that lie on this polygon. Afterwards, the original vertices are removed one by one. This re-tiling method performs better on models representing curved surfaces than

on models containing sharp discontinuities.

2.3 Progressive Construction

Based upon the work of Hoppe et al. [37], Hoppe [35] improves the energy function by adding two terms, a scalar term and a discontinuity term, which preserve the scalar field and discontinuity curves while simplification progresses. Furthermore, Hoppe introduces a novel idea to generate a continuous level-of-detail representation and provide progressive transmission through a geomorphing technique. This algorithm generates a sequence of edge collapses by the priority of their estimated energy costs. A detailed mesh M^n is reduced to successively coarser meshes M^i by applying this sequence of transformations:

$$M^n \xrightarrow{ecollapse_{n-1}} M^{n-1} \ldots \xrightarrow{ecollapse_1} M^1 \xrightarrow{ecollapse_0} M^0. \quad (1)$$

By the invertibility property of the edge collapse transformation, the detailed mesh M^n can be retrieved from the base mesh M^0 by applying the dual operation of edge collapse, the vertex split transformation:

$$M^0 \xrightarrow{vsplit_0} M^1 \ldots \xrightarrow{vsplit_{n-2}} M^{n-1} \xrightarrow{vsplit_{n-1}} M^n, \quad (2)$$

where $(M^0, \{vsplit_0, ..., vsplit_{n-1}\})$ is called a progressive mesh (PM) representation.

View-dependent construction

Instead of displaying the image with a series of sequential edge collapses, Xia et al. [70] construct a *merge tree* in a bottom-up fashion over the vertices of

the original object. This tree encodes two actions, edge collapse and vertex split, for each vertex with the perceptual distance metric. The tree structure is created from the totally ordered sequence of edge collapses based on edges' length. Care has been taken to prevent mesh fold over during the construction of the *merge tree*. Their algorithm incrementally produces view-dependent approximations in real time from one frame to the next by traveling through the merge tree. A similar approach by Hoppe [36], based on PM, develops a vertex hierarchy in a top-down fashion.

De Floriani et al. [15] introduce the multi-triangulation (MT) which uses a directed acyclic graph (DAG) encoding a partial order from the sequence of edge collapses as a guide to display the change of the resolution in a mutually independent way. The display of an approximation mesh corresponds to a *cut* in the DAG.

Progressive transmission

Guéziec et al. [26] also adopt a DAG to develop a surface partition scheme for the progressive transmission of large datasets. They propose to partition a surface in levels of detail (LODs) with a partial order given by the original sequence of edge collapses. In contrast with the view-dependent approach, the display of LODs is still in the same precedence as the original order of edge collapses.

2.4 Topology Simplification

Marching cubes

He et al.[32][33] create a topology simplifying algorithm which does not guarantee the preservation of local or global topology of a model but eliminates high frequency details in a controlled fashion. They establish levels of detail by converting an object into multi-resolution voxel representations using controlled low-pass filtering and sampling techniques. This algorithm assumes that the input model is a closed volume and then voxels can be determined to be in the interior or exterior of the object. In [32], the Marching Cubes algorithm [43] is adopted to generate triangles and a topology preserving algorithm is used as a post-process to remove redundant triangles. In [33], based upon the Splitting-Box algorithm [50], they introduce the AMC box for an adaptive Marching Cubes algorithm that recursively bisects the space until either an AMC box is found or a 2x2x2 box is reached. If an AMC box satisfies the quality criteria, triangles are produced accordingly. Finally, a stitching process is imposed to eliminate cracks generated amongst different levels of boxes. This algorithm also provides an error-controlled simplification, but it needs a great deal of memory and is time-consuming. Conversely, Shekhar et al. [60] establish an octree by traveling Marching cubes intersected with the surface of a model to delete redundant triangles. Then the octree is traversed level by level from bottom to top. At any level of the octree, if eight child cells meet the topology criteria, they are merged and replaced with a single parent cell.

Vertex clustering

Rossignac and Borrel [56] provide a general topology simplification scheme for general 3D models. Their algorithm uses a regular grid specified by the user to guide the subdivision of the object. All vertices falling within the same cell are merged together into a single vertex. A new triangulation is, therefore, created among these resulting vertices according to the topology of the original mesh. This method can achieve any level of reduction but can produce poor quality approximations. To improve the quality of approximations, Low and Tan [44] developed a modified grid partition technique based on a weighting of the vertices. They further enhance the visual appearance of the simplified images by rendering erratic edges using a thick-line primitive with varying pixel-width.

Vertex splitting

Popović and Hoppe [53] extend the Progressive Mesh (PM) representation to allow topological changes. They adopt the *generalized vertex split* and the *vertex unification* actions to form a *progressive simplicial complex* (PSC) representation. The base mesh generated by this approach consists of a single vertex.

Schroeder [59] gives a decimation algorithm based on *edge collapse* operations, while allowing "vertex splits" when a valid edge collapse is not available. This vertex split operation is poorly controlled since it does not localize its actions to individual holes, cavities, and bumps but performs them at all places in the model.

Edge collapsing

Garland and Heckbert [23] design their edge-collapse operation to allow the merging of pairs of unconnected vertices within a distance threshold. Pushing these ideas forward, Erikson et al. [21] propose a novel algorithm that automatically generates *virtual edges* based on an adaptive distance parameter. The merging of the unconnected regions is also controlled by preserving surface area. A view-dependent approach developed by El-Sana [20] creates a Delaunay triangulation among all the vertices and then generates a subset of the Delaunay edges as *virtual edges*. Their algorithm uses a spline-based distance metric combining the vertex positions and normals to synthesize the geometric and topology simplifications. To avoid mesh fold-overs at run-time, they propose an implicit dependency test based on the enumeration of vertices generated after each edge collapse.

Chapter 3

Greedy-Cuts Algorithm

3.1 Motivation

Silva et al. [61] introduced the *greedy cuts* method to simplify an input TIN surface into a new TIN with far fewer triangles. They defined both weak and strong feasibility tests for triangles in order to guarantee that the simplified mesh be within a specified error bound. Their method starts with a large polygon, enclosing the whole terrain to be triangulated, and then applies *greedy cuts*, which is a combination of *ear cutting*, *greedy biting*, and *edge splitting*, to triangulate the polygon inward within a specified error bound of the original surface. On each pass, the algorithm cuts from the perimeter the feasible triangle of largest area that fits the original data within a specified error tolerance. The main steps of their algorithm are as follows:

- Generate an initial polygon $\mathcal{P}$. This a simplified version of the standard min-link path method of Suri.

- Execute ear cutting on $\mathcal{P}$. An "ear"of a simple polygon $\mathcal{P}$ is a triangle that contains two adjacent edges on the polygon $\mathcal{P}$, and one diagonal edge internal to $\mathcal{P}$. This operation searches and cuts all possible ears of a simple polygon $\mathcal{P}$.

- Execute greedy biting on $\mathcal{P}$. This operation looks for a feasible triangle formed by an edge on the polygon $\mathcal{P}$ and one internal point of $\mathcal{P}$. Their algorithm uses a binary search on the grid points at the midpoint of the edge along the vector perpendicular to the edge.

- Execute edge splitting on $\mathcal{P}$ if both ear cutting and greedy biting fail to find a feasible triangle.

In general, the algorithm alternately performs ear cutting and greedy biting; ear cutting reduces the number of edges of polygon by one, while greedy biting increases it by one edge, and therefore, ear cutting will always be the final step of any run of the algorithm.

Our task was to extend this method to generate levels of detail for general 3D models. As addressed in [63], we can first decompose a 3D surface into several terrain-like patches, and then apply the greedy-cuts algorithm on each terrain-like patch. However, the greedy-cuts terrain simplification algorithm cannot be directly adopted to general 3D objects. Since the grid base is no longer available, the definition of a feasible triangle in their algorithm is not appropriate for general 3D cases. We shall propose a new definition of a feasible triangle for an arbitrary 3D surface. Moreover, an efficient and robust strategy is required to guarantee the termination of the greedy-cuts algorithm and to avoid skinny triangles. In the following sections, we begin by presenting

an efficient feasibility test for a general 3D surface based on two-sided Hausdroff metrics. We then implement several surface decomposition strategies and finally describe a new greedy-cuts algorithm for general 3D models.

3.2 Assumptions and Definitions

We assume a three-dimensional compact and orientable object whose triangular representation $\mathcal{T}$ has been provided to us. $\mathcal{T}$ is a well-behaved triangulation, which implies that every edge is either a boundary edge or shared by two adjacent triangles, no two triangles interpenetrate, and there are no "cracks" or T-junctions in $\mathcal{T}$.

In order to speak about a guaranteed error of approximation, it is necessary to have a precise notion of "distance" between the original surface and its approximation. In our work, we use the Hausdroff distance as an error measure.

Definition The Euclidean distance between a point x and a set $S \subset R^n$ is defined by

$$d(x, S) = \inf_{s \in S} d(x, s),$$

where $d(x, s)$ is the Euclidean distance between points x and s in R^n. The one-sided Hausdorff distance from the set S to the set T is defined by

$$d(S, T) = \sup_{s \in S} d(s, T).$$

This definition is not symmetric and in general $d(S, T)$ is not equal to $d(T, S)$. We say that $\mathcal{S}$ is ϵ-close to $\mathcal{T}$ with respect to the Hausdorff metric if

$$\forall t \in \mathcal{T}, \quad \exists s \in \mathcal{S} \qquad \text{such that} \quad d(t, s) < \epsilon \tag{3}$$

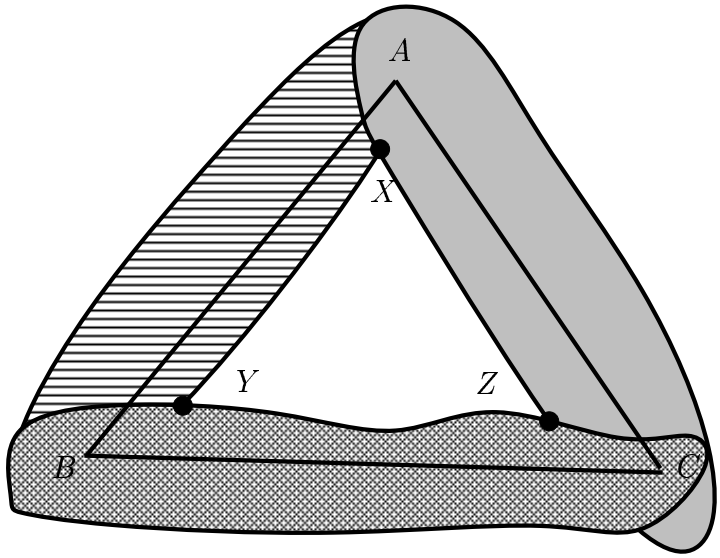


Figure 1: Three feasible regions.

and

$$\forall s \in \mathcal{S}, \quad \exists t \in \mathcal{T} \qquad \text{such that} \quad d(s,t) < \epsilon. \tag{4}$$

Note that with this definition the simplified mesh is also within a distance ϵ ("ϵ-close") to the original mesh. Given this triangular representation $\mathcal{T}$ and an approximation error bound ϵ, our goal is to generate, as quickly as possible, a topology-preserving approximation $\mathcal{S}$ which is ϵ-close to $\mathcal{T}$, using as few of the original vertices as possible.

3.3 Feasibility Test

In this section, based on the Hausdroff distance, we give a novel definition of feasibility for triangles in the simplified mesh.

Definition of triangle feasibility: A triangle P in the original mesh $\mathcal{T}$ is feasible with respect to a triangle Q in the simplified mesh $\mathcal{S}$ if $d(P,Q) < \epsilon$. A triangle Q in $\mathcal{S}$ is feasible if it can be completely covered by a connective patch of triangles $T \in \mathcal{T}$ that are feasible with respect to Q.

We design two phases of the feasibility test to investigate the candidate triangles which might be included in the simplified mesh. In Phase 1 we check the three edges of a candidate triangle Q and then the interior of Q, if necessary, is further examined in Phase 2.

Phase 1: Test the three edges of the tested triangle. This step runs three separate times, once for each of the three edges: $E_{A,B}$, $E_{B,C}$, and $E_{C,A}$. For each edge E, we first determine a region $\mathcal{A}$ on the original surface such that the Hausdroff distance from $\mathcal{A}$ to an tested edge E is smaller than ϵ, i.e. $d(\mathcal{A}, E) < \epsilon$, which is equivalent to condition (1). We say that the edge E passes the test if the region $\mathcal{A}$ covers this edge E within a distance ϵ, i.e. $d(E, \mathcal{A}) < \epsilon$. We mark these three regions $E_{A,B}$, $E_{B,C}$, and $E_{C,A}$ by mark 1, mark 2, and mark 4 respectively. (See Figure 1.)

Phase 2: If the three regions fail to cover the entire tested triangle, there exists an area bounded by these three regions. This uncovered area is tested in this phase. By identifying the three borders with different feasibility marks formed in Phase 1, several border lists are created for each region. The uncovered area is investigated and tested by searching inner points with the help of those

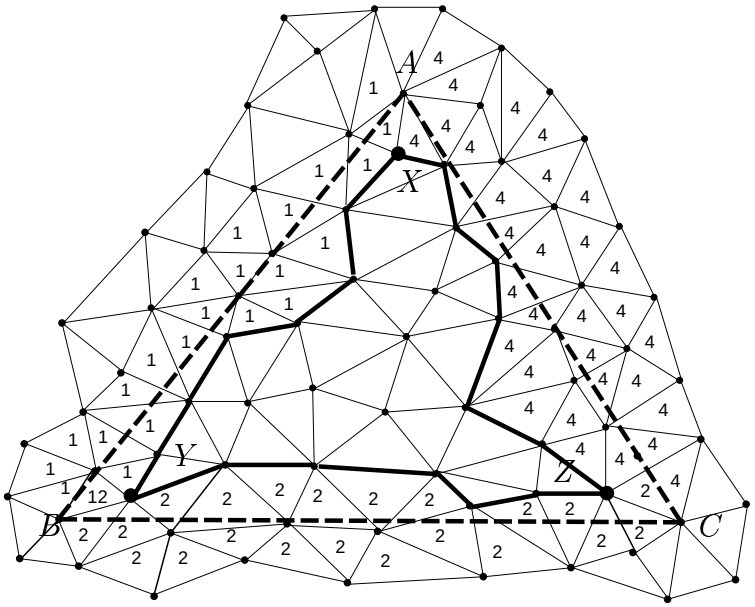


Figure 2: Three inner points.

border lists. For example, in Figure 2, points X, Y, and Z are the three inner points required. Each border path is composed of edges of border faces, and is guaranteed not to have holes.

3.3.1 Phase 1

For each of these three tested edges, $E_{A,B}$, $E_{B,C}$, and $E_{C,A}$, we attempt to determine a region on the surface so that for each face inside the region, the distance to the corresponding edge is less than ϵ. The first step of this phase is performed on edge $E_{A,B}$. We pick vertex A as the starting point and choose a feasible face F containing vertex A to be the starting face. We then perform a Breadth-First Search (BFS) of all faces on the surface starting with F. For

each face discovered in the search, we calculate the distance to the tested edge $E_{A,B}$. If the face is within a distance ϵ from the tested edge, we mark the face mark 1 and add it to the queue; if not, we ignore it. Inside the loop of the BFS, we dequeue one face from the BFS queue. Each of the face's three neighbors is examined, again calculating its distance from the tested edge and enqueuing it if it is close enough. The BFS continues this loop until the queue is completely empty. When the BFS is done, we need to determine whether or not the region we have grown extends from vertex A to vertex B. To check this, we only need to inspect the mark of vertex B. The mark of a vertex is defined by the mark of its adjacent faces. If any one neighboring face has the appropriate mark, then we define the vertex to have this mark as well. If the feasible region is extended to include vertex B, vertex B would have mark 1. If this first test succeeds, $E_{B,C}$ is the next to be tested and this region is marked 2. If $E_{B,C}$ succeeds, then $E_{A,C}$ is tested and this region is marked 4.

As the BFS proceeds during each of these tests, the border is recorded and grouped into different lists. Each list has the same feasibility mark. When a face is found to be infeasible during the BFS process, this face and its edge adjacent to a feasible face are assigned to the list that has the same mark as the infeasible face. If the infeasible face has not been marked any mark, then we created a list with feasibility mark 0. Those border lists will be further checked in Phase 2. As an example in Figure 3, a mark 1 border list on the boundary of the mark 4 region is created during the growth of the mark 4 region.

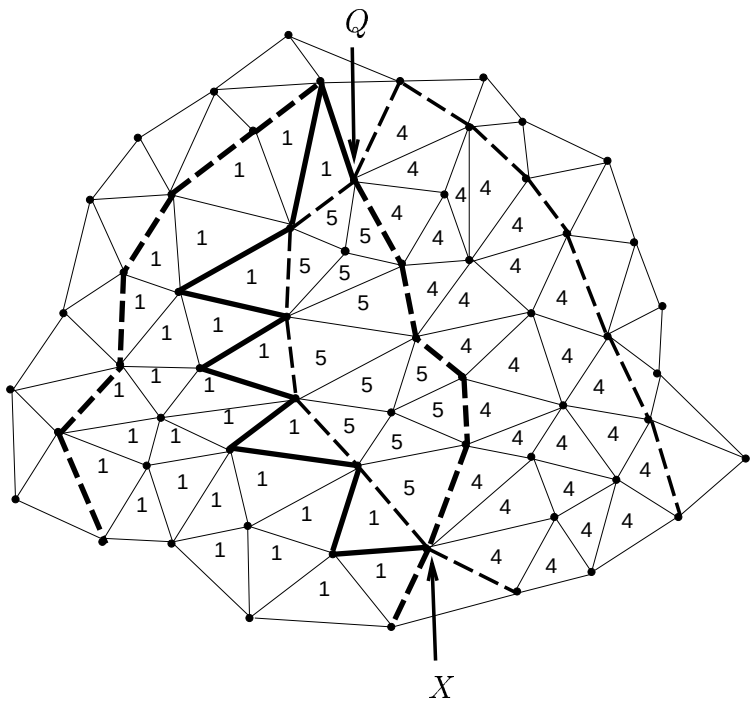


Figure 3: Two feasible regions overlap and a list of faces is found.

3.3.2 Phase 2

Although we have succeeded in covering the three edges of the tested triangle, we still need to identify the uncovered interior regions. These interior regions are bounded by two or three borders of the regions generated in Phase 1. In this phase, we begin by finding the so-called "inner points" which are the vertices at the intersection of the borders bounding the interior regions. In Figure 2, we have illustrated three inner points X, Y, and Z.

To search for these inner points we will need to examine all nonzero marked border lists built in Phase 1. Without loss of generality, we will use the regions

marked 1 and 4 to describe the detailed searching process. Figure 4 illustrates several possible types of overlap. We then impose three tests on the point X for each case.

Test 1: Check to see whether the point X belongs to the mark 0 border list of the mark 4 region. If it does, find another end point of this mark 0 border, called point R, and then go to Test 2. Otherwise, point X is not an inner point.

Test 2: Check to see whether point R belongs to the mark 1 border list of the mark 4 region. If it does, go to Test 3. Otherwise, point X **is** an inner point.

Test 3: Check the mark 0 border list of the mark 1 region to see if there exists a path from point R to point X. If there exists such a path then point X is an inner point; otherwise, it is not.

Figure 4 illustrates four basic cases corresponding to these three tests. In Figure 4(a), point X is not an inner point since the third (mark 2) region covers point X and hence Test 1 fails. Figure 4(b) demonstrates a successful case in Test 2 where the point R belong to a mark 0 border list of the third (mark 2) region. A situation may arise where the inner area is formed by only two regions as in Figure 4(d). This inner area is bounded by two mark 0 borders of mark 1 and 4 regions. Test 3 is to distinguish this case from the case in Figure 4(c). There is no mark 0 border of mark 1 region which links point R and point X, and hence point X is not an inner point.

When an uncovered inner area is discovered, we fill this area by beginning with these inner points. For each inner point, a zero-marked face adjacent to this inner point is chosen as the starting face. Then BFS is applied to search this uncovered area from this starting face. If the discovered zero-marked face

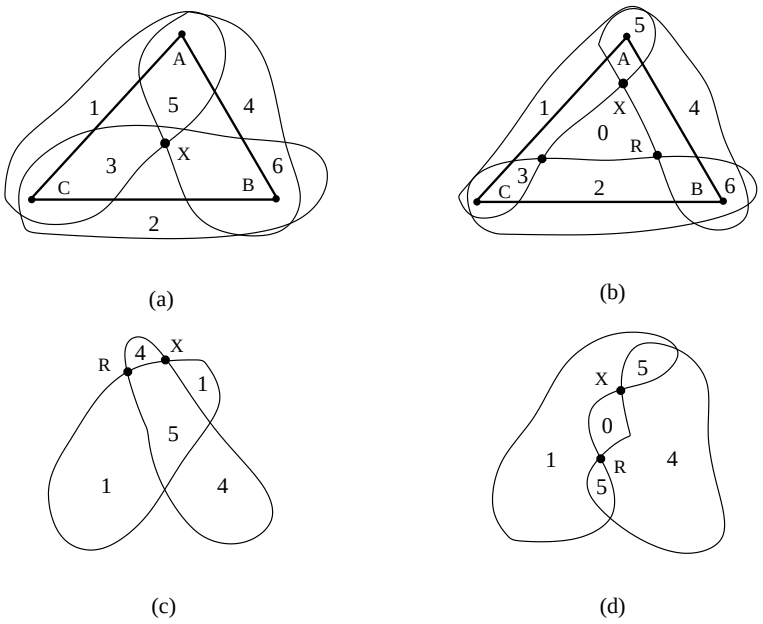


Figure 4: Four overlap cases.

is within a distance ϵ from the tested triangle, the face is marked with mark 8. Otherwise, the tested triangle is infeasible and the test process stops. The BFS process continues at other inner points until no zero-marked face is found. If all uncovered areas are marked, this tested triangle is feasible.

3.4 Greedy-Cuts Approach

Unlike terrain models, there is no base grid available for general 3D models. A possible way to extend the greedy-cuts algorithm to general 3D models, as addressed by Silva in his dissertation [63], is to decompose a 3D surface into several terrain-like patches. The boundary of each patch is a simple polygon.

The simplification can be achieved possibly by executing the terrain greedy-cuts algorithm to every patch. Each patch has an outward normal to guide the greedy-cuts process.

In order to explore the extension of the greedy-cuts algorithm, we first set up three main procedures, patch generation, ear clipping, and greedy biting, in a way similar to the operations implemented for terrain models, and used these as our guides for our search for more efficient algorithms. We use the feasibility test defined in the previous section for a triangle to define a feasible triangle within a specified error bound. Our purpose at this stage is to experience how the greedy cuts algorithm work for 3D objects, how the algorithm terminates or gets stuck, and how fast it could be. Now, we describe these three procedures as follows:

- **Patch generation**:
  - The surface of an object is divided into several connected patches. No patch is enclosed inside another patch. The boundary of a patch is a set of chains, which are lists of original edges. Ideally, each boundary chain is shared by exactly two patches, except in the open boundary case, in which a boundary chain belongs to only one patch. At this stage, we basically generate all boundary chains, and specify to which patches they belong.
  - Simplify all boundary chains according to the specified error parameter. This involves a two-dimensional curve simplification.
  - For each patch, link its simplified boundary chains to produce a simple polygon as a border.

- **Ear clipping**

  - For each polygon, this operation searches feasible "ears" to clip by traversing the boundary of the polygon. A feasible ear, containing two adjacent edges of the polygon, is confirmed by the feasibility test provided in the previous section. When an ear is clipped off, the polygon is shrunk and the corresponding patch is adjusted. For a terrain model, the investigation of an internal diagonal edge can easily be done in linear time by a simple traversal of the boundary of the polygon. Unfortunately, this property is not available for 3D models, and instead, we cut from the patch an area according to the test of the "feasible ear". The Breadth-First-Search process during the feasibility test is performed only on the faces belonging to the corresponding patch.

- **Greedy biting**

  - As the algorithm proceeds, if no more feasible ears exist and the simple polygon has yet to be closed, greedy biting is executed on the current polygon. Since there is no grid base for general 3D models, the search for the inner point requires special attention. At this moment, for a polygon P and each edge E of P, we simply search from among the endpoints of E for an inner point for which the resulting triangle has aspect ratio close to 1. We can impose an minimal (or maximal) threshold for the aspect ratio of the candidate triangles. If the triangle formed by the edge E and an inner point v is feasible, then we keep searching for inner points with better

aspect ratios for the resulting triangles. Otherwise, the other edges in P are investigated by the previously described procedure.

3.4.1 Patchification

Our first task is to decompose the surface of the original model into numerous terrain-like patches. We call this process as *patchification*. Here we assume the faces on the surface are all orientable triangles, although our approach can be directly applied to polygons. Our first approach in patchification is to generate big patches while maintaining the rough features of the surface. We would like to explore several different ways to investigate the shapes of the resulting patches. Generally, starting with a random face in the surface as the initial face, we use Breadth-First Search (BFS) as the order for picking faces from the surface, and then impose some collecting criteria to select faces into terrain-like patches. We choose BFS instead of Depth-First Search(DFS) since BFS collects adjacent faces first; and this way helps make round patches. Another approach will also be discussed later in chapter 4. If no more faces can be assigned to the current patch, we start from a face which has yet to be assigned in order to build a new patch. In the meantime, we also calculate the "patch normal" for each newly created patch in several different ways, one for each distinct criteria of patch construction. The patch normals are important when determining whether or not a triangle is feasible.

Part 1

A naive way to generate patches is to use the normal of the initial face as the patch normal. An adjacent face under BFS is collected if the inner product of its normal and patch normal is positive. The advantages of this method are the speed and simplicity. However, it could build numerous small patches if the models have high curvature. It won't be able to create only one patch for terrain models. We would like to generate as few terrain-like patches as possible. According to this consideration, we adopt an "angle range" approach. Instead of calculating the inner product of the patch normal and a face normal, we compute the "normal angles" of the three components, x, y, and z, of each face normal, setting the three "normal angles" of the initial face as the standard directions. Each time a new face is to be considered for addition into the patch, we calculate its three "normal angles" and enlarge the three "angle ranges" to include these three "normal angles". Although the "angle ranges" should not be wider than 180 degrees, we still allow two of the three ranges to be free. Given a new face, we verify that the newly enlarged "angle ranges" for those ranges that have not been freed, is not wider than 180 degrees. If an enlarged, and restricted, "angle range" failed this test, then we can free it as long as the number of free ranges is smaller or equal to 2. At the end of this procedure, at least one of the "angle ranges" will not be wider than 180 degrees.

A patch is done when no more adjacent faces from BFS can be added subject to these "angle ranges" constraints. We then start growing a new patch if there exists an unassigned face. This process repeats until all faces are marked. Every patch is assigned a number as well as the faces participating in

the same patch (i.e. the faces in different patches have different numbers). The boundary chains of a patch P are generated during the steps of building the adjacent patches P_i to patch P by collecting boundary edges shared by patches P and P_i. Each boundary chain is marked with the two "patch numbers" of patches P and P_i. We can therefore generate the borders of all patches immediately after the patchification process. The output of the patchification process is the set of boundary chains. To generate the patch normal for each patch, we simply compute the average of all the face normals in the same patch as the normal of the patch. Since our greedy-cuts algorithm is executed only on simple polygons, patches containing inner patches and/or holes are not allowed. Care has been taken to avoid this problem while building patches. The three vertices of faces previously added to a patch have been marked. Before we test the normal of a potential new face, we examine the marks of its three vertices. If these three vertices have been marked and it has more than one of its adjacent faces **not** belonging to the patch, then the face is rejected. (further details will be presented in chapter 4).

Part 2

After implementing the procedures described in Part 1, we found that the greedy collecting restriction might not allow the number of generated patches to reach the minimum number of patches into which the surface could be decomposed. Furthermore, it is likely to generate some "narrow" patches in conjunction with some "big" patches. It is not necessary to demand for the "smallest" decomposition of the surface, however, considering the difficulty of triangulating narrow patches, such patches should be avoided. It would

seem desirable to somehow enlarge the "small" patches while shrinking the adjacent "big" patches. This observation led us to a revision based on the "angle ranges" method.

Our strategy is to restrict the growth of the three angle ranges to some intervals determined by the assigned patch normal. We identify 26 types of patch normals, which are vectors (i,j,k) $\neq$ (0,0,0), i,j,k $\in \{0, 1, -1\}$. This means the original surface will be divided into 26 types of patches. When growing a patch, a patch normal N is **not** assigned to this patch in advance but is actually determined by the tendency of the growing patch. The addition of a new face to the patch makes the "angle ranges" increase in the same manner as in the previous algorithm. This new algorithm proceeds to build the patch proceeded as the previous one except that faces will only be accepted into the patch if the signs of their normal angles are "compatible" with the sign of the "angle ranges".

At first, new faces are added without restriction to the patch as long as the new "angle ranges" are not wider than 90 degrees. When the width of one of these "angle ranges" first exceeds 90 degrees but stays below 180 degrees, then a sign is given to that "angle range", i.e. that component of the patch normal. If the cosines of one "angle range" are positive then a +1 sign is given to that component of the patch normal. A similar condition gives a -1 sign but a 0 is assigned when the cosines of the "angle ranges" involve both signs. Checking now only those components where the patch normal has a +1/-1 sign, a new face is accepted if the sign of the cosine of its normal angle is the same as the sign of the patch normal. For these components, the width of the "angle ranges" must still be less than 180 degrees.

We have carefully implemented this dynamic algorithm, and found out that it might create many patches having normals (i,j,k) i = 0, or j = 0, or k = 0. We then introduced a "patch merging" procedure to further combine them into larger patches. The more advanced patch merging procedures are discussed in chapter 4. While merging patches, care has also been taken to avoid generating non-simple polygons. A simple constraint is imposed on the boundary chains shared by two patches; two patches are considered to be merged if they share only one boundary chain. Moreover, some boundary chains represent parts of the surface's sharp features, and these sharp features should be maintained. Two patches are therefore not merged if their shared boundary chain is sharp.

Figure 9 and Figure 10 shows two results from the implementation of these two algorithms. The "cup" model only generates four patches with normals: (0,0,1), (-1,0,0), (0,0,1), and (0,0,-1); the "mushroom" model generates 6 patches with normals: (0,-1,0), (0,1,0), (-1,-1,0), (0,1,0), (0,1,0), and (1,0,0) (The color patches are the initially generated patches and the patch borders indicate the result of merging small patches). They are the minimum terrain-like patches which can be used to cover these surfaces.

Before linking boundary chains to generate simple polygons for all patches, our algorithm adopted a method, depending on a user-specified parameter, to simplify all boundary chains. We used a greedy decimation procedure for curve simplification. Specifically, for a given boundary chain with n vertices $(v_1, v_2, v_3, ..., v_n)$, we dropped vertex v_2 if it was within distance ϵ of the segment v_1v_3; we then renumbered the vertices (v_i became v_{i-1}, for $i \geq 3$) and repeated. This procedure was applied successively on all boundary chains. To get a simplified boundary for a patch P, we linked boundary chains having

the "patch number" of the patch P, and the resulting simple polygon was oriented counterclockwise. A boundary chain is used twice in order to generate two simple polygons except when the surface has boundaries (or cracks), and in this last case the boundary chain is used once.

3.4.2 Patch Triangulation

We are now prepared to triangulate those simple polygons created by the patchification process. The main operators we adopted were *ear clip* and *greedy bite*. One *ear clip* reduces one edge of the polygon, while one *greedy bite* adds one edge to the polygon. Obviously, this implies the *ear clip* will be the last operation of the triangulation process.

Given a simple polygon P, we first execute *ear clip* sequentially on the edges of this polygon P. An ear (a triangle with two boundary edges) is clipped off from P if the ear is feasible according to the feasibility test. This *ear clip* action is repeated until all feasible ears are clipped. The polygon P was reduced by one edge for each successful clipping. If the polygon P hasn't been closed, we apply a *greedy bite* on the edges of the shrunk polygon P, which an inner vertex was found together with one boundary edge to form an feasible triangle. We then return to the polygon to perform as many ear clipping as possible. Those *ear clip* and *greedy bite* actions are repeated until the polygon is closed or no more feasible triangles are found.

What's the probability of failing to close a simple polygon? According to our experiments, it is likely to happen. In particular, We observed a situation where the edges of the shrunk polygon P were getting longer and increasingly thinner triangles were being generated by the successive ear clippings. It seems

that those long edges have to be split similar to the "edge split" action used in [61]. The question the is when and how to split an edge, and how many edges have to be split in order to finish the triangulation of a simple polygon. Can we always guarantee the closure of simple polygons? In the following section we present a "repair" process to overcome this problem.

3.4.3 Front Repair

In this section we explore another approach, referred to as the "wave front" method, to triangulate the simple polygons and overcome the difficulties of the simpler greedy-cuts approach. In the previous section we introduced a method where the cooperation of *ear clip* and *greedy bite* was used to triangulate a simple polygon, within a specified error bound. Whenever *ear clip* actions got stuck, we executed a single *greedy bite* action to rescue the process, and then returned to *ear clip* again.

The idea behind the "wave front" method is to imagine the triangulation of a simple polygon like the movement of a wave of water starting from the initial border and moving toward the center of the patch until it is closed. To implement this approach, still based on *ear clip* and *greedy bite*, we created a succession of *wave fronts*. These wave fronts then serve to control the operations of ear clip and greedy bite, and to repair bad choices made during those operations.

Given a simple polygon P, we define the initial *wave front* as P itself. To generate the "first wave front"we perform *ear clip* actions on P until no more feasible ears exist. At this point, the first wave front is the shrunk simple polygon P. If the polygon P has not been closed, we must generate

the next *wave front*. The first front move only consisted of *ear clip* actions while further movements of the front we will include *greedy bite* actions. In particular, the movement of the current *wave front* will include a sequence of *greedy cut* actions eventually followed by *ear clip* actions. We now take all possible *greedy bite* actions on the edges of the current wave front, not just a single *greedy bite*. With this arrangement, the termination of the wave front is not guaranteed since the edges of the new wave front could get longer with future *ear clip* actions and thus end the process in the greedy-bite stage with a stuck wave front. We therefore need a rescue method to reactivate the wave front. The direct way is to split long edges. The *edge split* action introduced by Silva et al. [61] split a stuck edge roughly in half or at a grid point close to the stuck edge, resulting in the division of one face into two smaller faces. However, since the locations of vertices in the general 3D surface are irregular and we did not introduce any Steiner points, splitting an edge may not possible due to a lack of vertices. This implies that we may need to delete more than one face to finish splitting an edge. There is another question, when should we split a stuck edge? If a stuck edge is not split right away, the future front moves could create many small faces, however, a stuck edge may be released (by a *ear cut* or an *greedy bite*) with future front movements. We have found that the appropriate time to repair a stuck edge is when it appears on a "sharp" turn of the wave front. These concerns have been incorporated into the following front repair process for each movement on the wave front.

Repair Process

According to our experiments, the wave fronts generated with *ear clip* and *greedy bite* might fail to advance because of the "sharp angles" of the front. Specifically, the problems occur when the following three conditions hold simultaneously (i) a sharp angle is formed one short edge and one long edge, (ii) the ear including these two edges is infeasible, and (iii) no feasible bites are available on these two edges. The idea of the repair process is to resolve the sharp angles of the wave fronts in advance, thus preventing the wave front from generating too many skinny triangles. The specification of a sharp angle depends on the requirement of a minimal aspect ratio of a triangle. In our approach we gradually released the constraint on sharp angles as the repair process proceeded.

The repair process was implemented immediately after the *greedy cut* and *ear clip* actions. Summarizing, we list the entire triangulation process for a simple polygon P.

- Initial wave front: the simple polygon P.
- First wave front move: clip the polygon until no more feasible ears exist.
- Further wave front moves: if the polygon P has not been closed after the first wave front move then continue to move the wave front W step by step until the polygon is closed. One step of the complete wave front move includes
  - Implement *greedy bite* actions on every edge of the wave front W.

- Implement *ear clip* actions on the current wave front W, until no feasible ears exist.
- Repair the sharp angles on the current wave front W.

The repair process includes

- Edge swap: an action to split an edge (see Figure 5).
- Refine faces adjacent to a vertex by performing a sequence of *face split* (see Figure 6).
- Delete vertices: the extreme result of a sequence of *face split* actions.

The algorithm repairs the sharp angle gradually from the angles with high *sharp degree*. A minimal aspect ratio bound can also be imposed to avoid extremely skinny triangles. If there are no more possible bites and clips, or no further possible repair under the current definition of a sharp angle, then the degree of the sharp angle is reduced, and we repeat these three procedures. We define the *front faces* as the recently generated faces connected to the vertices in the current front, the repair process is designed to only affect the front faces. A more comprehensive description of the repair actions is necessary.

When a sharp angle is detected, we try to repair this sharp angle from the longer edge E of the two edges which form this angle. First, we try "edge swap" as shown in Figure 5. The face (triangle) containing E is deleted from the list of faces which have been generated, and two triangles, *acd* and *bcd* are created if they are feasible. If this first attempt fails, then we refine the adjacent faces connected to vertex a. This refinement procedure executes a sequence of *face split* actions. A *face split* action does one *greedy bite* and one

(or two) *ear clip* as shown in Figure 6. First, the face *abd* is deleted from the *front faces*. *Face split* will then try to generate two feasible faces by executing one *greedy bite* followed by an *ear clip* on the edges *ad* and *bd*. If this procedure succeeds, we make another *ear clip* at *gcb* to finish repairing the sharp angle, i.e. to close the sharp angle. If the first *face split* fails, when for example no vertices are inside the face *abd*, we recursively split the next adjacent faces of vertex a. For instance, in Figure 6 the next deleted face would be *adh*. If this procedure fails on all adjacent faces of vertex a, then the vertex a is deleted. This is would be the extreme result of a sequence of *face split*.

If the wave front is stuck after the *ear clip* and *greedy bite* actions, we can repair the front and guarantee the closure of the polygon. Suppose the current wave front F is stuck. For every edge which is not original edge in the stuck front, we claim that it is possible to assume that there exists a path inside the *front faces* (see Figure 8(c)), which contains original edges connected to the two vertices of this edge. Even if this is not strictly true, it is nonetheless possible to reduce to this case. If some of the vertices in the path are out of the scope of the current *front faces*, then our repair process will move the current front back to enclose those vertices. As a special case, the adjacent faces to one of the vertices of such a bad edge will be deleted by our third repair step because these faces cannot cover any vertices of the original triangulation. A common example of this last case occurs when an edge on the wave front created by a *greedy bite* action and the generated triangle do not cover any vertices and therefore will be deleted.

Hence, we suppose this path is enclosed in the *front faces*. By linking those paths, there exists a loop L of original edges containing all vertices of F. If we

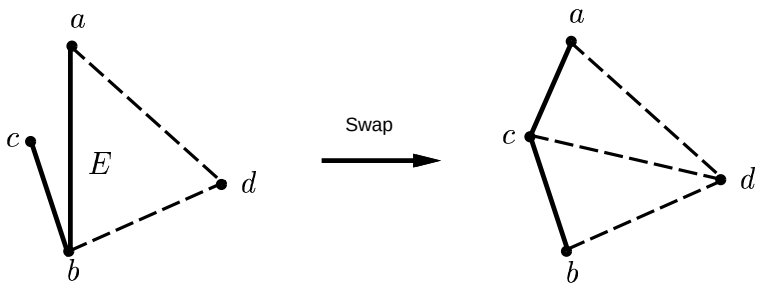


Figure 5: Edge swap in action.

can repair the current front F and move F back to L, then there exists at least one feasible triangulation of L, the obvious one formed by original triangles, to close the wave front, and which could be found by searching over all possible bites. The question is now can our repair process transform the current wave front F into the loop of original edges L? Those vertices belonging to L, but not in F, can be added by swapping edges or by refining the adjacent faces of vertices in F as shown in Figure 7 and Figure 8 . The stuck front F can therefore be transformed to L. Figure 11 demonstrates the entire greedy-cuts process on a single patch from the Fandisk model.

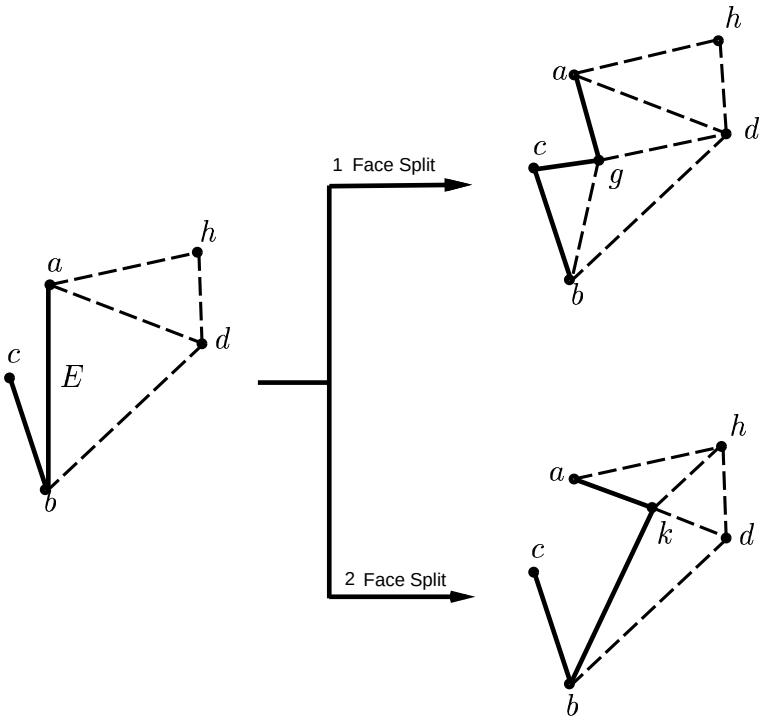


Figure 6: Face split in action.

3.5 Results and Conclusion

We have successfully implemented the greedy-cuts algorithm to general 3D models. The simplified mesh generated by our algorithm is ϵ-close to the original surface, as measured by a two-sided Hausdroff metric. Figure 12 shows the results of our experiments on the Cup, and Mushroom models. Unfortunately, we found that the greedy-cuts algorithm we implemented was not as fast as we had expected. In particular, when we increase the aspect ratio of a triangle to prevent it from generating skinny triangles, the front repair process took much longer to complete its task. Another reason for its low speed is that

the terrain-like patches were too big, i.e. the border (a simple polygon P) of a patch contained too many edges. To find a feasible ear and make a feasible bite, we had to avoid the newly generated triangle from intersecting other edges of the simple polygon P. This operation depends on the number of edges of the polygon P. Furthermore, although our algorithm gave a guaranteed error bounded simplification, the visual quality of the simplified mesh was not comparable to those of other leading simplification approaches when the number of simplified triangles was held fixed. In order to guarantee the simplified mesh within the error bound, our greedy-cuts algorithm generated many triangles to meet this requirement. The error bound had to be increased in order to reduce the number of triangles in the simplified mesh, hence resulting in poor visual quality. This is a common drawback of simplification algorithms based on refinement approaches with guaranteed error bounds, as can be seen in the currently existing algorithms mentioned in chapter 2. We also noted that the ability of maintaining the features of the original surface was a crucial factor for a high visual quality simplification. If we generated 'big' patches, some feature edges in the original surface might not be retained. With these observations in mind, in the next chapter, we explore a new approach and present an advanced algorithm for fast and memory efficient surface simplification.

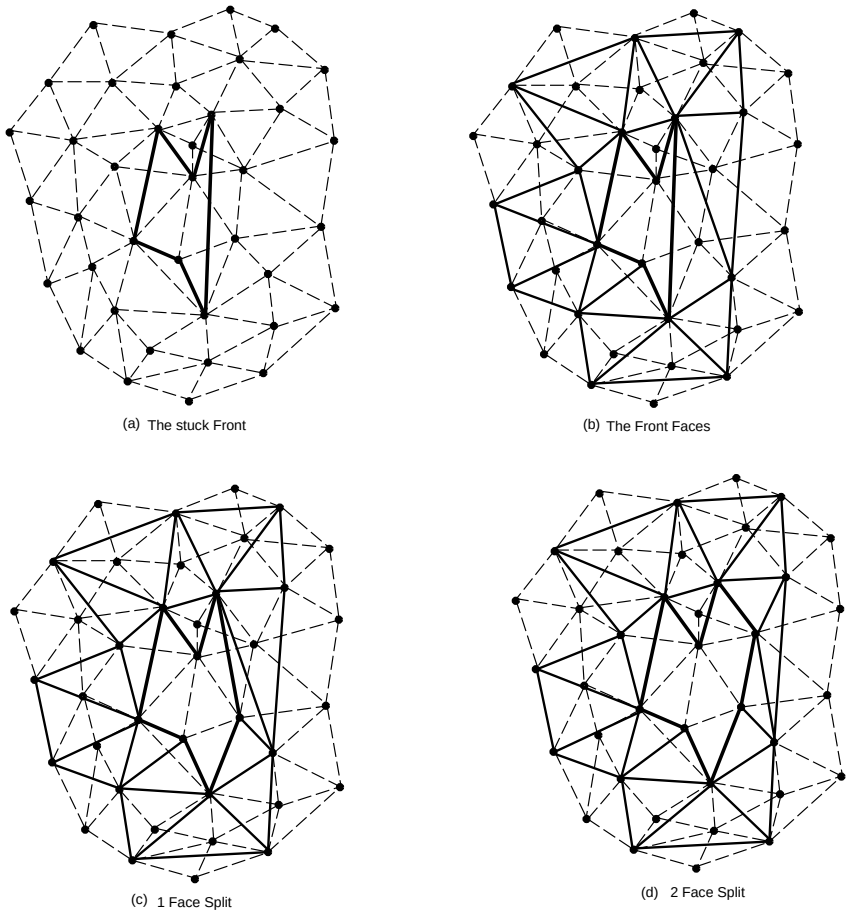


Figure 7: Front repair in action.

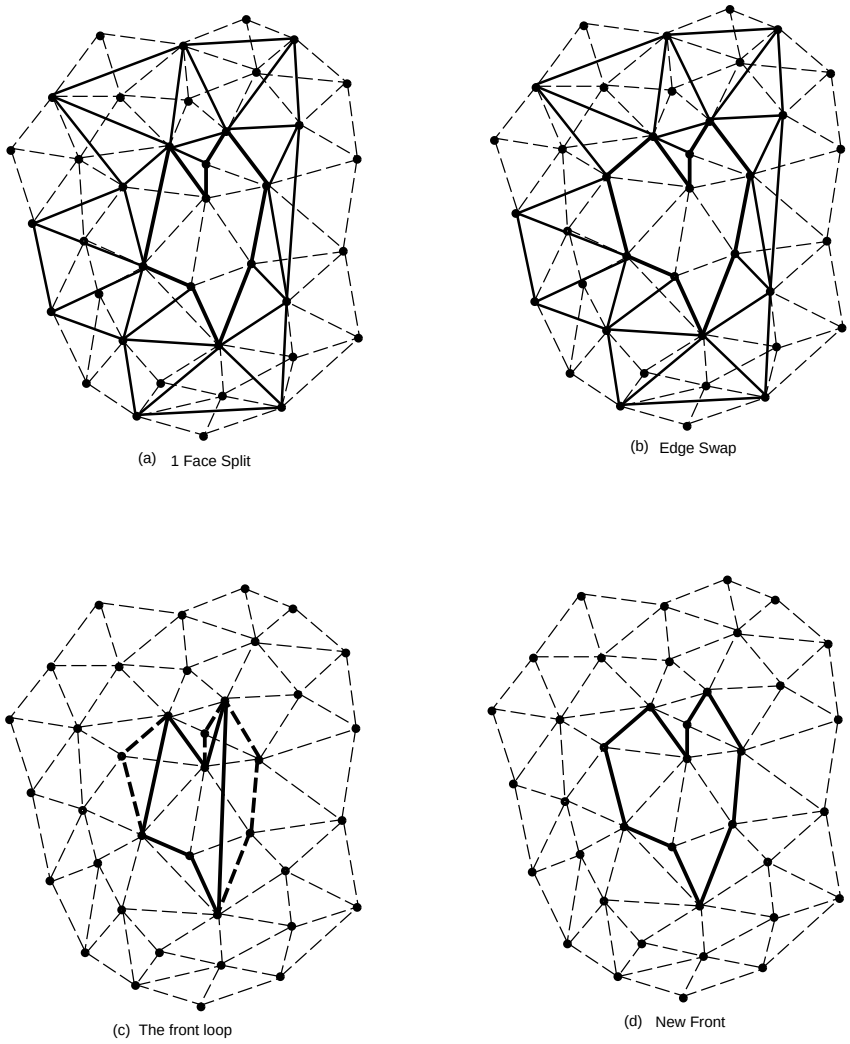


Figure 8: Front repair in action (continued).

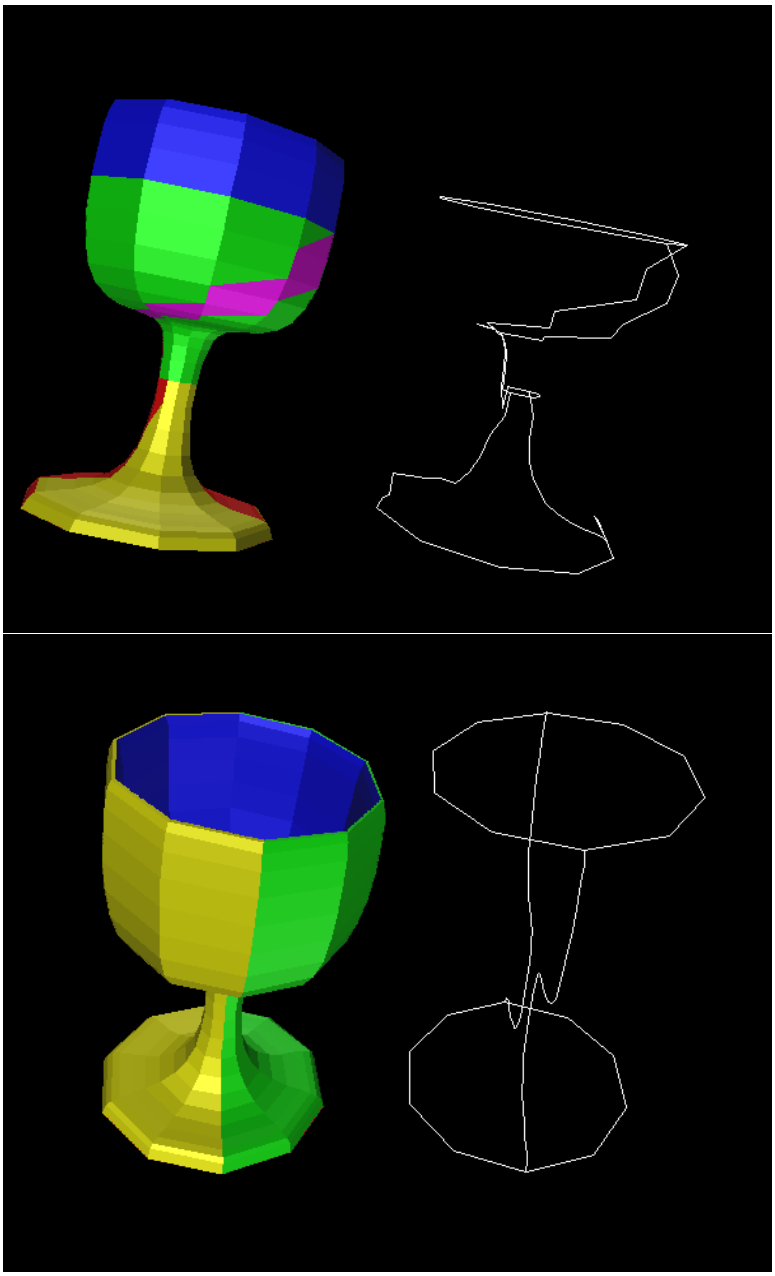


Figure 9: The Cup patches generated by two different patchification strategies: the upper image from Part 1 and the lower image from Part 2.

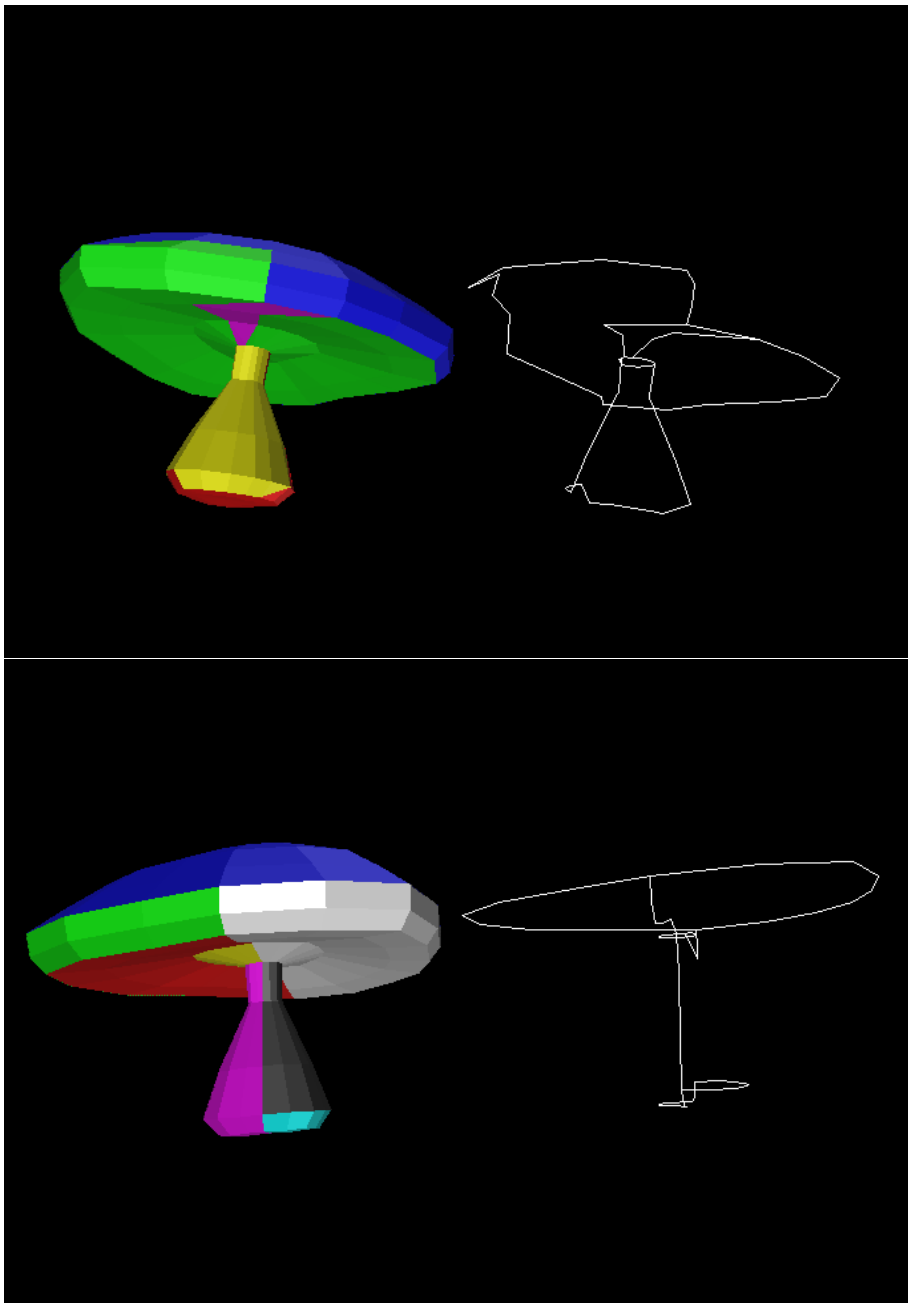
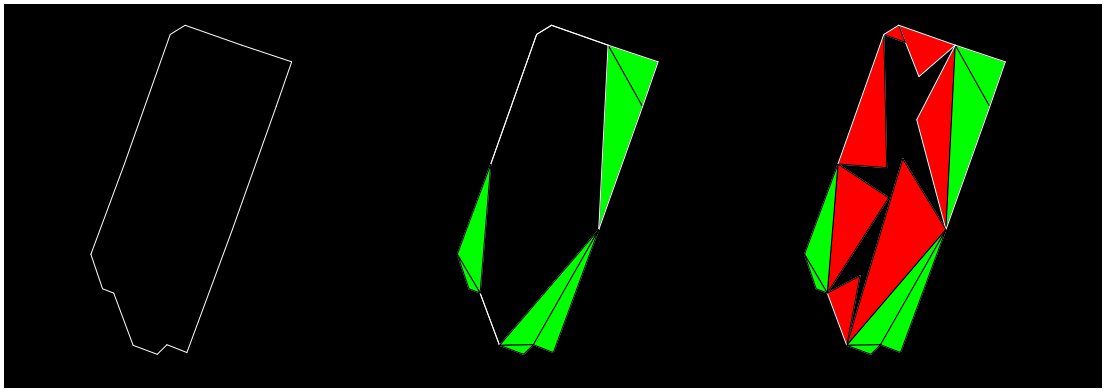
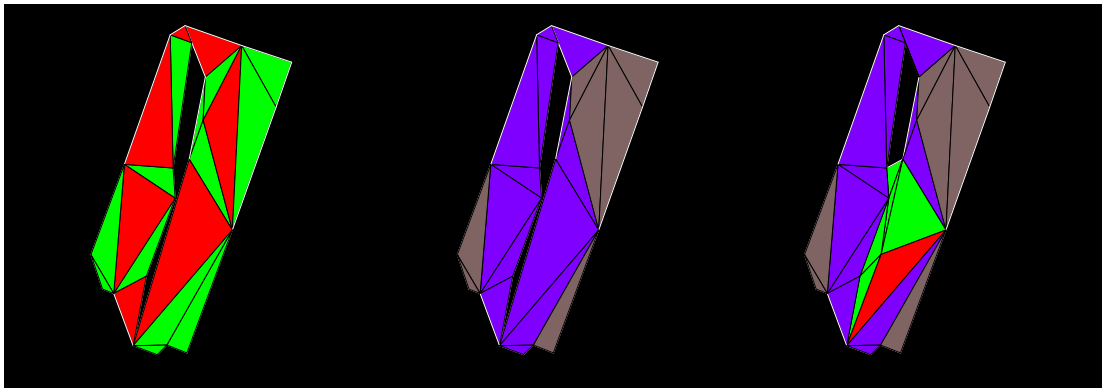


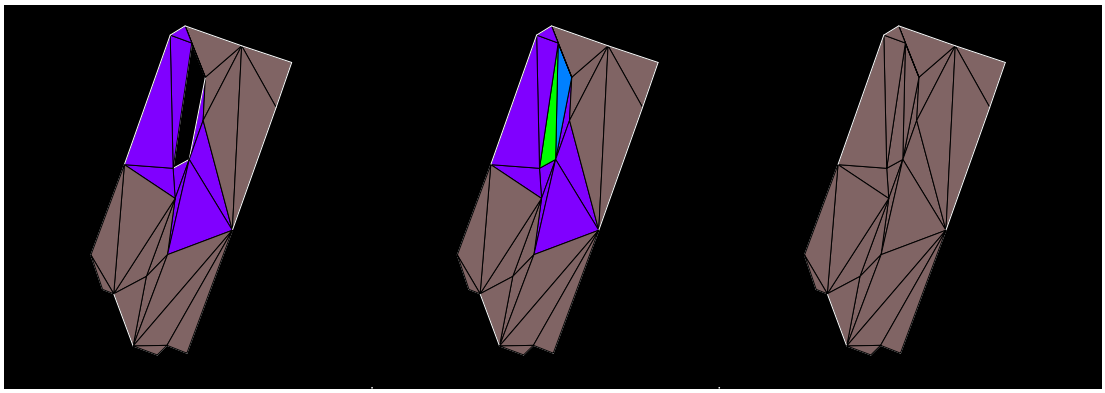
Figure 10: The Mushroom patches generated by two different patchification strategies: the upper image from Part 1 and the lower image from Part 2.



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

Figure 11: The greedy-cuts actions: (a) the initial polygon, (b) the first wave front, (c) greedy biting (d) ear clipping, (e) front faces, (f) front repair, (g) front faces, (h) ear clipping, (i) front close.

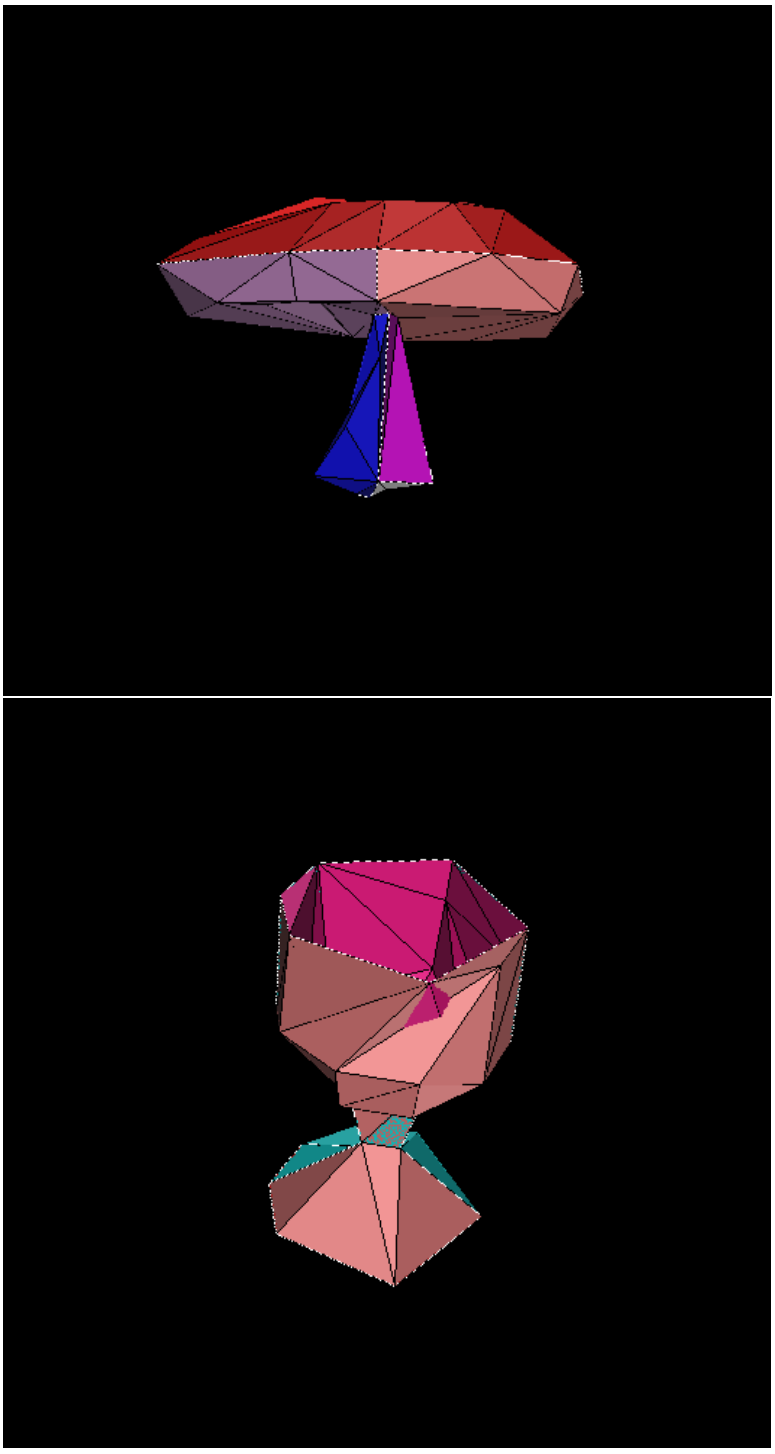


Figure 12: The simplified Mushroom image with 106 triangles and the simplified Cup image with 88 triangles.

Chapter 4

SPBM Algorithm

4.1 Motivation

We continue in this chapter our study of a problem that has received a great deal of attention over the last several years – that of simplifying a polygonal surface, in order to be able to render an approximation of it very quickly. This problem is at the heart of geometric compression and the construction of levels of details. Among the plethora of features a given technique should have, its speed and memory usage stand out since a major incentive for the development of simplification techniques is the need to generate simplified versions of truly large models, possibly composed of millions of geometric primitives. When dealing with large models, efficiency is very important, and is quite possibly the main reason the vertex-clustering technique of Rossignac and Borrel has been so popular in commercial applications, in spite of its potentially low simplification quality. Before the recent work of Garland and Heckberk, edge-collapse algorithms were seen as slow, and very memory intensive, and despite their

favorable characteristics, were impractical. With their elegant quadric-error metric, as implemented in QSlim, Garland and Heckbert eliminated several of the shortcomings of edge-collapse algorithms for the simplification of large models.

Our motivation for this work is to improve on the speed and memory usage of mesh simplification algorithms. Our main insight is the fact that any given curve defined over a surface provides useful information about the surface in a neighborhood of the curve. For many surfaces, a set of "profile curves" seems to capture most of the shape information contained in the surface, but at a much lower cost. Indeed, the importance of identifying profile curves for 3D models has been addressed for a long time in computer graphics [5] [16] [38] [48] [52] [57] [67]. Dooley and Cohen [16] described the perceptual significance of silhouette, discontinuity, and contour in clarifying geometrical structure for complex models. Ma and Interrante [48] demonstrated that a display of profile curves can be used in conjunction with surface or volume rendering to lead to a shaded rendering. Those ideas lead us to develop a new method, which uses a patchification technique for computing the profile curves. The computation of the patches is the only time that the actual surface is used in our algorithm. Subsequent phases of our algorithm only consider curves. As will be seen later, this makes it possible for our algorithm to be both very fast and memory efficient.

Our new method, the "Simplified Patch Boundary Merging" (SPBM) method, is based on clustering facets of S into "patches", simplifying the patch boundaries ("profile curves"), merging appropriate pairs of adjacent patches, and

retriangulating the resulting patch boundaries in order to obtain a new simplified surface. A key feature of our SPBM method is that it does *not* store and maintain the geometry of the surface patches themselves; rather, the method gains much of its speed and simplicity from the fact that it works on the *one*-dimensional boundary curves for patches. These profile curves capture much of the appearance of a surface when it is rendered, as they tend to define the most prominent silhouettes and visual features of the model.

The main contributions of our new SPBM method are (1) its speed, and (2) its low memory consumption, while producing simplifications with very good visual fidelity. While the SPBM method does not guarantee that every point of the approximate surface lies close to a point of the original surface, it does guarantee that the profile curves (patch boundaries) lie within a user-specified tolerance of the original surface. Since the profile curves are chosen in a manner that tends to capture the visual features of the original model, having these curves approximated in the output model leads to very good visual fidelity, particularly for the efficiency of the new method. The remarkable fact that we demonstrate is that a fairly simple and fast algorithm suffices to obtain a reasonably high degree of visual fidelity in approximating polygonal surfaces.

4.1.1 Related Work

The work of Kalvin and Taylor [39] on "SuperFaces", and Hinker and Hansen [31] on their geometric optimization paper are closely related to ours. Kalvin and Taylor [39] devised "SuperFaces", which is perhaps the prior method most closely related to our own. The main distinction between our method and the

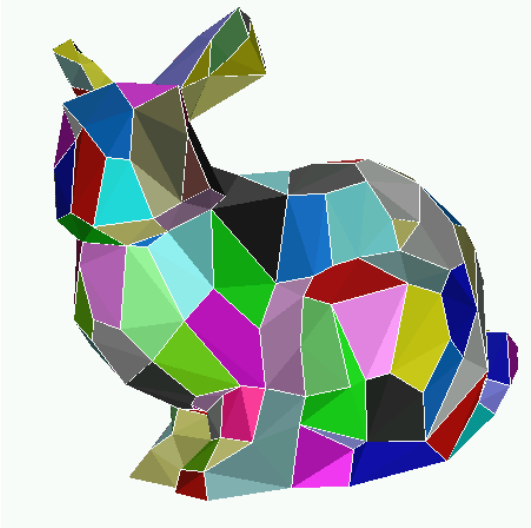


Figure 13: Low resolution approximation of the Stanford Bunny (69K faces). SPBM yields an approximation having 525 faces, based on a patchification into 185 patches. On an SGI R10K, SPBM takes 3.09 seconds (2.41 of which is reading data from disk). In comparison, QSlim 2.0 requires 8.70 seconds (5.69 of which is the simplification algorithm itself).

Superfaces algorithm is that we abandon the provable bound on the approximation error in terms of the Hausdorff distance between surfaces, replacing it with a guaranteed error bound on the profile curves that form boundaries between patches. This allows our method to be faster and more memory efficient. The region growing strategy of Hinker and Hansen [31] is different, since in ours, the region is guaranteed to remain homeomorphic to a disk. Also, our technique allows for multiple level-of-detail approximations to be generated from a single patchification process. In other related work on surface decomposition, Chazelle *et al.* [8] investigated the problem of decomposing polygonal surfaces into convex patches, showing that the optimization problem is NP-hard, while providing an experimental study of various heuristics. While their

problem is significantly different from our version of "patchification", as they require convexity, their "flood" method partially motivated our own patchification scheme.

4.2 The Algorithm

We assume that the input is given by a triangular mesh, S, which is a set of n triangles such that any one edge of a triangle is common to at most *two* triangles. We do permit an edge to be a border edge, belonging to only one triangle, but we do not consider here the non-manifold case in which an edge belongs to three or more triangles. (Our technique applies to non-manifold surfaces, but our experiments are all on manifold data, so we omit discussion here of the extension to the non-manifold case.) Of course, a vertex may belong to multiple triangles. We allow quite general, "real-world" data, in which the surface may self-intersect. If the faces of the input model are not yet triangulated, we use the robust triangulation package "FIST" ([29, 30]) to preprocess the model into a triangular mesh.

The output of our algorithm is a new triangular mesh, with vertices among the original input vertices. We do not allow Steiner points; thus, we do not attempt to preserve volume, per se, as this will not be possible at very low resolutions.

Our algorithm can be decomposed into two main phases. The first phase consists of a "patchification" algorithm, which partitions a given polygonal surface into multiple "patches" (sets of contiguous triangles), based on a user-defined parameter (controlling allowed angles between surface normals, which

in essence controls the *maximum* surface sampling), in conjunction with a set of simple heuristics designed to facilitate "well-shaped" patches, with boundaries that respect surface discontinuities. Each patch is represented only by its boundary curve (in contrast with the prior "SuperFaces" approach, which requires a full representation of the triangulation within each patch). Level-of-detail generation is then performed in three phases: (1) simplification of the polygonal curves bounding the patches, (2) selective patch merging, and (3) patch triangulation. Merging involves deleting the boundary between two patches, if their common boundary consists of a single segment and their associated normals are nearly parallel (according to the user-specified parameter). At the conclusion of the merge phase, patches are retriangulated using a prioritized version of ear clipping.

We now describe each of these steps in further detail.

4.2.1 Patchification

The first step in our algorithm is partitioning the surface model into multiple *patches*, each of which is a simply connected set of triangles. This step strongly influences later steps of the algorithm, since the "structure" of the simplified object is determined by this patchification of the original surface.

Ma and Interrante [48] stressed the benefit of averaging the normal directions of neighboring triangles for clarifying the "profile curves" in a complex model, but did not discuss how many such triangles should be included. Their approach only intended to find the "profile curves", not to generate a subdivision of a surface. In designing our patchification algorithm, we had a few goals in mind. First, and foremost, we attempted to have the patches capture the

"smooth" regions of the model, while having their boundaries respect the discontinuity curves ("profiles") that give the model its key visual features. We do not require patches to be convex, but we do wish them to be *terrains* (*i.e.*, height fields) with respect to some "patch normal" vector η (meaning that any line parallel to η should intersect a patch in a connected set, usually a single point). Second, we wanted a very simple and efficient algorithm for creating patches: any super-linear complexity in patchification would automatically make our overall algorithm too slow to be used in practice.

With these goals in mind, we designed a very simple "growth" procedure for creating (and defining) our patches. Our procedure bears some resemblance to the "flooding" heuristic of Chazelle *et al.* [8]. Basically, starting from a random face (triangle) of S, we start to grow a patch, P, and grow it one face at a time using a constrained breath-first search (BFS).

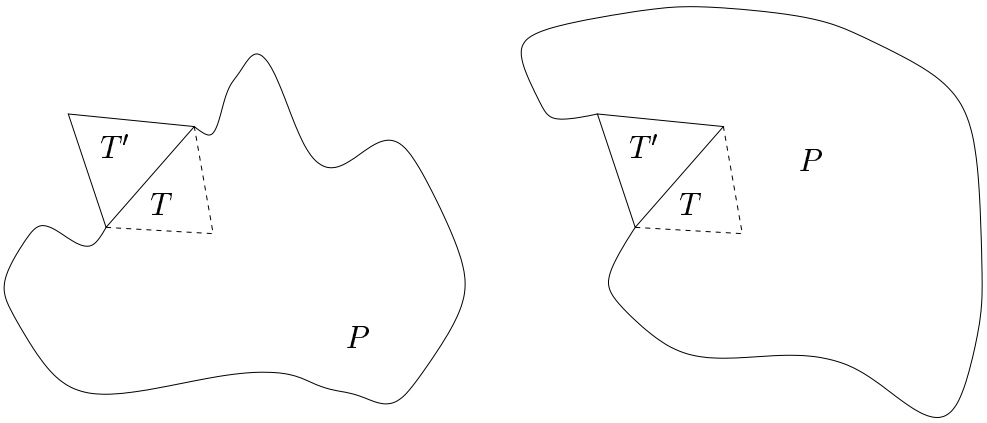


Figure 14: Triangle T has just one free neighbor, T'. Left: T' shares only one edge with the patch P; Right: T' shares two edges with P.

4.2.2 Patch "Growth" Algorithm

The triangles that are contained in the patch P are stored in a *queue*, Q. Each patch P has an associated *patch normal* vector, η_P. Each triangle of S is marked too with the patch (if any) to which it belongs. A triangle that does not yet belong to any patch is said to be *free*; a vertex or a triangle that already belongs to some patch is said to be *marked*. An invariant of the patch growing procedure is that we maintain the *simplicity* of the boundary of P; i.e., the patch is a simply connected subset of the surface S, homeomorphic to a 2-disk, with no "pinch off" points or "holes".

Let $T \in Q$ be the triangle of the patch P whose neighbors are now being expanded in the BFS. There are four cases, depending on how many neighbors of T are free:

0 free neighbors Then any edge of T that corresponds to a neighbor in a patch distinct from P is marked as a patch boundary edge; no neighbors of T are added to P.

1 free neighbor Let T' be the free neighbor. If Feasible (T, T', P) (see below for definition), then T' is added to the patch P (T' is enqueued in Q). Refer to Figure 14.

2 free neighbors Let T' and T'' denote the free neighbors. If Feasible (T, T', P) *and* Feasible (T, T'', P), then T' and T'' are both added to the patch P (enqueued in Q); otherwise, neither are added to the patch P, and T is *removed* from P. Refer to Figure 15. (The rationale for removing T in this case is to make the patch "fat", in the sense of not having long and skinny tentacles.)

3 free neighbors This can be the case only when T is the first triangle of the first patch P. In this case, we add to P those neighbors T' for which Feasible (T, T', P) is true.

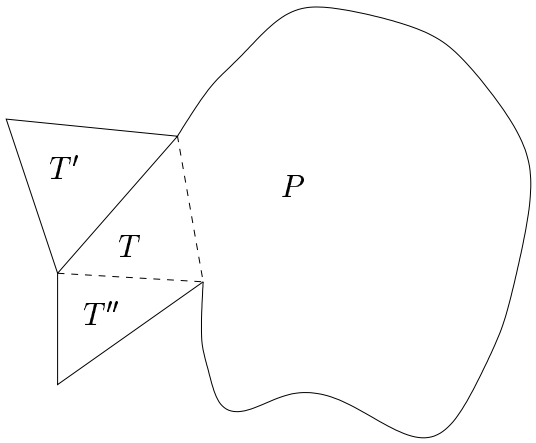


Figure 15: Triangle T has two free neighbors, T' and T''. Potentially, T is removed from the patch P.

With each triangle that we add to P, we update the patch normal, η_P, which maintains the average normal among the triangles of P (optionally weighted by the area of the triangles).

When one patch P is done being extended (no more adjacent free neighbors can be added, subject to our feasibility constraints), then the algorithm selects a triangle that is neighboring to P and uses it as the "source" to start growing a new (neighboring) patch. This process continues until all triangles are marked, indicating that the surface has been completely covered by non-overlapping patches.

Two neighboring patches, P and P', share part of their boundaries; the connected components of the common boundary are polygonal chains, which we refer to simply as *chains*. Each chain is stored in an *edgelist*; it links two vertices (the first and last along the chain), which are called *chain endpoints* (or *endpoints*, for short). We define the *patch graph* to be the dual graph of the decomposition induced on S by the patches; specifically, the patch graph has nodes corresponding to patches and edges corresponding to chains that form the common boundary components between neighboring patches.

4.2.3 Triangle Feasibility Criteria

We now specify precisely what we mean by a free triangle T', neighboring $T \in P$, being *feasible*: *Feasible* (T, T', P) performs the following steps:

(a) If all three vertices of T' are already marked as members of P, and if the other two neighbors of T' are both *unmarked* or marked as members of patches other than P, then we return, with "INFEASIBLE". (Adding T' to the patch would cause the boundary of P to "pinch off", violating the simplicity of P.) See Figure 16.

(b) We test if the normal, $\eta_{T'}$, to T' makes a "small" angle with both the normal η_T and the patch normal η_P. Specifically, if $\eta_{T'} \cdot \eta_P \geq \alpha_1$ *and* $\eta_{T'} \cdot \eta_T \geq \alpha_2$, then we return with "FEASIBLE".

Here, α_1 and α_2 are user-specified parameters that control the extent to which triangles of a patch are nearly coplanar. Actually, we allow up to four parameters, as α_1 and α_2 are allowed to have different values in each of two cases: If T' shares *two* of its edges with triangles in P (triangle T,

plus another triangle $T'' \in P$), then α_1 and α_2 are taken to be smaller (by default, taken to be 0, corresponding to an angle of 90 degrees); if T' shares just one edge with P (namely, the edge in common with T), then α_1 and α_2 are typically assigned larger values, to reflect the requirement that the angle with $\eta_{T'}$ should be smaller in this case. (By default, we use values corresponding to angles of roughly 25 degrees and 20 degrees, respectively.)

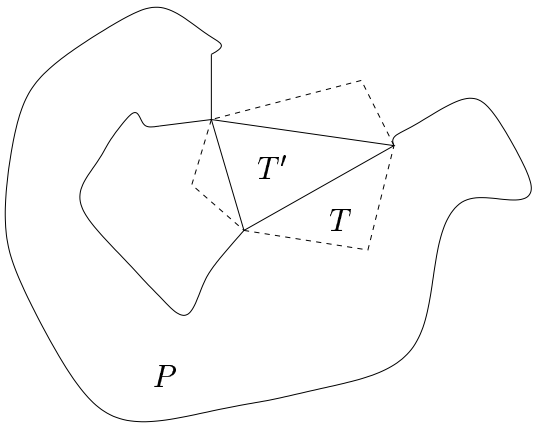
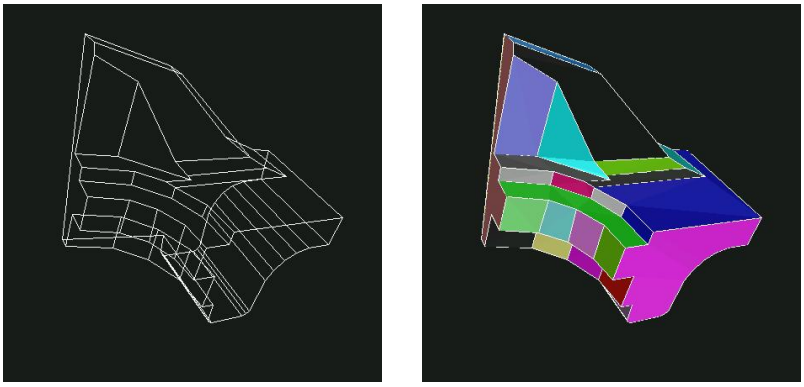


Figure 16: The neighbor, T', of $T \in P$ is not added to the patch P, since it would cause P to fail to be simple, creating a "pinch".

Examples. The result of our patchification on a finely tessellated sphere is a set of nearly circular patches having approximately the same size. The size of the patches goes down as the parameters α_1 and α_2 increase (i.e., as the angular tolerance goes down), and the patches are the original triangles themselves if the angular tolerance is zero. Similarly, a cylinder is patchified



(a) (b)

Figure 17: Depicted are (a) the simplified boundary curves, and (b) the final patches after merging.

into a set of bands along the length of the cylinder. In Figure 13 we illustrate the result of patchification on the Bunny model, by coloring patches distinctly. See also Figs. 17b, 20a, 21a, 22a, and 24a.

4.2.4 Level-of-Detail Generation

While the patchification process has, in fact, already resulted in a simplification of the surface (since vertices interior to patches have been dropped), we often think of it as a form of preprocessing, before doing a more refined level-of-detail generation, as is done in the next phase of the SPBM algorithm. In fact, for a fixed patchification, we can generate multiple levels of detail based on varying the parameters that control the next two steps of the algorithm: boundary simplification and patch merging.

4.2.5 Boundary Simplification

Once the patches have been computed, we move to the second phase of our simplification algorithm, which simplifies the chains that form the patch boundaries. Each chain is represented by a list $(v_1, v_2, \ldots, v_k)$ of vertices in 3-space. For a given user-specified error tolerance $\epsilon > 0$, we use a simple divide-and-conquer refinement technique to simplify the subchain linking v_i to v_j $(1 \leq i < j \leq k)$. (Initially, $i = 1$, $j = k$.) If the subchain lies within ϵ of the line segment $\overline{v_i v_j}$, then we are done – the subchain is approximated by the single segment. Otherwise, we include vertex v_m, with $m = (i + j)/2$, in the approximation and recursively simplify the subchains from v_i to v_m and from v_m to v_j. We have experimented with other methods of approximating chains, but we have found that this simple method works well in practice (see Fig. 17a); it also readily permits a hierarchy of chains to be computed, at multiple levels of detail. This previous feature of chain simplifications allows our SPBM algorithm to be generalized from the computation of a single simplified surface, the case on which we concentrate above, to a multi-resolution hierarchy. While our chain approximation technique is not guaranteed to produce a minimum-vertex ϵ-approximation, it can be shown to be within a small factor of optimum in the worst case. For recent theoretical results from computational geometry on simplifying chains in three dimensions, see [6]. Considering the applification of progressive generation and view-dependent approach as our future works, the ability to store the hierarchy approximated boundary cures is very important for our future work on an amplification of the SPBM algorithm implementing progressive generation and view-dependent simplification. That multi-resolution hierarchy can be stored in merely one run of

simplification from the initial mesh to the base (coarsest) mesh. With this requirement, we reverse the boundary simplification procedure from bottom to top. We recursively considers pairs of vertices v_i and v_j and the segment (v_i, v_m), where $m = \frac{i+j}{2}$, is included in our simplified curve if and only if it is within ϵ-tolerance but the segment (v_i, v_j) is **not** within the tolerance. While a base mesh is generated, all of the boundary curves are straightened. This bottom-up approach has recovered the hierarchy of approximated boundary curves.

4.2.6 Patch Merging

In the patch merging phase, we attempt to merge pairs of adjacent patches, P and P' in order to further decrease the complexity of the surface. This is a recursive process that begins with the original patches and *simplified* chains and through the merging of patches and collapsing of chains, generates a successor of larger and larger patches until no further merging is possible within the parameters that specify the level of approximation (see Fig. 17b).

Specifically, we merge P and P', if they satisfy both of the following conditions:

(1) P and P' each have all of their boundary chains being singleton edges, and their common boundary consists of a single (one-edge) boundary chain, and

(2) $\eta_P \cdot \eta_{P'} \geq \alpha_1$; so that the patch normals are "nearly" parallel, according to the parameter α_1 specifying the angular tolerance.

When we merge P and P', we delete the edge (u, v) that is their common boundary. If this results in the endpoint u (resp. v) having degree two (being the endpoint of exactly two other one-edge chains) then we delete u (resp. v), and replace the two corresponding one-edge chains with a single one-edge chain. We also update the patch normals when we merge P and P', computing the new (possibly weighted) patch normal for the combined patch.

Every time that a patch P is considered for merging, we attempt to further "straighten" its boundary by collapsing one or more of its links. We only collapse a chain (u, v), bordering patches P and P', when the following conditions are met:

(a) No other patch P'' has both u and v on its boundary (see Fig. 18),

(b) $\eta_P \cdot \eta_{P'} \geq \alpha_1$, and

(c) The length of (u, v) is a constant factor smaller than the length of the longest chain. (Good results are obtained by using a factor of $\frac{1}{5}$.)

If these conditions are met, then we perform a chain-collapse, replacing (u, v) with the endpoint (u or v) that corresponds to the larger variation in surface normals.

4.2.7 Boundary Preservation

The boundary of a mesh is given by the set of all edges belonging to exactly one triangle. It is essential to preserve this boundary as much as possible. A careless collapse action could seriously destroy the boundary as illustrated in Fig. 19b. Special care has been taken with those chains linked to the boundary.

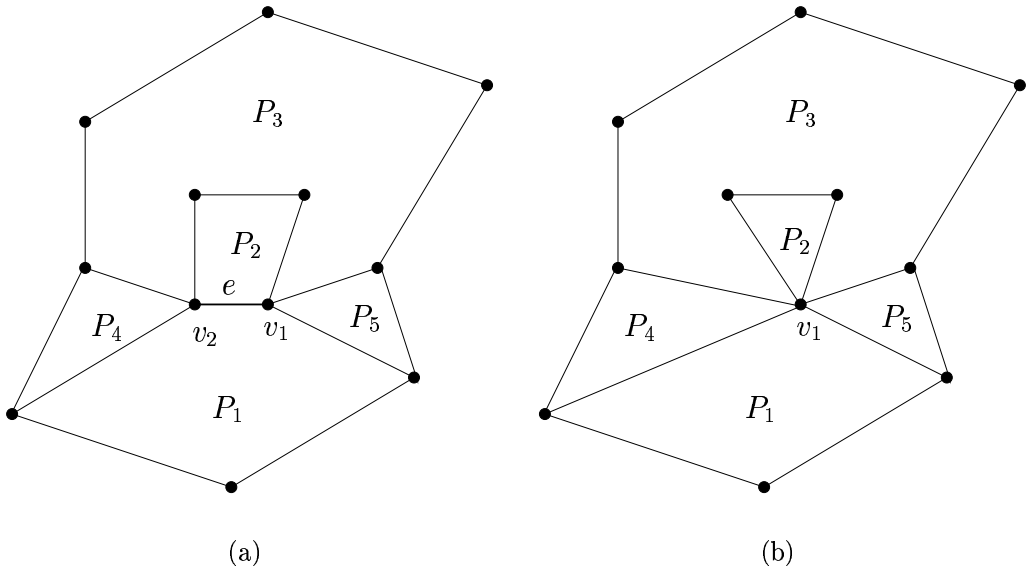


Figure 18: Depicted are (a) a patch P_3 with both vertices v_1 and v_2 on its boundary, and (b) a patch P_2 anchored at v_1 after collapsing the chain E to v_1.

Those chains are only allowed to collapse to the vertex **not** on the boundary, however we allow boundary chains to collapse into either side.

4.2.8 Patch Triangulation

At the conclusion of patch merging, we are ready to retriangulate the simplified patches that remain. This is done with a simple "ear clipping" procedure; we use a somewhat simplified version of the more sophisticated ear clipping of FIST [29, 30]; see also O'Rourke [51].

It is important to realize that, even though the patches are generated in

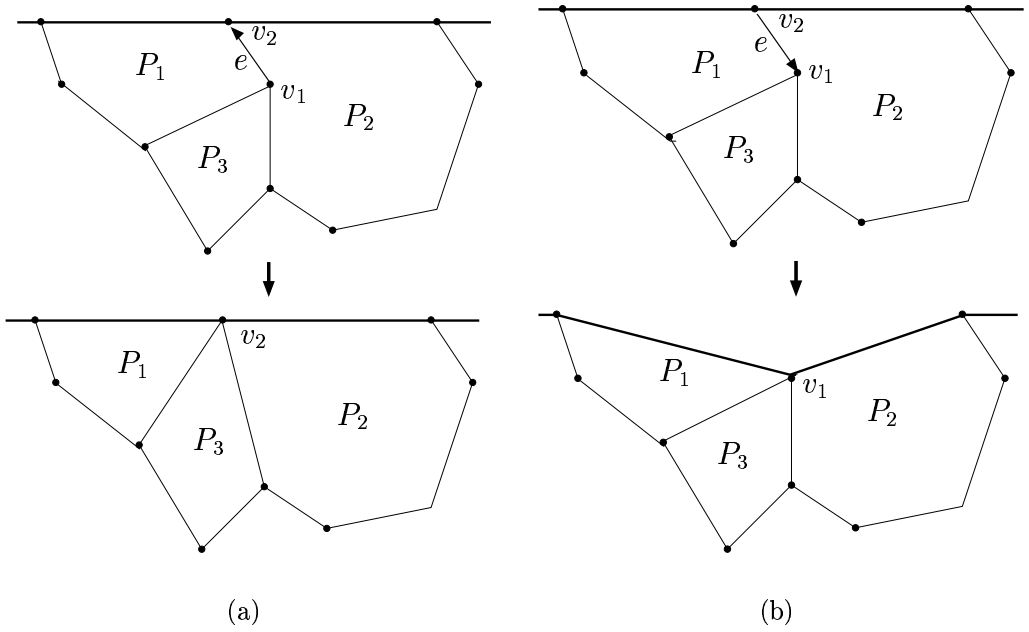


Figure 19: Depicted are (a) a chain collapsed to a boundary vertex, and (b) how the boundary is destroyed when the collapse goes the other way.

such a manner that the normal vectors fall within some small cone, it is possible that the boundary of a patch P does not project as a simple polygon onto the plane that is orthogonal to η_P. (Consider, for example, a "spiraling ramp," as one may see in a parking garage.) While these cases may be very rare in practice (indeed, in our experiments we had to generate such cases specially, for testing purposes), our goal is to handle all cases robustly, and not to restrict the type of input surfaces we can handle. One approach we have used to this end is to apply the FIST triangulation system directly to triangulate all patches; since FIST is guaranteed to produce a triangulation of

any closed polygonal chain (even if its projection is not a simple polygon, and even if it has arbitrarily bad artifacts), this approach guarantees that our system never fails to produce *some* triangulation that is at least combinatorially a valid triangulation. (Recall that the problem of determining if an unknotted polygonal loop in 3-space can be triangulated (without Steiner points) to yield a surface homeomorphic to a 2-disk is NP-complete [7]. Thus, FIST does not claim to "solve" this hard problem, but instead has a set of highly effective heuristics designed to try to avoid triangulations that self-intersect.) However, the triangulations produced by FIST for "spiraling ramps" and similar situations in which the projection of the patch boundary self-intersects were found not always to be the best for purposes of visual fidelity (often producing triangles whose normals were not close to the patch normal). Thus, we have opted to provide three options to the user: (1) to use FIST; (2) to use our simplified version of the FIST heuristics (which are guaranteed to give valid triangulations as long as the projected patch boundaries do not self-intersect); and (3) a modification of our patch growth and merging methods, in which we explicitly forbid a patch from having a self-intersecting boundary projection. (Method (3) currently uses a relatively naive method of testing, checking for each change in a patch boundary if the resulting segments in the projection create a self-intersection. Realizing that this is inefficient, in general, we are implementing a fast and simple grid-based hashing technique to prune most of the segment intersection tests.) The experiments reported here are based on the default method, method (2), which we have found to work very well in practice, while being able to handle all of the large test suite of models in our experiments.

Using η_P, we can readily define what it means for an ear (corresponding to three consecutive vertices (v_{i-1}, v_i, v_{i+1}) about the boundary of P) to be *feasible*: the corresponding segment $v_{i-1}v_{i+1}$ must project to a valid diagonal of the simple polygon obtained when projecting the boundary of P onto the plane orthogonal to η_P. This is readily checked with a few simple cross products ([51]).

In order to obtain good quality triangulations, i.e. with good aspect ratios, we have found it advantageous to prioritize the ear clipping: we clip ears according to increasing angle (interior, convex angle).

We remark that there can arise a case in which a triple (v_{i-1}, v_i, v_{i+1}) defines an ear for *both* patches, on each side of the chain that contains the triple. In this case, we do not want to include both triangles that would result from clipping (they would have oppositely oriented normals and cause a visual artifact); instead, we choose to keep that triangle whose normal vector is most nearly parallel to the corresponding patch normal.

4.3 Experimental Results

Our algorithms have been fully implemented and tested. One of the strengths of our method is the simplicity of the algorithms, as we have described them. All the steps are very simple and intuitive, and the implementation follows their description very closely. Our implementation is in "C" and is readily portable, running on PCs and workstations. We report our results based on an R10000 SGI Indigo2 workstation (195 MHz CPU, 256 MB RAM, 32 KB data/instruction cache, and 1 MB secondary cache, IRIX 6.5). Compilation

was with "cc -O2". (For comparison, on a Pentium Pro PC (180 MHz, 128MB) the running times go up by 33% on average.)

In order to evaluate the simplification rate and the quality of the levels of the reductions, we ran a battery of tests on several public domain datasets. For these tests, we compare our prototype SPBM implementation, with QSlim 1.0 and QSlim 2.0. (Originally, all our comparisons were performed with respect to QSlim 1.0, but on March 17th, 1999, Michael Garland released QSlim 2.0, a new and improved version of his state-of-the-art simplification code.) For each of our five datasets, we computed three different level-of-detail approximations, each of these were computed with all three codes. For the larger dataset consisting of over one million triangles – the Buddha – we were unable to run QSlim 1.0, since our test machine ran out of memory.

In Table 1, we summarize the running times obtained with each code, for each one of the level-of-detail generations. In Figs. 20 to 24, we show side-by-side comparisons of the approximations generated with SPBM and QSlim 2.0.

Here is a summary of our comparisons for each dataset.

(1) The Sphere dataset is a very small dataset, consisting of 1026 vertices and 2048 triangles. This dataset is included in our experiments primarily for visual quality assessment of the approximations. Both techniques seem to provide comparable approximations (see Fig. 20).

(2) The Femur dataset consists of 76,794 vertices and 153,322 triangles. This mesh was extracted from volumetric data [43]. SPBM is almost five times faster than QSlim 1.0; and three times faster than QSlim 2.0,

CPU Time Comparison Table												
	Femur (153,322 tris)			**Bunny (69,451 tris)**			**Fandisk (12,946 tris)**			**Buddha (1,087,716 tris)**		
f	997	545	270	1504	1000	525	460	266	150	12,700	7400	3500
SetUp	7.223	7.224	7.225	3.487	3.485	3.482	0.550	0.548	0.549	–	–	–
Init	3.881	3.879	3.880	2.178	2.177	2.178	0.290	0.289	0.289	–	–	–
QSlim-1.0	19.56	19.64	19.67	8.58	8.62	8.64	2.43	2.44	2.45	–	–	–
T_{tot}	30.66	30.74	30.77	14.24	14.28	14.30	3.27	3.28	3.29	–	–	–
SetUp	3.00	3.00	3.00	1.33	1.34	1.33	0.225	0.225	0.225	21.43	21.75	21.65
Init	2.92	2.92	2.92	1.68	1.67	1.68	0.22	0.22	0.22	22.78	22.80	22.82
QSlim-2.0	13.96	14.00	14.00	5.69	5.70	5.69	1.19	1.19	1.19	130.0	123.7	126.0
T_{tot}	19.88	19.92	19.92	8.70	8.71	8.70	1.635	1.635	1.635	174.21	168.25	170.47
SetUp	5.18	5.17	5.24	2.40	2.41	2.41	0.44	0.44	0.44	43.96	43.83	43.80
Patch	1.14	1.15	1.14	0.54	0.53	0.53	0.08	0.08	0.08	9.84	9.82	10.3
SPBM	0.26	0.18	0.17	0.45	0.26	0.15	0.14	0.08	0.05	6.18	4.83	6.60
T_{tot}	6.58	6.50	6.55	3.39	3.20	3.09	0.66	0.60	0.57	59.98	58.48	60.7

Table 1: Simplification results of running SPBM, QSlim 1.0, and QSlim 2.0 on four different datasets. All times are in seconds. Here, f is the number of faces in the simplified model. T_{tot} is the total time (in seconds) required to simplify the models. For SPBM, T_{tot} is divided into set-up time (loading the dataset from disk), Patchification ("Patch"), and level-of-detail generation time ("SPBM").

for computing level-of-detail approximations of this dataset. The visual quality of the approximations can be seen in Fig. 21. For all we could see, both SBPM and QSlim 2.0 seem to produce comparable "visual-quality" approximations of the Femur.

(3) The Stanford Bunny dataset consists of 34,834 vertices and 69,451 triangles. This mesh was generated by merging several range scans at Stanford University [69]. SPBM is over four times faster than QSlim 1.0; and two and a half times faster than QSlim 2.0. The visual quality of the approximations can be seen in Fig. 22. Although it is difficult to see from the picture, in close examination, we found out that SBPM seems to generate better approximations than QSlim 2.0, especially around the neck, ears, and back areas.

(4) The Fandisk dataset consists of 6,475 vertices and 12,946 triangles. This is a CAD dataset. SPBM is about five times faster than QSlim 1.0; and two and a half times faster than QSlim 2.0. The visual quality of the approximations can be seen in Fig. 23. Even though from this angle it seems that both SPBM and QSlim 2.0 produce comparable approximations, under closer inspection, we found that QSlim (both versions) produce a triangulation artifact (this is shown in Fig. 23 (g) and (h)).

(5) The Stanford Buddha dataset consists of 543644 vertices and 1,087,716 triangles. This is the largest and most complex dataset we used in our experiments. This mesh was generated by merging a large number of range scans consisting of millions of points at Stanford University [14]. We were not able to run QSlim 1.0 on this dataset due to the limited

amount of memory on our test machine. SPBM is almost five times faster than QSlim 2.0. Several different resolution models are shown in Fig. 24. For the Buddha, after extensive examination, we believe QSlim 2.0 has better visual quality than SPBM.

In summary, SBPM was able to generate high-quality approximations of all of these models. There are three main classes of data represented: meshes from an automatic range scanner (Bunny, Buddha), meshes from a commercial CAD system (Fandisk), and meshes extracted from volumetric datasets (Femur); and include meshes of up to one million of triangles.

With respect to speed, our current implementation is about five times faster than QSlim 1.0, and 2.5–3 times faster than QSlim 2.0. We believe that for the Buddha dataset, the reason SPBM was *five* times faster than QSlim 2.0 (instead of the average 2.5–3 time), is due to QSlim 2.0 using more than 256 MB of memory.

We estimate that our memory usage is close to 100 bytes per vertex in comparison to the 270 bytes per vertex of QSlim 2.0 [22]. Our method has a very small overhead for extra storage since, other than the face-adjacency information contained in the input model, the only data structure utilized by the algorithm is the edge list that stores the profile curves (patch boundary chains) and their simplifications. Furthermore, most of the memory requirements are only needed during the patchification process. For actual level-of-detail generation, the memory requirements are very low.

4.4 Conclusion

We have proposed a simple, fast, and memory-efficient surface simplification technique. It works by first partitioning a given polygonal surface into multiple patches. Simplification is then performed in two phases: simplification of the polygonal curves bounding the patches followed by patch merging. Finally, the resulting simplified patch boundaries are triangulated to yield a final approximate surface model.

Our method has been implemented in a system, SPBM, which has been extensively tested. In comparisons with a leading method, implemented in the newly released QSlim 2.0, we have seen an average speedup of a factor of two and a half times, while using less memory and producing good visual fidelity. In some instances (particularly CAD models), the approximations we obtain have better visual fidelity; in other instances, our approximations have less visual fidelity than QSlim.

Some final remarks:

- The patches generated by our patchification process respect the sharp curves of the original models. See the example of the multi-colored patches for the Fandisk in the color plates.

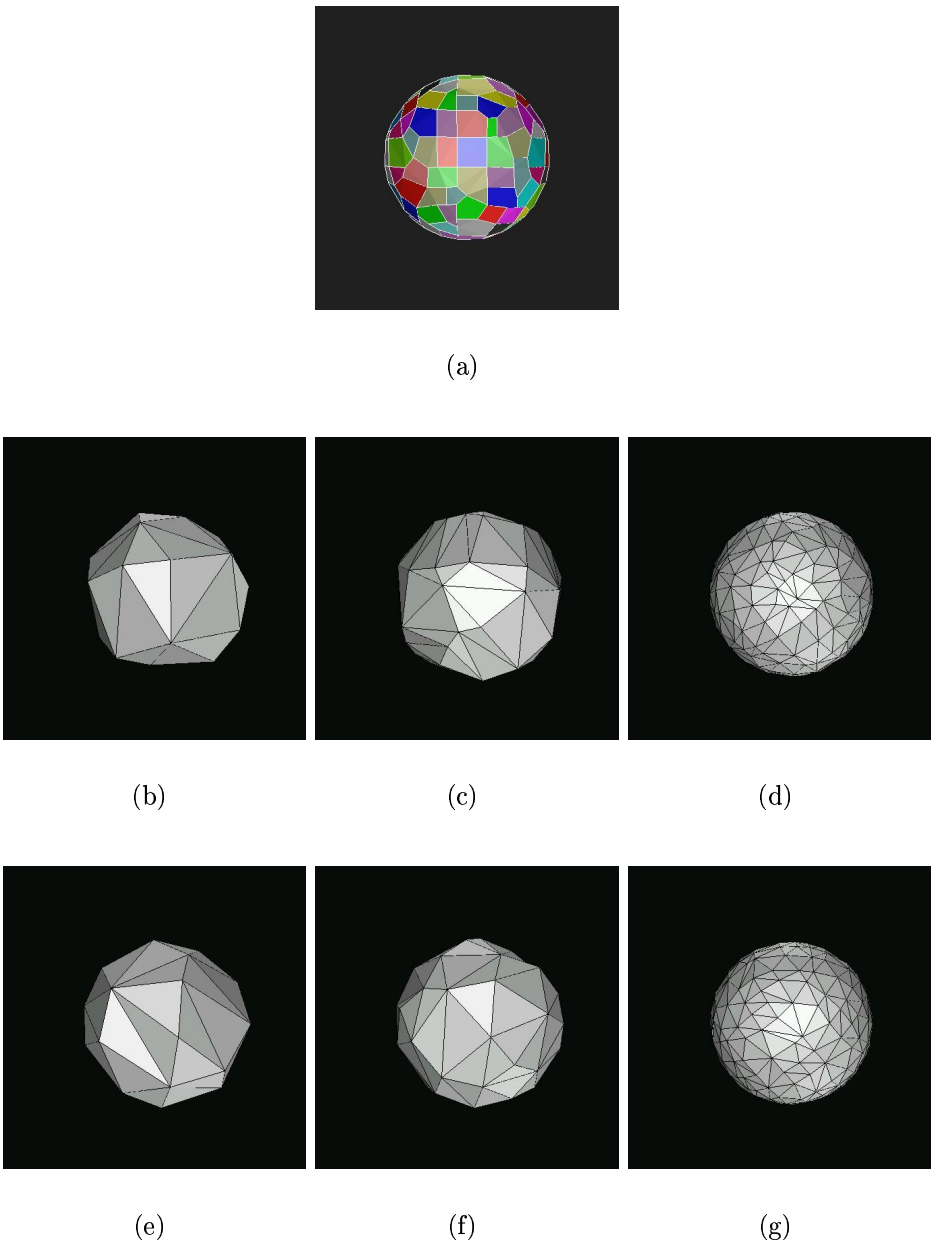
- Our merging procedures are steady and smooth. In particular, the merging process of the three levels of detail of femur model are very smooth, even at the extremely high simplification ratio of 0.1%.

- Our triangulations tend to be very good, composed of well-shaped triangles. This is achieved without performing costly "edge swap" operations,

as many other approaches require.

- Our technique is very memory-efficient, since it only operates on a set of simplified patch boundary curves, deferring the retriangulation step to the very end.

We expect to make the code publicly available, as it will serve as a tool for other graphics researchers. This will also permit more extensive testing and comparison with other methods under development.

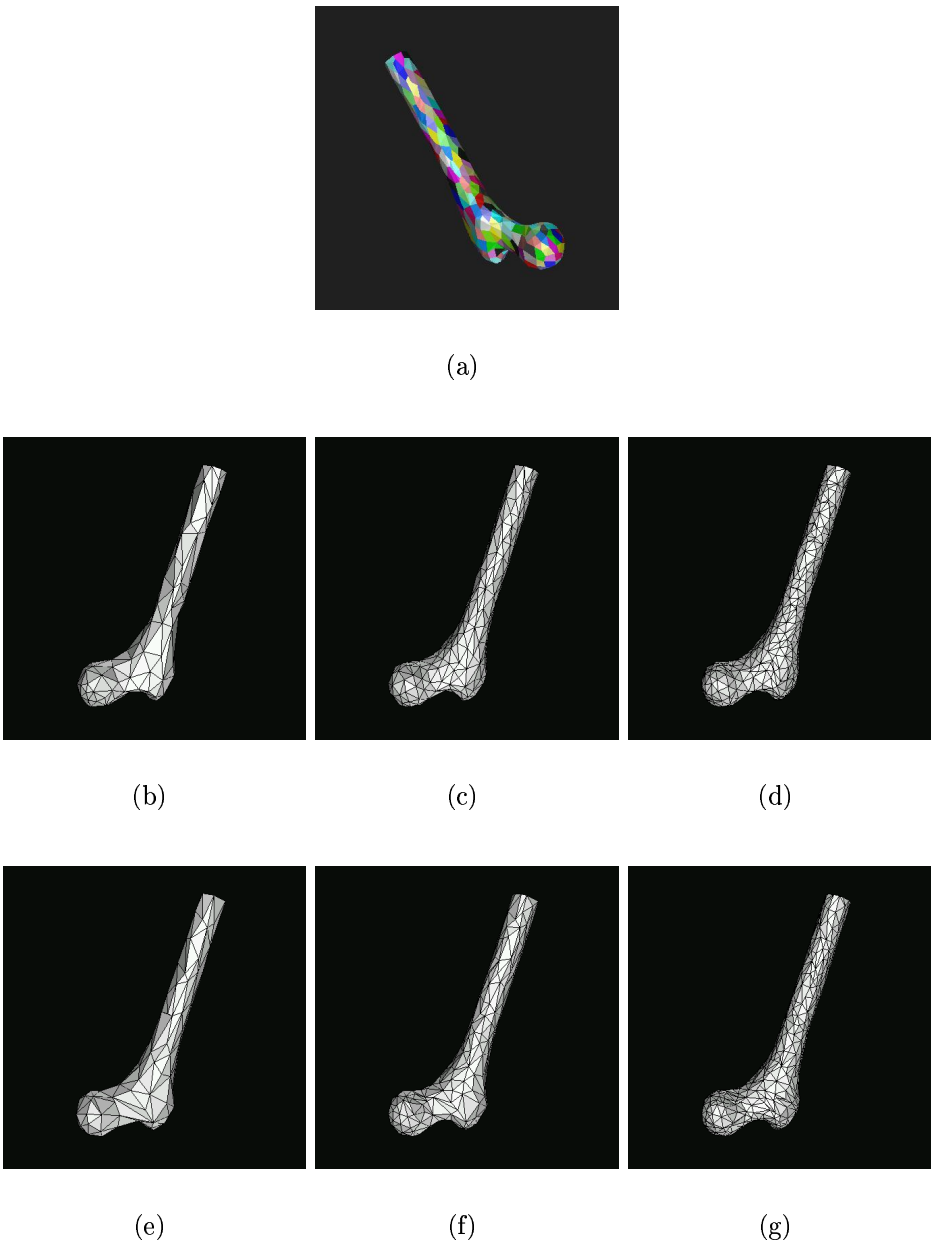


(a)

(b) (c) (d)

(e) (f) (g)

Figure 20: Sphere dataset. (a) shows a typical patchification of the sphere. (b)–(d) shows the 52-, 100-, and 500-face approximation of the sphere with SPBM. (e)–(g) shows the results obtained with QSlim 2.0.

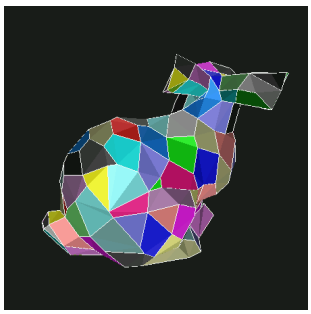


(a)

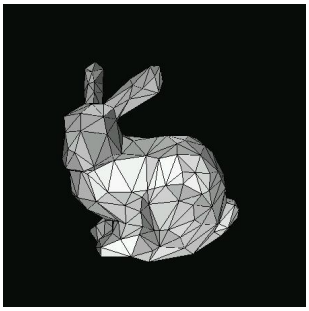
(b) (c) (d)

(e) (f) (g)

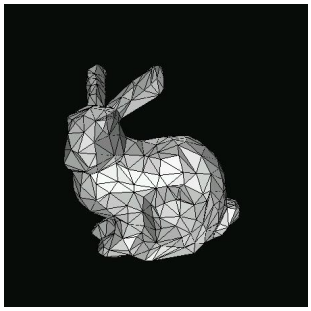
Figure 21: Femur dataset. (a) shows a typical patchification of the femur. (b)–(d) shows the 270-, 545-, and 997-face approximation of the femur with SPBM. (e)–(g) shows the results obtained with QSlim 2.0.



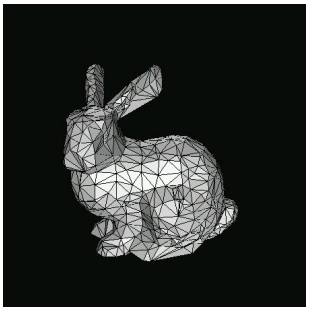
(a)



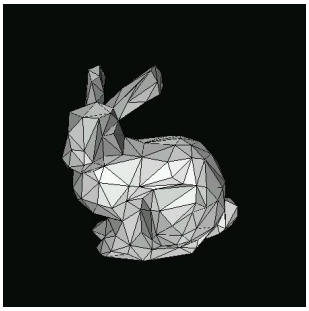
(b)



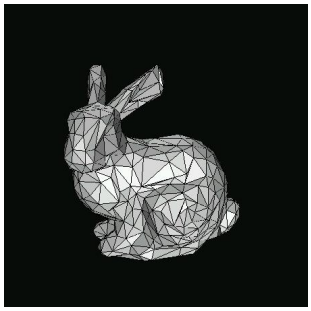
(c)



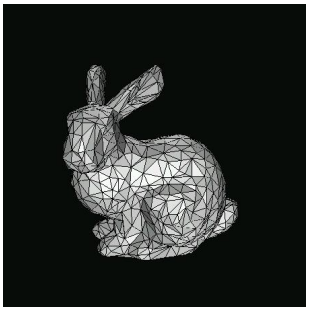
(d)



(e)

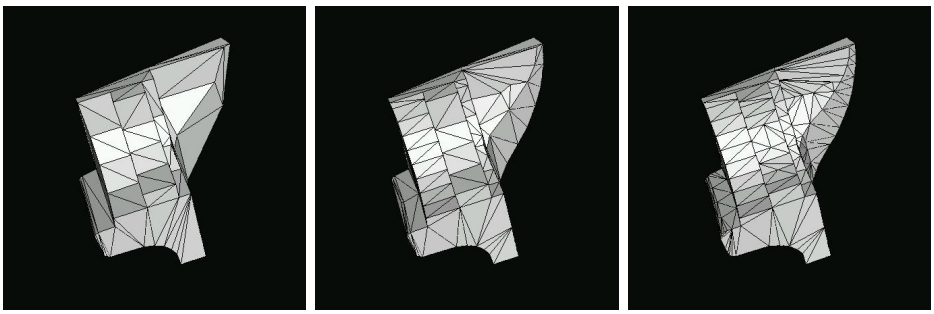


(f)

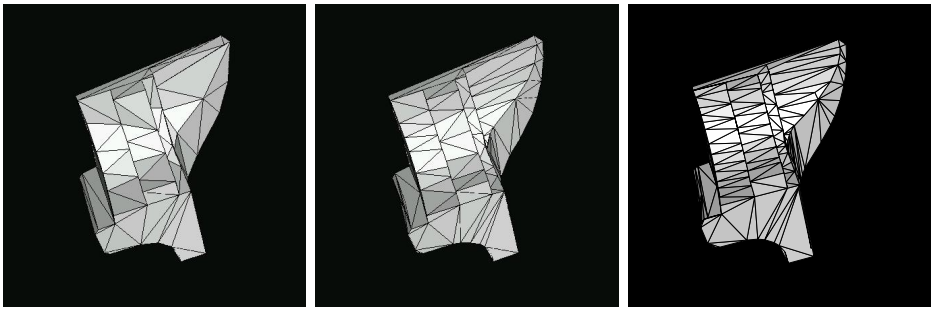


(g)

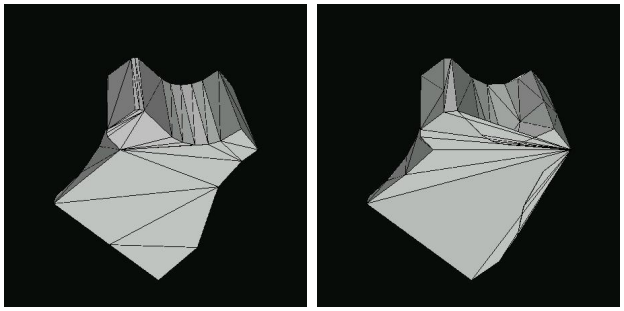
Figure 22: Bunny dataset. (a) shows a typical patchification of the bunny. This is similar to Fig. 13, but looking from the back. (b)–(d) shows the 525-, 1000-, and 1504-face approximation of the bunny with SPBM. (e)–(g) shows the results obtained with QSlim 2.0.



(a) (b) (c)

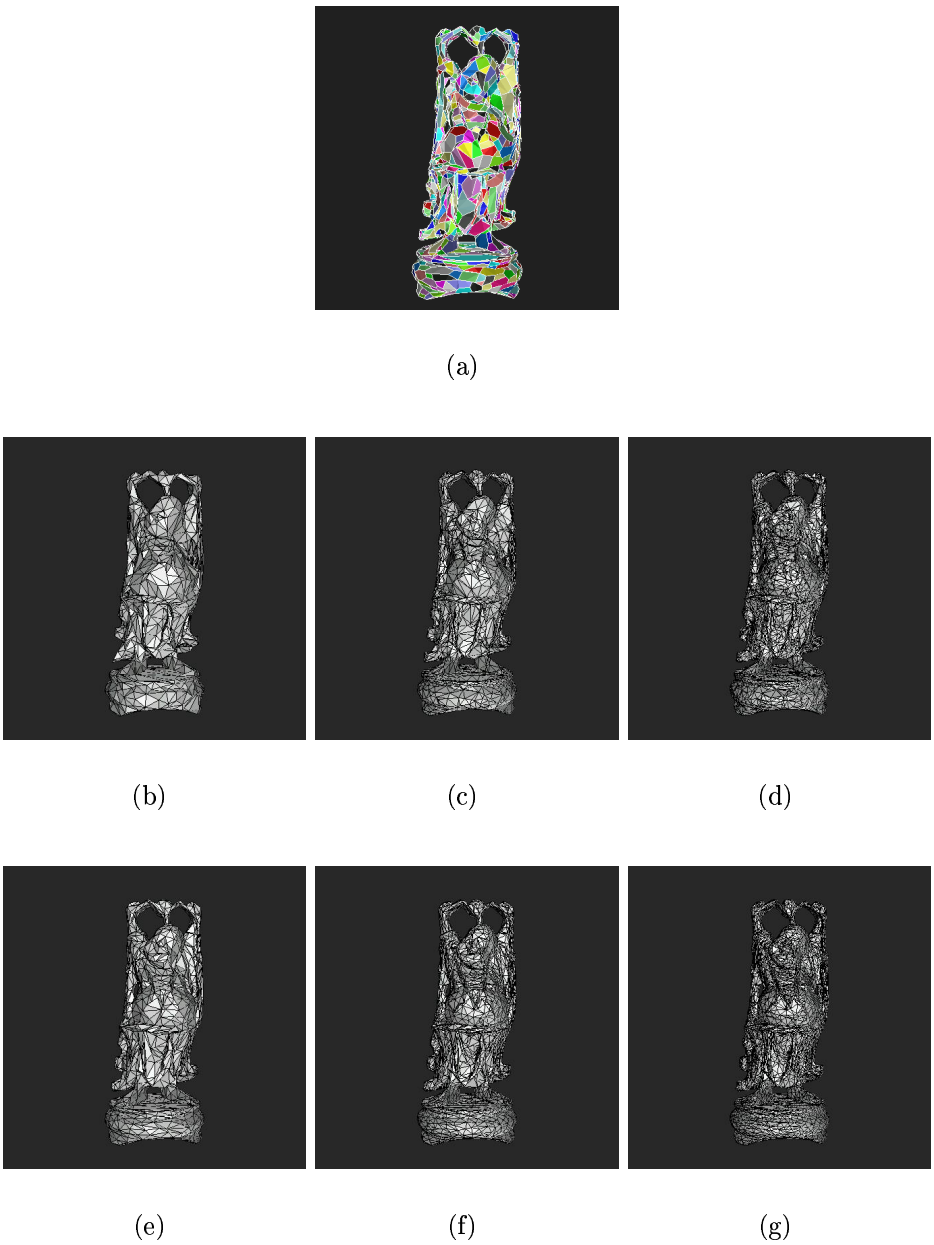


(d) (e) (f)



(g) (h)

Figure 23: Fandisk dataset. (a)–(c) shows the 150-, 266-, and 460-face approximation of the fandisk with SPBM. (d)–(f) shows the results obtained with QSlim 2.0. (g) and (h) show the other side of the fandisk, (g) was computed with SPBM, and (h) was computed with QSlim 2.0. Note that (h) has a severe artifact.



(a)

(b) (c) (d)

(e) (f) (g)

Figure 24: Buddha dataset. (a) shows a typical patchification of the buddha. (b)–(d) shows the 35000-, 7400-, and 12700-face approximation of the buddha with SPBM. (e)–(g) shows the results obtained with QSlim 2.0.

Chapter 5

Topology Simplification Algorithm

5.1 Motivation

Most methods for computing simplifications of models are designed to preserve the topology of the input, keeping holes and tunnels that appear in the full-resolution model. While preservation of genus is important in some applications (e.g., molecular modeling), it is not required in many applications that demand only a visually close approximation to the original model. Indeed, preservation of topology may be highly undesirable in cases involving virtual flythroughs of CAD models and structures that have a large number of small holes or tunnels. Demanding that the simplified models in the hierarchy preserve topology in these cases can lead to drastically more complex polygonal approximations than would be required to maintain adequate visual fidelity, especially in models having numerous small holes. (Consider, for example, a

simple "plate" having a large grid of small holes; see the "Fixture" model in the color plates.) In short, topology preservation may prevent simplification.

We address the problem of computing simplifications of polygonal models without constraints on the preservation of the initial topology. Some of the simplification algorithms based on edge-collapse [24] and [59] have the capability of collapsing a hole, resulting in a single point; however, this single point is hard to remove in the following collapses since the accumulated error imposed after collapsing a hole is very high. These shortcomings are demonstrated in Figure 25 and Figure 26 created by QSlim, a state-of-the-art, very elegant edge-collapse based simplification tool. Figure 25(b) shows the ten points after collapsing ten small holes in the Disk model. The boundary has been collapsed, while these ten coplanar points have not been removed, even further collapses are taken and the shape has been drastically destroyed as showed in Figure 25(c). Similar results are presented from the Fixture model, Figure 26, and the Battery model with small bumps on the top, Figure 30(b) and (c). This led us to explore the extension of our SPBM algorithm to support topology simplification. In this chapter, we give a method of simplification that removes "small" holes (those having size below a user-specified threshold), while also removing other small features (such as "bumps", "cavities", and "cracks").

Our method is based on a "patchification" of the surface into connected sets of faces having similar normal vectors followed by a carefully-designed set of heuristics, based on a combination of local geometry and more global topology, intended to identify features such as holes, bumps, cavities, and cracks. The method works the extension of our work on the SPBM algorithm.

The primary advantages of our new method over prior algorithms include:

(1) speed – nearly an order of magnitude;

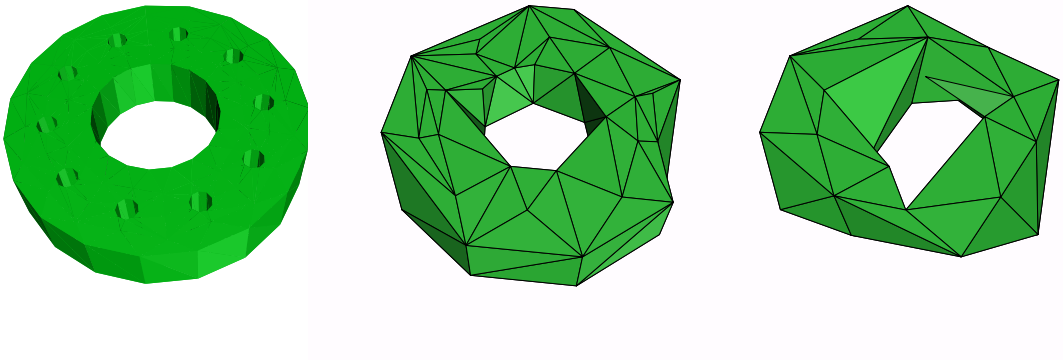
(2) very low memory consumption;

(3) a one-step simplification, which allows just the undesirable features (holes, bumps, cavities, cracks) to be removed, without requiring further geometric simplifications (as in [18, 19]); and,

(4) visual fidelity – our approach is designed to maintain visual fidelity, through the most prominent silhouette and profile features, while removing small features that are below the specified tolerance. In contrast, for example, the genus-reducing decimation approach [59] does not remove holes until no further simplification is possible without changing the genus; this results in several visual artifacts, as we will show in Section 5.4.

5.2 Previous Work

Most directly related to our work is the prior research on topology-simplifying methods. He *et al.* [32, 33] devise a topology simplifying algorithm which does not guarantee the preservation of local or global topology of a model but eliminates high frequency details in a controlled fashion. They establish levels of detail by converting an object into multi-resolution voxel representations using controlled low-pass filtering and sampling techniques. This algorithm assumes that the input model is a closed volume and therefore that voxels can be determined to be in the interior or exterior of the object. In He



(a) (b) (c)

Figure 25: The results of simplifying the Disk model from Qslim-1.0: (a) original with 752 triangles, 11 holes; (b) and (c) are the simplified images with 150, 72 triangles, respectively.

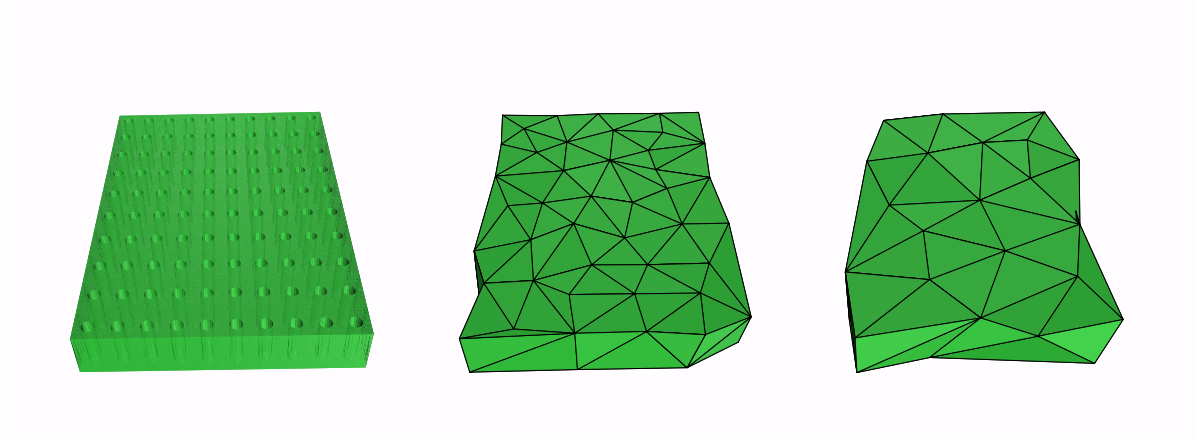
et al. [32], the marching cubes algorithm [43] is adopted to generate triangles and the a topology preserving algorithm is used, as a post-process, to remove redundant triangles. In [33], based upon the Splitting-Box algorithm [50], they introduce an AMC box for an Adaptive Marching Cubes algorithm which recursively bisects the space until either an AMC box is found or a 2-by-2-by-2 box is reached. If an AMC box satisfies the quality criteria, triangles are produced accordingly. Finally, a stitching process is imposed to eliminate cracks generated amongst different levels of boxes. This algorithm also provides an error-controlled simplification but it needs a great deal of memory and is time-consuming. Conversely, Shekhar *et al.* [60] establish an

octree by traveling marching cubes intersected with the surface of a model to delete redundant triangles. Then the octree is traversed level by level from bottom to top. At any level of the octree, if eight child cells meet the topology criteria, they are merged and replaced with a single parent cell. Schroeder [59] gives a decimation algorithm based on *edge collapse* operations, while allowing "vertex splits" when a valid edge collapse is not available. This vertex split operation is poorly controlled since it does not localize its actions to individual holes, cavities, and bumps but performs them at all places in the model (see Figure 31 and Figure 32).

One of the most recent research efforts on topology/geometry simplification is that of El-Sana and Varshney [18, 19] who devised a very clever method of extending the concept of α-shapes to polygonal models in the L_∞ metric. They are able to reduce genus, remove bumps ("protuberances"), and repair some cracks, while performing error-bounded surface simplification. Here, we build upon their efforts, while addressing some of the limitations of their method.

5.3 The Algorithm

We utilize a variant of the SPBM approach in order to detect holes, cavities, and bumps in a polygonal model. First, we use the patchification process of the SPBM approach, resulting in a decomposition of the surface into patches of polygons. During the patchification procedure, we classify edges that appear in patch boundaries according to whether or not they are "sharp": An edge is *sharp* if the angle between the outward normal vectors of the two incident triangles is greater than a specified threshold, θ_s. The implementation uses an



(a) (b) (c)

Figure 26: The results of simplifying the Fixture model from Qslim-1.0: (a) original with 18,796 triangles, 100 holes; (b), and (c) are the simplified images with 250, 150 triangles, respectively.

interval of values for the dot product of the unit normal vectors; the default is currently the interval (-0.3,0.7), corresponding roughly to angles between normal vectors that are greater than 45 degrees (i.e., dihedral angles that are less than 135 degrees).

Our heuristic is based upon a search for appropriate *sharp loops* (closed sharp chains) of sharp edges that bound potential holes, bumps and cavities. As each sharp loop is discovered, we further classify these sharp loops according to their *orientations*; the orientations will help us to distinguish between holes (which will be bounded by a pair of sharp loops with the same (clockwise) orientation) and bumps/cavities (which will be bounded by a single sharp loop or by a pair of sharp loops oppositely oriented). The orientation of a

loop also identifies which adjoining patches are considered to be the "walls" of the hole/bump/cavity, versus the (locally) "flat" plane. If we find a clockwise sharp loop L, we search each adjoining "wall" patch for an incident sharp chain that can be completed into a sharp loop L', which will serve as the "partner" for loop L. If no such sharp loop L' is found, then L defines a bump/cavity that we potentially fill. If we do find a partner sharp loop L', then we examine its orientation. If L' is also clockwise, then this pair of sharp loops defines a hole that we potentially "fill," by declaring L and L' to define new patches, and discarding the wall patches that are trapped inside the filled hole. If L' is counterclockwise, then (L, L') defines a bump or a cavity (we will not bother to distinguish); potentially, we remove the bump/cavity by filling the clockwise loop L.

Above we mentioned "potentially" filling a (clockwise) sharp loops that bound a hole/bump/cavity. Whether or not a sharp loop is filled depends on its "size". The size is defined in terms of an estimated diameter of the loop. If the loop meets the size criterion (its estimated diameter is less than the threshold α), then the loop is now declared to be the boundary of a new patch, while the "wall" patches incident to it are discarded.

We then perform a clean-up phase in which we merge adjacent patches that are nearly coplanar, i.e., whose (average) normals are nearly the same (within a user-specified angle, α_1). This is in fact a stream-lined version of the patch-merging phase of SPBM, since we exploit the fact that the patches being merged are assumed all to have normals that are similar. (This means we can avoid computing new average normals, as would be the case if we merged in patches one by one in SPBM.) We avoid creating islands within the merged

patch by requiring that each patch being merged into the larger patch shares just one connected component of its boundary with the larger patch.

Finally, we give the user two options for completing the processing:

(1) We can process the resulting patchified model with the remaining phases of the SPBM simplification method; specifically, we do boundary simplification, patch merging, and triangulation.

(2) We can perform just the final triangulation phase of SPBM, without further simplification. This may be desirable for some CAD models, if the goal is to keep the profiles exactly as they were input, only with holes/bumps/cavities removed. We remark that this is one of the strengths of our approach, to be able to preserve full silhouette details, when compared, e.g., with the method of El-Sana and Varshney [18, 19], which requires use of a polygonal simplification algorithm to be used in conjunction with the topology simplification.

In Figure 29, we used the Disk model to demonstrate these two choices. First, ten small holes of the Disk model were removed without further SBPM simplification (Figure 29(b)), in order to maintain the profile. we then can further executed SPBM simplification to obtain a coarse approximation (Figure 29(c)). In (Figure 29(d)), we also show the result of the second method of approximation which removed all the holes without destroying the outer profile.

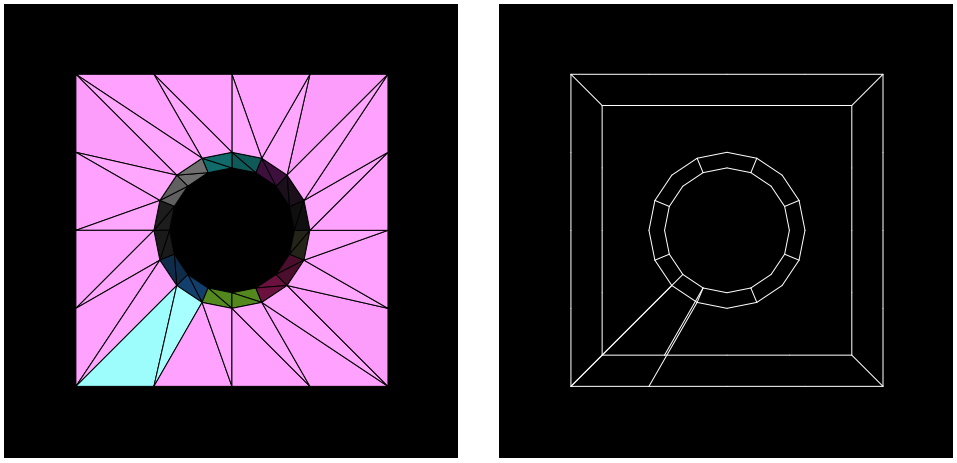
5.3.1 Finding Sharp Loops

In order to find sharp loops, we look at the sharp chains, one by one. We classify the patches incident on the sharp chain C as being on the "flat" side or the "wall" side of the chain, as follows. We distinguish two cases:

(1) *The first three vertices, v_1, v_2, and v_3, of the chain are (substantially) non-collinear.* We consider the plane π determined by the first three vertices, v_1, v_2, and v_3, of the chain and compare the normal of π with the average normal of the two triangles that are left of edges v_1v_2 and v_2v_3, as well as with the average normal of the two triangles that are right of edges v_1v_2 and v_2v_3; the side whose average normal is closest to the normal of π is declared the *flat* side, while the other side is the *wall* side. (If the average normals on both sides are not very close to the normal of π, we instead consider the last three vertices of the chain as defining π.) Figure 27(a) demonstrates the *flat* side as the flat planes, and *wall* side as patches inside the hole.

(2) *The chain C has only two vertices (one edge v_1v_2), or the first three vertices of the chain are collinear (or nearly so).* We look for other sharp chains (if any) incident on the end of C, and consider them one by one (until success) in combination with chain C, creating, in effect, a longer initial sharp chain. We use this as our starting point, as in case (1), to define a plane π and to define the flat and wall sides of the initial chain.

In fact, we have a preference in our search for chains of type (1): we continue searching the set of sharp chains, processing only those of type (1).



(a) (b)

Figure 27: Depicted are (a) the result of patchification of a hole model. (b) the boundary chains from (a).

After we have completed our investigation of all sharp chains of type (1), only then do we consider chains of type (2).

Our search for sharp loops is designed also to have a preference for starting the search with a sharp chain that is a "convex hole chain" (meaning that the chain appears locally to be a border of a convex hole in the flat plane). This is a heuristic designed to speed the search process, since most holes, in practice, have at least part of their boundary being convex. (In fact, if a hole is not smooth and has all of its bounding sharp chains reflex, then our method will not identify it as a hole.)

The actual search for a sharp loop proceeds by extending a sequence of sharp chains, $C_1, C_2, \ldots, C_k$, using any one of the sharp chains incident on

the end of C_k (in practice, there are usually very few options – 0, 1, or 2) to extend the sequence. If the sequence closes, creating a loop L, we call a procedure *Find_Partner_Loop(L)*, described below. If we get stuck, without another sharp chain to extend the sequence, we mark all of the chains in the sequence so far as "DONE", and proceed to begin a new search with some other sharp chain. The meaning of the notation "DONE" is that such a chain is not considered again for being the initial chain in the search for a sharp loop. (It may in fact be used again within some other candidate sequence of sharp chains, just not as the initial chain of such a sequence.)

The search process terminates when there are no candidate sharp chains left, meaning that all have been incorporated into loops or marked "DONE."

5.3.2 Finding Partner Loops

The search procedure, *Find_Partner_Loop(L)*, is designed to find another sharp loop, if one exists, whose wall side shares at least one patch with the wall side of L. The procedure begins by checking the orientation of L, to determine if the flat side lies inside the loop (as in the case of the bottom/top of a cavity/bump) or outside the loop (as in the case of a hole). The procedure proceeds only if the orientation is clockwise. Figure 28(a) shows the orientations of two sharp loops of a hole. Loop L_1 and loop L_2 have the same clockwise orientation. During the hole detection process, one of them will be the leading loop in the search for the other. In contrast, the sharp loops, L_1 and L_2 of a bump in Figure 28(b) have opposite orientations. Only the sharp loop L_1 oriented clockwise will be the leading loop. L_2 is the sharp loop for which L_1 is searching.

We examine each patch on the wall side of L, and use each of its sharp

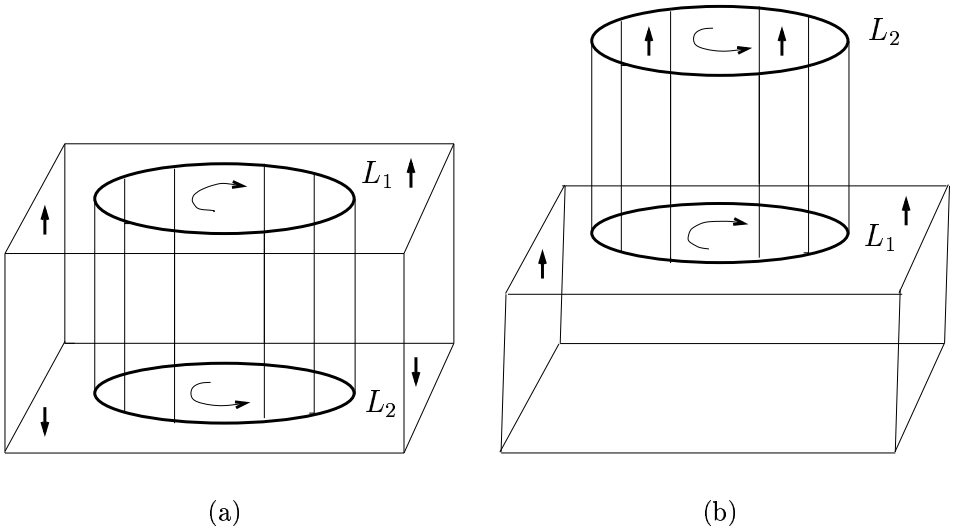


Figure 28: The orientations of sharp loops with respect to the flat sides. (a) the orientations of hole loops: L_1 and L_2 are clockwise. (b) the orientations of bump loops: L_1 is clockwise and L_2 is counterclockwise

boundary chains as the seed of a search for a sharp loop, proceeding in the search as we described in Section 5.3.1 above. If we find such a sharp loop, L', we output this loop, and the pair of loops (L, L') is considered to define a feature for possible removal.

5.3.3 Removing Holes, Bumps, and Cavities

We begin by checking the "sizes" of the pair of sharp loops, L and L'. Only if both loops are deemed to be "small" do we consider the feature to be removable. Note that if we were to remove (close) one loop that is small, while leaving the other loop unchanged (meaning that the wall patches are

left intact), we could create a non-manifold surface. (In contrast, creation of a non-manifold surface can in fact happen in the earlier method of [18, 19].)

Our test for the "size" of a loop is based on comparing the diameter of the set of loop vertices to the user-specified tolerance α. The diameter can be computed in time $O(n \log n)$, for a loop with n vertices, using a convex hull algorithm. Alternatively, it can be estimated by using a bounding box, in time $O(n)$. However, we have found it to be adequate, in practice, to approximate the exact diameter using a simple $O(1)$-time sampling method that selects either a pair of points (in the convex loop case) or a triple of points (in the general case; we then estimate the diameter using that of the circle through the three points). The sampling is based on selecting points that are spaced evenly (in index) along the list of loop vertices.

Once we have deemed both sharp loops to be "small," the removal of the corresponding feature is very simple: each clockwise loop (at least one of L and L' is clockwise, by the nature of our algorithm) is declared to be a patch boundary, while the adjoining wall patches are discarded. Figure 30

5.3.4 Repairing Cracks

Our approach can also cope with models that have "cracks" and can repair them, allowing it to perform some amount of model cleanup of potential glitches in the input data. (A similar feature is provided in the method of [18, 19].) In particular, we consider the set E_1 of edges that have only one incident triangle to be potential crack edges. Each such edge is considered to be oriented in the direction such that the one triangle is to the left (when viewed from the outside of the model). These directed edges E_1 form a network which we

search for directed loops. (For the present, we will omit a discussion of this search procedure.) The resulting loops of crack edges are "closed" by declaring them to be new patches. (A size criterion of a different type can be applied to filter out those crack loops that may be features of the model that one wants to preserve; e.g., we can apply a threshold to the radius of a largest inscribed disk within the candidate patch.)

5.4 Results and Discussion

In order to evaluate the performance of our implementation (in C) of the algorithm described in the previous section, we applied it to several datasets, including those reported in [18, 19], allowing us to do direct performance comparisons with this prior work. (We thank Jihad El-Sana for providing us with these datasets.) Images of some of the models are given in the color plates. We report our performance based on an R10000 SGI Indigo2 workstation (195 MHz CPU, 256 MB RAM, 32 KB data/instruction cache, and 1 MB secondary cache, IRIX 6.5). Compilation was with "cc -O2". Table 2 summarizes the results of our experiments on these datasets.

Images of many of the models, and of some simplifications, are shown in the color plates. Figure 31 also show typical results of applying the "Topology Modifying Progressive Decimation" algorithm of Schroeder [59] (as implemented and available in **vtk**) to two models ("Battery" and "Disk"). It is seen that decimation may generate fairly erratic simplifications, resulting in low visual fidelity. This is because it is performing simplification that attempts to preserve topology, and only removes a hole (changing the genus) when no

genus-preserving decimation step is available. Also, for example, **vtk** is unable to simplify the "Fixture" dataset below 81 triangles(see Figure 31(c)); its final output contains several "singleton edges" floating in the middle of the model, where the holes used to be.

In generating the data for our Table 2, each level of detail of a model is generated directly from the original model. For some models we show results for various values of the threshold α. In these cases, the model changes in different stages according to α; e.g., for a small value of α, the small holes in the "Disk" model get removed, while for a larger value of α, even the large central hole is removed.

Our whole simplification approach is orders of magnitude faster than the "Genus-Reducing Simplification" method for those datasets reported in [18, 19] (using a single R10000 processor of an SGI Challenge machine). In particular, for the "Disk" (while removing the small holes) they used 0.3 seconds for genus reducing, and another 8.4 seconds for the remaining (genus-preserving) simplification; in contrast, we obtained our result in a total of 0.06 seconds (including reading the data, performing orientation tests and normal computations for triangles). For the "Box" they worked for 0.3 seconds, plus 2.0 seconds, while we clocked 0.08 seconds. We also remark that their final "Box" simplification results in 14 triangles, as opposed to the 12 triangles in our simplification, which gives the cleanest possible "box." For the "Block" model, they used 5.6 seconds, plus 20 seconds, while we needed 0.36 seconds. Finally, for the "Fixture" model, they computed for 129 seconds, plus 42 seconds, while our algorithm needed 1.03 seconds in total (and also have a final simplification of 12 triangles, versus 62). (We remark that there appears to be an inconsistency

reported in the table of [18, 19], where they report the number of triangles in "Block" to be 17644, instead of the 7084 that are in the accompanying figure. Similarly, the timings shown for "Fixture" seem to be for the model having 18796 triangles, not the 74396 that is reported. The authors of [18, 19] are in the process of addressing this possible inconsistency.)

The reductions of most datasets reach the minimum number of triangles required for maintaining the original shape of the models; e.g., the "Battery", "Box", and "Fixture" each get reduced to just 12 triangles. This demonstrates that our merging process does a good job of cleaning redundant patches on the flat planes.

While our method has some similarities in how it detects holes, when compared with [18, 19], our approach is substantially faster for several reasons:

- **fast determination of sharp edges:** Based on our patchification approach, we do not need to investigate all original edges since all sharp edges are located on the borders of patches. Instead, we only need to check all lists of edges during the patchification process (the first phase of the SPBM approach).

- **fast determination of loops of sharp chains:** In the approach of [18, 19], they adopted a linear-time depth-first scanning strategy to search "tessellation chains." Their approach depends on the number of sharp edges. However, the running time of our search depends only on the number of sharp *chains*, which are the lists of sharp edges.

- **fast removal of holes, bumps, and cavities:** When our method cleans holes, bumps, and cavities, we first search all patches composing

them, and then remove the relevant patches; in contrast, the method of [18, 19] searches the set of all triangles to be deleted.

- **fast simplification:** After the completion of our feature detection process, the simplification is almost complete, as all that remains is either (a) the retriangulation of patches (if the user decides to output without further simplification), or (b) the patch boundary simplification, merging, and retriangulation steps of SPBM. In contrast, the method of [18, 19] requires a more involved simplification procedure, as the "flat" patches of the model (from which the holes were removed) have not yet been simplified at the completion of their hole-filling procedure.

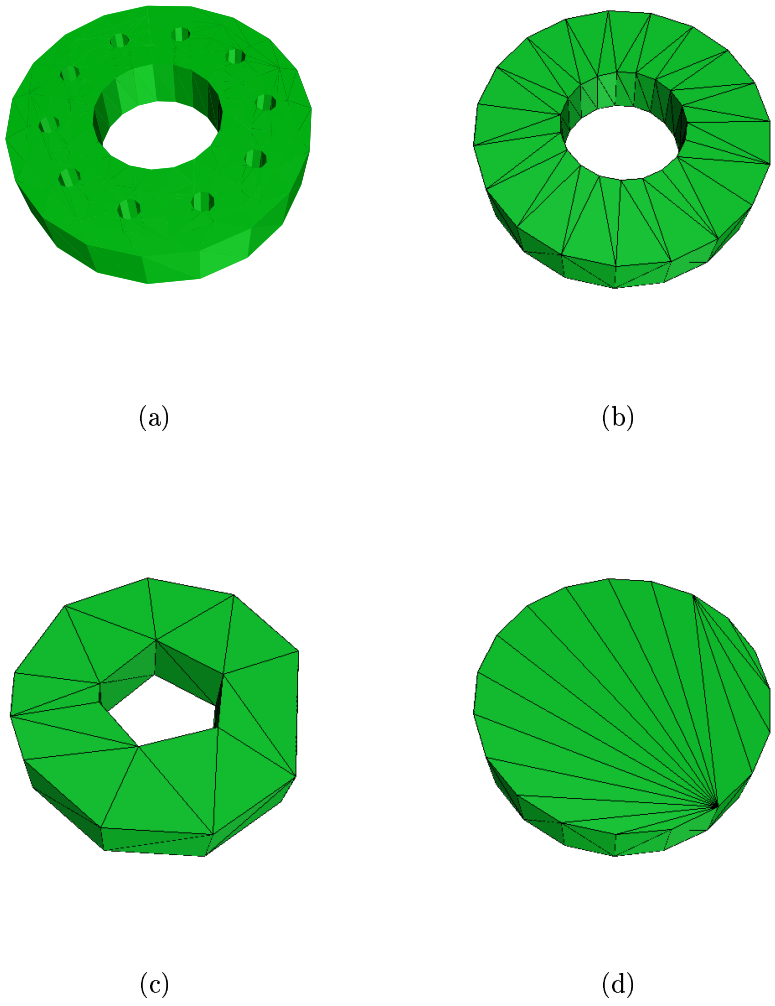
The memory required by our method is also very low; it is essentially the same as required by SPBM and reported in chapter 4. The point is that, other than the original input data, very little storage is needed, since, other than the face-adjacency information contained in the input model, the only data structure utilized by the algorithm is the edge list that stores the profile curves (patch boundary chains) and their simplifications. In contrast, most other methods require auxiliary data structures for storing intermediate versions of the simplified model.

Nevertheless, our approach has some limitations. First, our method relies on features being defined by relatively "sharp" edges; e.g., a "smooth" hole as in a bagel would not be removed using our algorithm. Second, our method potentially removes some features that one may want to keep, such as a long skinny "pipe" connection between two larger components (picture a "dumbbell"). Of course, in some applications, removing such skinny pipes

Experimental Results						
Model	**Original Tris**	h	c	b	**Final Tris**	**Total Time (s)**
Battery	616	0	0	9	(162, 132, 12)	(0.07, 0.06, 0.04)
Disk	752	11	0	0	(152, 72)	(0.1, 0.06)
Box	1,612	4	0	0	12	0.08
Gear	2,478	10	30	0	(1372, 232)	(0.78, 0.19)
Cylinder	5,392	51	0	0	(356, 312)	(0.7, 0.65)
Block1	6,356	16	0	0	28	0.30
Block2	7,084	32	0	0	44	0.36
Block3	17,644	80	0	0	44	0.97
Fixture1	18,796	100	0	0	12	1.03
Fixture2	74,396	400	0	0	12	8.32

Table 2: Results of our approach on several datasets. Here, h is the number of holes detected, c is the number of cavities, and b is the number of bumps. For "Battery" we give results for $\alpha = 4000$,$\alpha = 6500$, and $\alpha = 12000$; for "Disk" we give results for $\alpha = 4$ and $\alpha = 121$; for "Gear" we give results for $\alpha = 3$ and $\alpha = 4$; for "Cylinder" we give results for $\alpha = 20$ and $\alpha = 400$. Timings are in seconds, and are *total* times, including reading the data, performing orientation tests and normal computations for triangles.

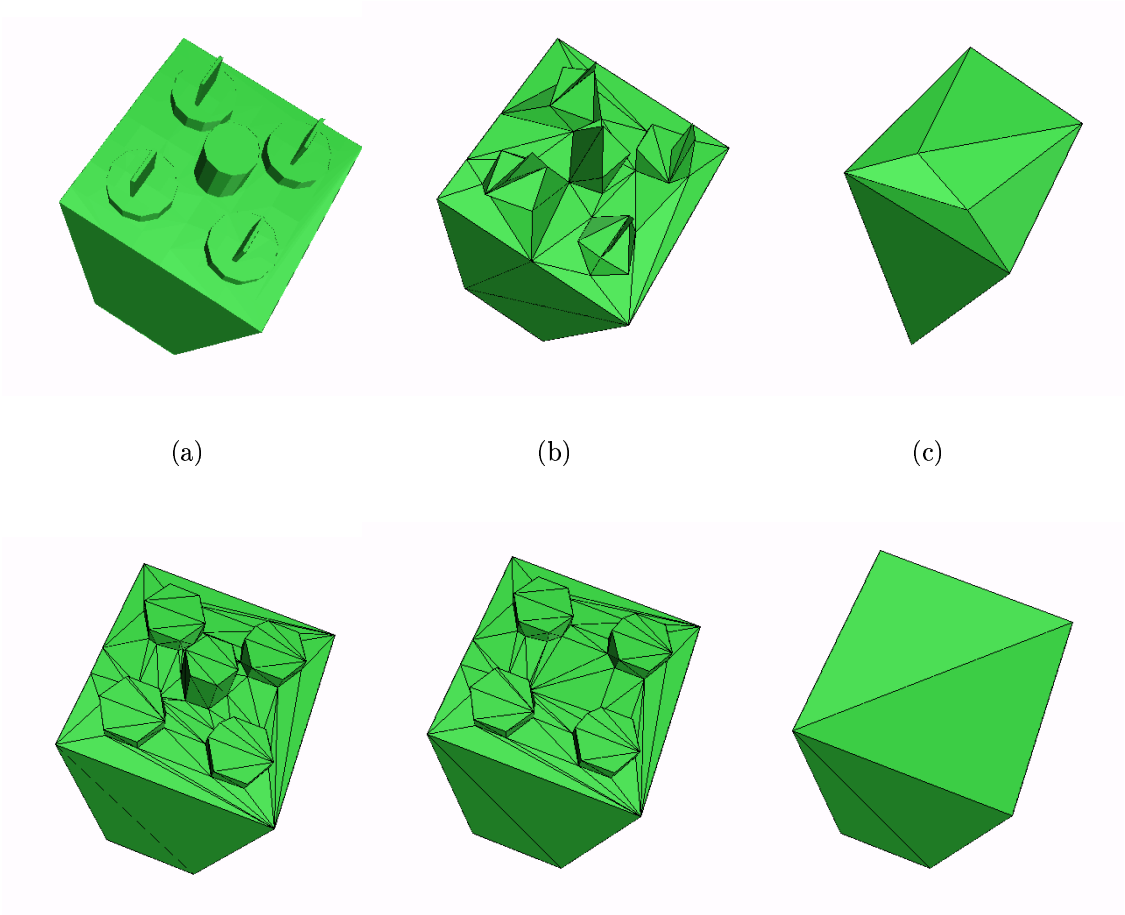
may be the desired outcome; however, further testing would have to be added to our algorithm to make this distinction. Third, as with SPBM, we are not guaranteeing that the simplified model lies within a rigid tolerance ϵ of the original; we only preserve Hausdorff distance on the simplified chains of edges that form patch boundaries (and result in visual profiles).



(a) (b)

(c) (d)

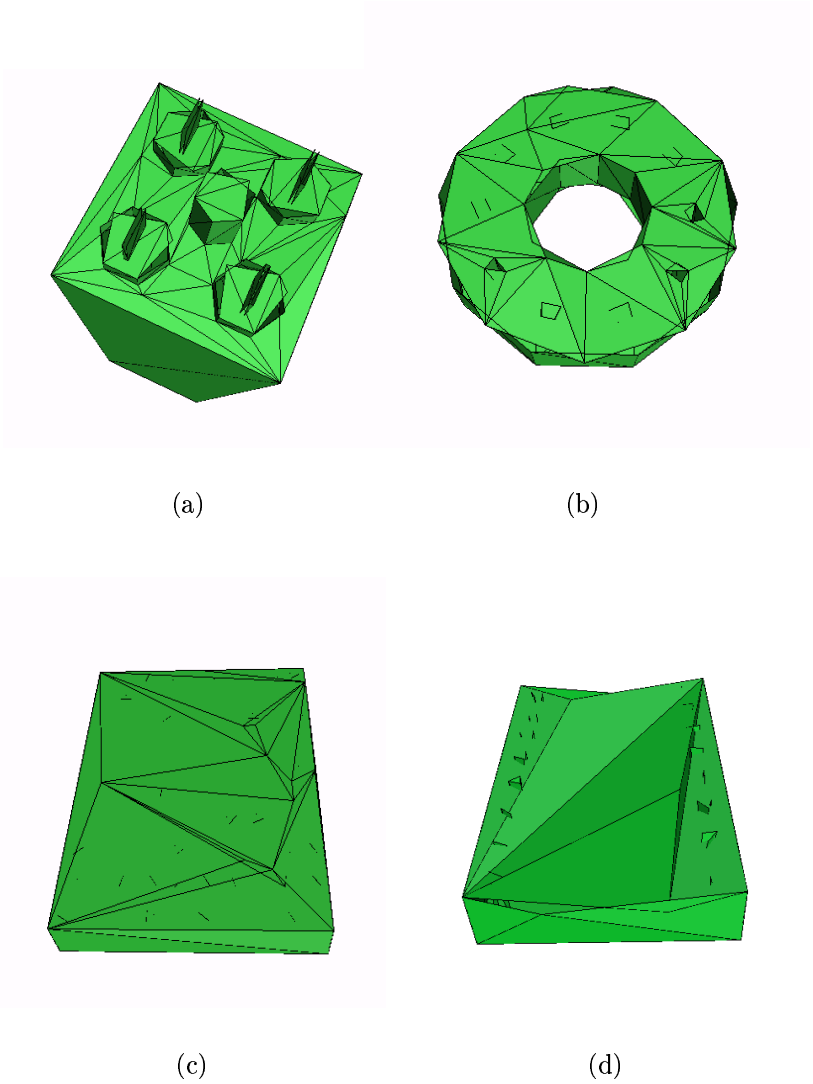
Figure 29: The results of simplifying the Disk model: (a) original with 752 triangles, 11 holes; (b) first level topology simplification, removing 10 small holes; (c) further SPBM simplification with 64 triangles, or (d) second level topology simplification with 72 triangles.



(a) (b) (c)

(d) (e) (f)

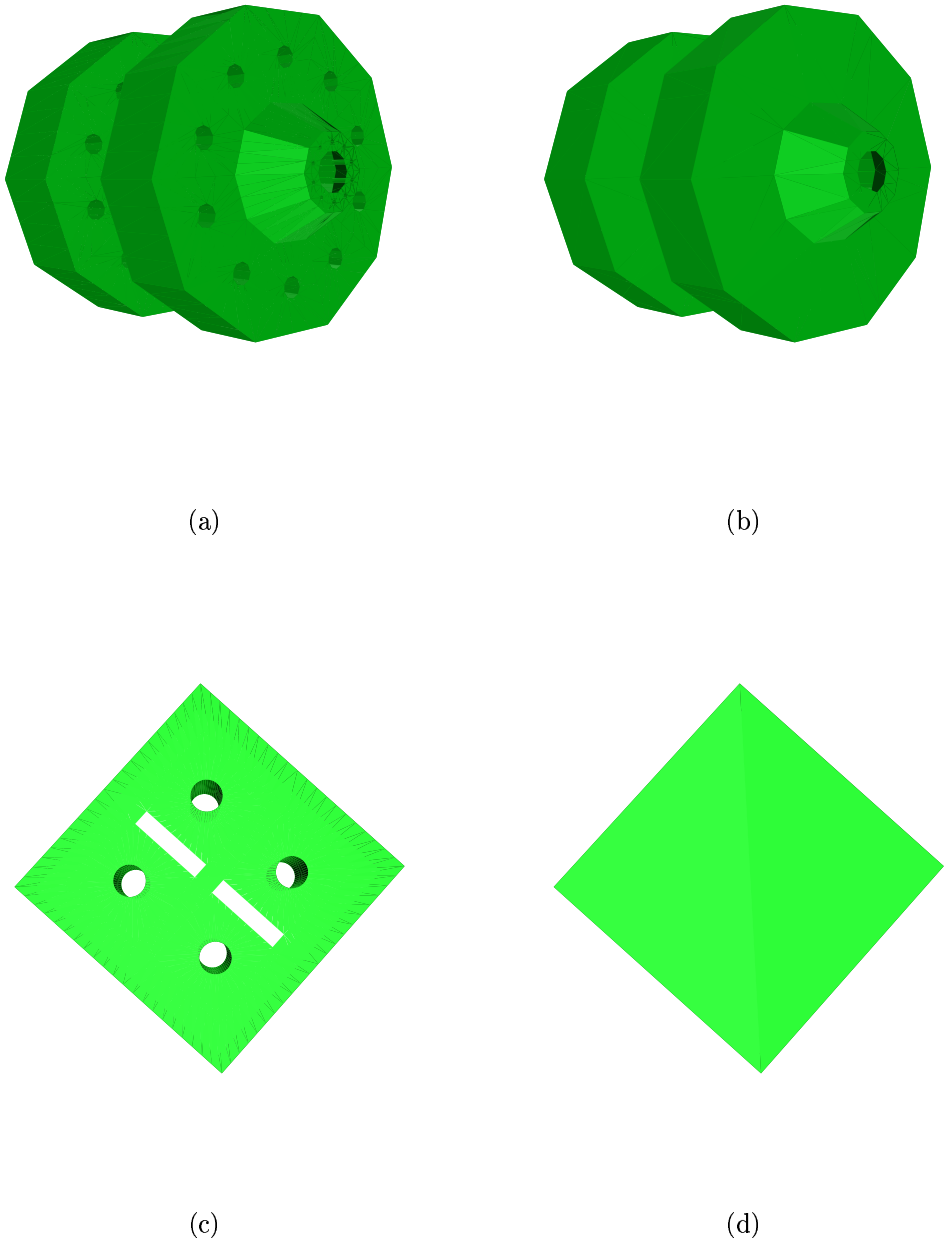
Figure 30: The results of simplifying the Battery model: (a)original with 616 triangles; (b), and (c) are the simplified images from Qslim-1.0 with 100, 12 triangles, respectively; (d), (e), and (f) are our results with 162, 132, and 12 triangles respectively.



(a) (b)

(c) (d)

Figure 31: The results from vtk: (a) the simplified Battery image with 154 triangles; (b) the simplified Disk image with 220 triangles; (c) the simplified Fixture image with 81 triangles; (d) the simplified Block image with 154 triangles.



(a) (b)

(c) (d)

Figure 32: Other our results: (a) the original Box image with 1,612 triangles; (b) the simplified Box image with 12 triangles; (c) the original Gear image with 2,478 triangles; (d) the simplified Gear image with 232 triangles.

Chapter 6

Conclusions and Future Directions

In this dissertation, we have presented three surface simplification algorithms for general 3D models: greedy-cuts triangulation, patch boundary merging simplification, and genus reduction simplification. At the heart of our techniques is the *patchification* process which decomposes the original surface into numerous patches. We have demonstrated the properties of several simple heuristic *patchification* algorithms through several experiments.

Our first contribution was to propose an the extension of the greedy-cuts algorithm of Silva and Mitchell [61, 62, 63] (which worked only for terrain) for general 3D models with bounded error. This technique attempts to triangulate by adding the largest possible triangle at each stage. In order to extend the algorithm for general 3D surfaces, we developed a technique for bounding the error of the approximation. Also, we proposed a repair procedure to guarantee the termination of this algorithm. Our experiments showed

that general 3D greedy-cuts was slow. However, we have found that there is a alternative method which accelerates the entire process at the expense of the ability to explicitly bound the error. By discarding the burden of the greedy-cuts triangulation process, we developed SBPM, a fast and memory-efficient simplification technique based on a patch merging scheme. This technique produces good visual quality while being must faster. Our results are very encouranging. In fact, our SBPM technique is faster than QSlim 2.0 (currently leading 3D simplification code), while using less memory and producing results of similar visual quality. Our third contribution builds on SBPM, and extends it to deal with models with high genus and small bumps by an efficient hole detection strategy. Again, compared to the previous technique of El-Sana [18], we have obtained a dramatic speedup.

In our experiments we show that our algorithms provide high speed and very good visual fidelity for CAD models (see Fandisk model in Chapter 4). The output of SBPM is not just a simplified mesh. It also includes the borders of all patches which include the feature curves. The identification of surface feature curves has been widely used in the recent years [54].

There are many opportunities for future work, including: automatic level-of-detail generations [21], and generalized view-dependent simplifications [20]. We believe our approach is extensible in similar ways. Given its versatility, speed, and simplicity, it might lead to improved techniques with these characteristics.

6.1 Patch Boundary Merging Simplification

The main strength of our SPBM algorithm is its speed and low memory consumption. Our future research agenda is to extend the functionalities of the patchification process and the patch merging process to cover an automatic, general, and view-dependent approach.

- **better visual quality**: In general, SPBM generates approximations with high visual fidelity for CAD models but slightly poor quality for non-smooth surfaces. We plan to explore more advanced techniques to enhance the visual quality of the simplified mesh. For this, first, we would need a more sophisticated patchification strategy, one that can handle non-smooth surfaces while maintaining most of the surface features. Then, we plan to consider the implications of adding Steiner points in the triangulation (e.g., a simple way of creating a Steiner point is to calculate the average position of all vertices on the patch boundary).

- **automatic**: At this time, SPBM uses a user-specified error parameter to guide the approximations. We are interesting in exploring extensions that are able to automatically generate several levels of detail (without user intervention). It would be desirable that such a hierarchy be built in a single pass of the simplification algorithm. (At this time, our technique would require multiple passes.)

- **general**: Lots of polygonal datasets often contain non-manifold vertices, edges, cracks, and coincident polygons. We plan to extend SBPM to also

deal with imperfect models. A possible approach is to extend the patchification process in order to include the identification of these defective parts, and then handle them separately.

- **view-dependent**: Once SBPM is able to automatically compute several level of details, it is possible to assign error weight to each edge, and stored these in a "merging diagram" which can be used for view-dependent simplification. There are several interesting issues here, including exploring different techniques for patch triangulation, possibly real-time triangulations (which are done on-line) that generate triangle strips directly.

6.2 Topology Simplification

With respect to the topology simplification, we also have several interesting things to work on:

- **multiple objects**: Similar to the discussion in the previous section, we would also like to create a general, automatic, and view-dependent topology simplification algorithm. Since our hole detection process identifies all holes, bumps, and cavities during the preprocessing phase of the simplification algorithm, we can add this information, for example the size of a hole, to the "error weight" of the border chains of this hole and then automatically remove this hole. As another important step, we could extend our method in order to deal with multiple objects. Recently, Erikson [21] and El-Sana [20] have proposed new topology

simplification schemes for multiple objects based on the generations of "virtual edges" among unconnected objects. Using edge-collapse actions, two unconnected objects are joined together by collapsing the "virtual edges" connected to these two objects. Their methods can drastically reduce the number of approximated triangles while preserving the volume of the entire environment. We wish to investigate a similar but less complex approach. We propose to construct an adaptive 3D grid partition for the entire environment and then to generate "virtual edges" among the patch borders for unconnected objects residing within the sub-domain of the grid. In contrast with those previous methods, we could attempt to generate "virtual patches" among unconnected objects or separate branches of a single object during the preprocessing phase for the entire environment by connecting "virtual edges" and with boundary chains. These "virtual patches" could then participate in the merging process. We believe that this approach will provide a well-controlled topology simplification.

- **robust**: Our current hole detection process within our topology simplification algorithm cannot accurately identify a hole or a pipe. This process only locally searches "hole loops" among sharp edges specified during the patchification process. Since a hole and a pipe both contain two hole loops, no further information is generated by our current approach which could distinguish them. To overcome this problem, we could introduce grid partitions to check the adjacent environment of each detected hole. Unfortunately, since our method relies on these relatively "sharp" edges, holes with smooth curvature cannot be recognized by our

current detection process. A possible way to identify these smooth holes is based on our advanced patchification process. The second patchification strategy introduced in chapter 3 could specify the patch normals, and then after comparing all patch normals close with respect to the grid structure, we could identify those smooth holes that need to be filled.

Bibliography

[1] http://www.cs.cmu.edu/afs/cs/user/garland/www/multires/index.html.

[2] http://www.limsi.fr/Individu/krus/CG/LODS.

[3] http://www.ams.sunysb.edu/~jsbm/surfapprox.html.

[4] M. E. Algorri and F. Schmitt. Mesh Simplification. *Computer Graphics Forum (Eurographics'96 Proc.)*, 15(3): 78-86, 1996.

[5] D. Banks. Interactive Manipulation and Display of Two-Dimensional Surfaces in Four-Dimensional Space. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 197–207.

[6] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently Approximating Polygonal Paths in Three and Higher Dimensions. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, 1998.

[7] G. Barequet, M. Dickerson, and D. Eppstein. On Triangulating Three-Dimensional Polygons. *Comput. Geom. Theory Appl.*, 10(3):155–170, 1998.

[8] B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for Polyhedral Surface Decomposition: An Experimental Study. *Comput. Geom. Theory Appl.*, 7:327–342, 1997.

[9] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A General Method for Preserving Attribute Values on Simplified Meshes. In *Visualization 98*, page ?? IEEE Computer Society Press, 1998.

[10] P. Cignoni, C. Montani, and R. Scopigno. A Comparison of Mesh Simplification Algorithms. *Computers and Graphics*, 22/1:37–54, 1998.

[11] J. Cohen, M. Olano, and D. Manocha. Appearance-Preserving Simplification. In *Proc. SIGGRAPH '98*, Computer Graphics Proceedings, Annual Conference Series, pages 115–122, July 1998.

[12] D. Cohen-Or. Special Issue on Scene Simplification. *Computers and Graphics*, 22/1:1–2, 1998.

[13] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright Simplification Envelopes. *Computer Graphics (SIGGRAPH '96 Proceedings)* 1996

[14] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 303–312, August 1996.

[15] L. De Floriani, P. Magillo, and E. Puppo. Efficient Implementation of Multi-Triangulation. *IEEE Visualization '98* pages 43-50 October 1998.

[16] D. Dooley and M. Cohen. Automatic Illustration of 3D Geometric Models: Lines. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, pages 77–82.

[17] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 173-182, August 1995.

[18] J. El-Sana and A. Varshney. Controlled Simplification of Genus for Polygonal Models. In *IEEE Visualization '97 Proceedings*, pages 403–410, San Francisco, CA, 1997. ACM/SIGGRAPH Press.

[19] J. El-Sana and A. Varshney. Topology Simplification for Polygonal Virtual Environments. *IEEE Trans. Visualizat. Comput. Graph.*, 4(2):135–144, Apr. 1998.

[20] J. El-Sana and A. Varshney. Generalized View-Dependent Simplification. *Proceedings Eurographics'99.*, to appear, Aug. 1999.

[21] C. Erikson and D. Manocha. GAPS: General and Automatic Polygonal Simplification. *Proceedings of 1999 Symposium on Interactive 3D Graphics* , pages 79–88, Apr. 1999.

[22] M. Garland. Personnal Communication, February 1999.

[23] M. Garland and P. S. Heckbert. Surface Simplification using Quadric Error Metrics. In *Computer Graphics (SIGGRAPH'97 Proceedings)*, pages 209–216, 1997.

[24] M. Garland and P. S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error metrics. In *Visualization 98*, pages 263–269, October 1998.

[25] A. Guéziec. Surface Simplification Inside a Tolerance Volume. *Technical Report RC 20440*, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10958, 1996.

[26] A. Guéziec, F. Lazarus, G. Taubin, W. Horn. Surface Partitions for Progressive Transmission and Display, and Dynamic Simplification of Polygonal Surfaces. In*Proceddings VRML'98*, pages 25-32, February, 1998.

[27] B. Hamann. A Data Reduction Scheme for Triangulated Surfaces. *Computer Aided Geometric Design*, 11(2): 197-214, April 1994.

[28] P. S. Heckbert and M. Garland. Fast Polygonal Approximation of Terrains and Height Fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[29] M. Held. Efficient and Reliable Triangulation of Polygons. In *Proc. Comput. Graphics Internat. 1998*, pages 633–643, June 1998.

[30] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. Technical report, University at Stony Brook, 1998.

[31] P. Hinker and C. Hansen. Geometric Optimization. In *Visualization 93*, pages 189–195, October 1993.

[32] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel Based Object Simplification. In *IEEE Visualization '95 Proceedings*, pages 296–303. ACM/SIGGRAPH Press, 1995.

[33] T. He, L. Hong, A. Varshney, and S. Wang. Controlled Topology Simplification. *IEEE Trans. Visualizat. Comput. Graph.*, 2(2):171–184, Apr. 1996.

[34] T.-C. Ho, J. S. B. Mitchell, and C. T. Silva. The Simplified Patch Boundary Merging Algorithm for Polygonal Surface Simplification. Manuscript (submitted), March. 1999.

[35] H. Hoppe. Progressive Meshes. *Computer Graphics Proc. Ann. Conf. Series (Proc. SIGGRAPH '96)*, pp. 99-108, Aug. 1996.

[36] H. Hoppe . View-Dependent Refinement of Progressive Meshes. In*Proceedings of SIGGRAPH'97*, pages 198-208, Aug. 1997.

[37] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, Aug. 1993.

[38] V. Interrante, H. Funchs, and S. Pizer. Enhancing Transparent Skin Surface with Ridge and Valley Lines. *Visualization 95.*, page 52-59 1995.

[39] A. D. Kalvin and R. H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Comput. Graph. Appl.*, 16(3):64–77, May 1996.

[40] R. Klein, G. Liebich, and W. Straβer. Mesh Reduction with Error Control. *IEEE Visualization '96* pp. 331-338.

[41] A. W. E. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Computer Graphics Proceedings, Annual Conference Series 1998*, pages 95–104, 1998.

[42] P. Lindstrom and G. Turk. Fast and Memory Efficient Polygonal Simplification. In *Visualization 98*, pages 279–286, October 1998.

[43] W. Lorensen and H. Cline. Marching cubes: A High Resurlution 3D Surface Construction Algorithm. *Comput. Graph.*, 21(4):163–170, 1987.

[44] K-L Low and T-S Tan. Model simplification using Vertex-Clustering. *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 75–81, 1997.

[45] D. Luebke. A Survey of Polygonal Simplification Algorithms. Technical Report 97-045, Dept. Computer Science, University of North Carolina at Chapel Hill, 1997.

[46] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In*Proceedings of SIGGRAPH'97*, pages 198-208, Aug. 1997.

[47] M. Lounsbery, T. DeRose, and J. Warren. Multiresolution Analysis for Surface of Arbitrary Topological Type. In *Transactions on Graphics*, 16(1) 34–73, January 1997.

[48] K-L Ma and V. Interrante. Extracting Feature Lines from 3D Unstructured Grids. In *IEEE Visualization '97 Proceedings*, pages 285–292, San Francisco, CA, 1997. ACM/SIGGRAPH Press.

[49] J. S. B. Mitchell and S. Suri. Separation and Approximation of Polyhedral Surfaces. *Computational Geometry: Theory and Applications*, **5**, 1995, 95–114.

[50] H. Muller and M. Stark. Adaptive Generation of Surfaces in Volume Data. *The Visual Computer*, 9:182–199, 1993.

[51] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[52] D. E. Pearson and J. A. Robinson. Visual Communication at Very Low Data Rate. *Proceedings of the IEEE*, volume 4 pages 795-812, April 1985.

[53] J. Popović, and H. Hoppe. Progressive Simplicial Complexes. In*Proceedings of SIGGRAPH'97*, pages 217-224, Aug. 1997.

[54] R. Raskar and M. Cohen. Image Precision Silhouette Edges. *Proceedings of 1999 Symposium on Interactive 3D Graphics* , pages 135–140, Apr. 1999.

[55] R. Ronfard and J. Rossignac. Full-Range Approximation of Triangulated Polyhedra. *Computer Graphics Forum*, 15(3), Aug. 1996. Proc. Eurographics'96.

[56] J. Rossignac and P. Borrel. Multi-Resolution 3D Approximation for Rendering Complex Scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy.

[57] S. Satio and T. Takahashi. Comprehensible Rending of 3-D Shapes. In *Proc. SIGGRAPH '90*, pages 197–206, 1990.

[58] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of Triangle Meshes. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[59] W. J. Schroeder. A Topology Modifying Progressive Decimation Algorithm. In *IEEE Visualization '97 Proceedings*, pages 205–212, San Francisco, CA, 1997. ACM/SIGGRAPH Press.

[60] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-Based Decimation of Marching Cubes Surfaces. In *IEEE Visualization '96 Proceedings*, pages 335–342. ACM/SIGGRAPH Press, 1996.

[61] C. T. Silva, J. S. B. Mitchell, A. E. Kaufman. Automatic Generation of Triangular Irregular Networks using Greedy Cuts. *IEEE Visualization '95* pp. 201-208.

[62] C. T. Silva, J. S. B. Mitchell. Greedy Cuts: An Advancing Front Terrain Triangulation Algorithm. *ACM Symposium on Geographic Information Systems 1998*

[63] C. T. Silva. Parallel Volume Rendering of Irregular Grid. PhD thesis, *Dept. of Computer Science, State University of New York at Stony Brook*, 1996.

[64] M. Soucy and D. Lauredeau. Multi-Resolution Surface Modeling from Multiple Range Views. In*Conf. on Computer Vision and Pattern Recognition (CVPR '92)* pages 348-353, June 1992.

[65] M. Soucy and D. Lauredeau. Multiresolution Surface Modeling Based on Hierarchical Triangulation. *Computer Vision and Image Understanding* Vol. 63. No. 1. January. pp. 1-14. 1996.

[66] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive Forest Split Compression. In *Proc. SIGGRAPH '98*, Computer Graphics Proceedings, Annual Conference Series, July 1998.

[67] G. Taubin. Estimating the Tensor of Curvature of a Surface from a Polyhedral Approximation. In *Proceedings of the 5th International Conference on Computer Vision (ICCV)*, pages 902–907, 1995.

[68] G. Turk. Re-Tiling Polygonal Surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[69] G. Turk and M. Levoy. Zippered Plygon Meshes from Range Images. In *Proceedings of SIGGRAPH '94)*, Computer Graphics Proceedings, Annual Conference Series, pages 311–318, July 1994.

[70] J. C. Xia, J. El-Sana, A. Varshney. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models. *IEEE Transaction On Visualization And Computer Graphics* Vol. *3* No. 2 April-June 1997.