1995

Unstructured surface and volume decimation of tessellated domains

Kevin Joseph Renze
Iowa State University

Unstructured surface and volume decimation of tessellated domains

by

Kevin Joseph Renze

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Mechanical Engineering
Major: Mechanical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Members of the Committee:

Signature was redacted for privacy.

Signature was redacted for privacy.

Signature was redacted for privacy.

Signature was redacted for privacy.

Iowa State University
Ames, Iowa
1995

To Janet, with love ...

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEFINITIONS

The following general definitions and notation have been extracted from Preparata and Shamos [28] for reference convenience.

- A polygon is *simple* if there is no pair of nonconsecutive edges sharing a point.
- A simple polygon partitions the plane into two disjoint regions, the *interior* (bounded) and the *exterior* (unbounded) that are separated by the polygon.
- A simple polygon is said to be *monotone* if its boundary can be decomposed into two chains monotone with respect to the same straight line.
- A domain D in E^d is *convex* if, for any two points q_1 and q_2 in D, the segment $\overline{q_1 q_2}$ is completely contained in D. From an informal viewpoint, a convex polygon has a boundary which could be traversed in an orderly counterclockwise direction using exclusive left-hand turns.
- A polygon is defined to be *convex* if its interior is a *convex set.*
- A simple polygon P is *star-shaped* if there exists a point z not external to P such that for all points p of P the line segment $\overline{zp}$ lies entirely within P.

Additional definitions and notation are consistent with those presented by Okabe et al. [27].

- The number of incident edges at a given vertex is the vertex *degree.*
- A *two-dimensional* Delaunay tessellation is called a Delaunay *triangulation.*
- A *three-dimensional* Delaunay tessellation is called a Delaunay *tetrahedrization.*

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Dr. James H. Oliver, my major professor, for his technical guidance, encouragement, and friendship. I'm also grateful for the many invaluable technical discussions shared with Dr. Richard Hindman and Dr. Lin-Lin Chen. The insight and criticism offered by Dr. James Bernard, Dr. Martin Vanderploeg, and Dr. Jerald Vogel during the course of this investigation are also greatly appreciated. I truly enjoyed the working environment shared with Philip Theruvakattil, Yunching Huang, Jim Troy, the *O'boys*, and the multitude of other talented colleagues in the Iowa State University Visualization Laboratory.

Financial support for this research was provided by National Science Foundation Young Investigator Award (Grant No. DDM-9258114) and by funds from the Office of Naval Research (Grant No. N0001492J4092). Computing resources were made available by the Iowa State University Visualization Laboratory and by the Iowa Center for Emerging Manufacturing Technology.

The horse leg sections and initial surface connectivity were acquired from Joel Marquart. The F-117 aircraft surface data was obtained courtesy of Dr. Marshal L. Merriam and Dr. Kumaran Kalyanasundaram. Dr. Thomas Ramin created the curved channel volume geometry (CFD mesh). John C. Minor generated the heat sink volume geometry. Special thanks to Maria Giannakouros for editing the manuscript.

My primary source of inspiration to complete graduate school was a deep respect for the challenging work; competent, dedicated individuals; and humorous, open-minded intellects I encountered amidst the majority of persons I worked with in research and industrial environments. In particular, I would like to thank my former mentors and colleagues from The Boeing Company and NASA Ames Research Center. I entertain a strong conviction that practical industry experience is critical to maturing as a professional engineer. As such my view of a Ph.D. finds value in the "demonstrated ability to define, solve, and communicate technical results for relevant problems."

The best engineering educators possess outstanding academic credentials, pursue creative, relevant research interests, practice engineering in conjunction with industrial and government organizations, and convey classical and current information in a dynamic learning environment. I owe a debt of thanks to all the dedicated educators who have touched my life.

Finally, special thanks to my parents for stressing the importance of education. I know you didn't mean for me to take it to this extreme, but I don't always listen. Thanks for walking me to the bus stop on that first fateful day ...

ABSTRACT

A general algorithm for decimating unstructured discretized data sets is presented. The discretized space may be a planar triangulation, a general 3D surface triangulation, or a 3D tetrahedrization. The decimation algorithm enforces Dirichlet boundary conditions, uses only existing vertices, and assumes manifold geometry. Local dynamic vertex removal is performed without history information while preserving the initial topology and boundary geometry. The tessellation at each step of the algorithm is preserved and, in the pathological case, every interior vertex is a candidate for removal. The research focuses on how to remove a vertex from an existing unstructured n-dimensional tessellation, not on the formulation of decimation criteria. Criteria for removing a candidate vertex may be based on geometric properties or any scalar governing function specific to the application. Use of scalar functions to adaptively control or optimize tessellation resolution is particularly applicable to the computer graphics, computational fluids, and structural analysis disciplines. Potential applications in the geologic exploration and medical or industrial imaging fields are promising.

CHAPTER 1. INTRODUCTION

The casual observer may be introduced to the general topic of tessellation (i.e., space discretization) by an analogy to a childhood learning exercise. The game is *dot-to-dot*: a combination of counting, drawing, visualization, and imagination. From a global viewpoint, the child is taking a problem and discretizing the space by connecting a series of points with line segments. As the child defines the image boundary, the space is divided into interior and exterior regions of interest. The visual impact of this simple subdivision technique is significant, usually allowing the individual to recognize the image prior to completion.

Although the child's problem is of simple visual interest, it introduces the tessellation concepts of subdividing a spatial domain, defining boundaries, and constructing a connectivity scheme. Many computational problem solving algorithms are based on the simple concept of domain discretization introduced above.

Motivation

The term *"decimate"* describes the process of removing polygons, edges, or vertices from a geometric configuration. The goal is to intelligently reduce the number of primitives required to accurately model the problem of interest. Unstructured decimation algorithms have emerged primarily from research in surface reconstruction and computer graphics. However, scientific and engineering applications of spatial discretization to model physical systems and visualize solutions are abundant. Therefore, the potential impact of a general decimation scheme may be quite broad.

Experimentally measured data are commonly generated by medical, laser, and satellite imaging equipment, which are capable of producing large volumes of geometric data—often more information than can be displayed or manipulated in reasonable time. Sources include magnetic resonance scanners, range cameras, and satellites. Numerically generated data sources encompass medical and industrial computed tomography and algorithms such as Marching Cubes [23], which extracts isodensity surfaces from volume data. Tessellated domains are required for

engineering design, analysis, and simulation applications in fields such as computational fluid dynamics (CFD) and finite element analysis (FEA). Other potential applications for decimation algorithms include terrain modeling, seismic mapping, and meterological modeling.

An example of a surface reconstruction application is illustrated in Figure 1.1. The aircraft geometry was reconstructed by using a laser digitizer to scan an existing model. The point domain was subsequently tessellated and a virtual milling algorithm was applied to identify the actual surface [24]. This test geometry was utilized by Merriam [26] in an effort to demonstrate the potential of quickly completing 3D geometry input, mesh generation, flow solution, and analysis for computational fluid dynamics design applications. The target time frame for this complete design cycle is one continuous 24 hour period.

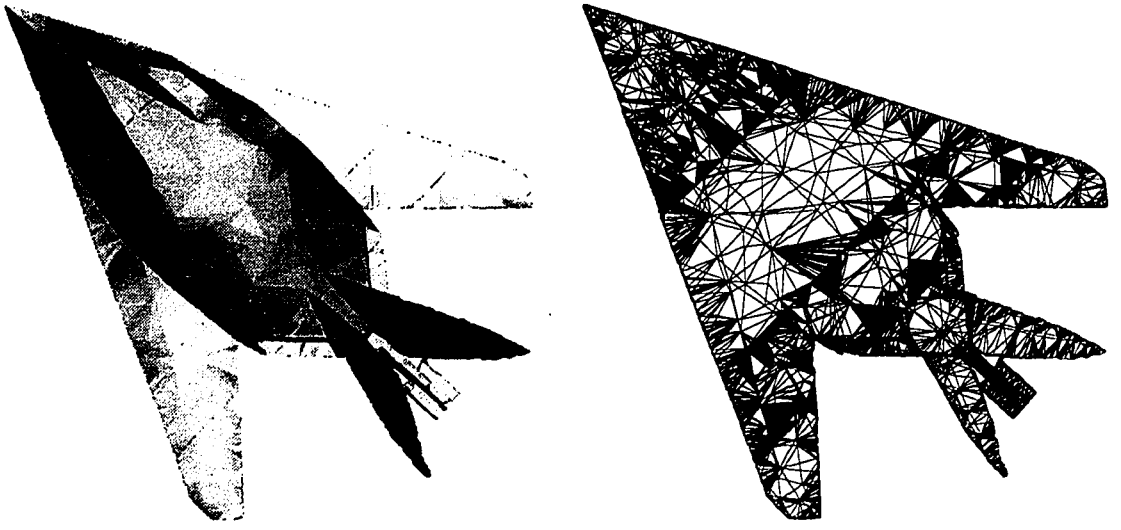


Figure 1.1: F-117 aircraft surface and sting, 30,925 vertices: shaded (left), wireframe (right)

This research builds on the recent work of Schroeder et al. [30], Turk [32], and Hoppe et al. [20] to generalize surface decimation for unstructured volume applications. Assuming no history information, the goal is to perform local, dynamic vertex removal from an unstructured tetrahedrization while preserving the initial tessellation topology and boundary geometry. The result is an original, robust, local, n-dimensional unstructured decimation algorithm.

Decimation Origins

According to Webster's Dictionary, the term *decimate* originally meant "to kill every tenth person," but has evolved to include "the destruction of any large proportion of a group." The choice of this term for use within the numerical discretization community seems natural given that structured approaches matured before unstructured methods. The simple, albeit naive, sub-sampling technique is consistent with Webster's definition of decimation and the structured mentality. The more general definition befits unstructured applications.

A brief survey of filter-based and adaptive methods with potential decimation ramifications is provided by Schroeder et al. [30]. Filter-based methods begin with a large number of primitives and remove or replace elements to decrease model size. Techniques include sub-sampling, which uses every n^{th} data point to reduce the set to manageable dimensions; and averaging, which resamples data using adjacent points. The latter method tends to distort high frequency features. Adaptive methods selectively resolve specific features of interest, eliminating primitives in certain "uninteresting" regions while inserting additional primitives in others. The volume of literature on adaptive methods is immense, encompassing octree methods, deformable models, fitting schemes, implicit modeling, and approximation methods.

Surface Decimation

Surface decimation algorithms typically utilize one of three geometric entities—vertices, edges, or faces—as the primitive element for removal. The choice of geometric primitive may be driven by the specific application or by the data representation scheme.

Schroeder et al. [30] present a vertex-based algorithm for decimating general surfaces. The method computes a local split-plane and uses a recursive loop-splitting procedure to re-triangulate the hole resulting from vertex removal. Multiple passes are made over all vertices in the set until user-specified tolerances defining the vertex distance to an average plane or vertex distance to an edge are satisfied. These decimation criteria generally yield crisp feature resolution, but may wash out gradual or soft features because the criteria are evaluated based on the current, not the original, state of the mesh.

Turk [32] describes a vertex-based surface tiling method that preserves the geometry and topology of the original model. The goal of Turk's research is to automatically generate polygonal models at various levels of detail for graphics rendering applications. The technique first uniformly distributes new vertices on an existing mesh and subsequently moves vertices via a

point repulsion computation based on distance and curvature. The global system connectivity and removal of the original vertices are computed by a series of local planar triangulations using a constrained greedy triangulation algorithm. The method is designed for use with general curved surfaces and therefore is ill-suited for use in resolving sharp features.

An edge-based polygonal reduction algorithm for general parametric surfaces is described by Khan [22]. The method sorts candidate edges by weighted surface curvature values and defines a crease angle criterion that each triangle in the final mesh must satisfy. A sequential quadratic programming maximin optimization algorithm is used to select a new vertex location within the feasible parametric region to replace the removed (collapsed) edge. The hole is then re-triangulated. Sharp feature resolution is attained by using knot vector information to define parametric subdomains. Subsequent recursive application of the reduction algorithm for each subdomain maintains critical boundary features.

Hamann [19] presents a triangle-based surface decimation algorithm for discrete data sets. Principal curvature values at each vertex are computed and used to weight each triangle. The minimum weight triangle is then removed, the hole is re-triangulated with the addition of a vertex, and the local weights are recomputed. The added vertex is chosen based on the construction of a local bivariate function that approximates the surface. The iterative procedure continues until the specified reduction percentage is attained.

Hoppe et al. [20] cast surface decimation as a mesh optimization problem in which an objective function based on the competing interests of data fidelity and conciseness of representation is minimized. The number of vertices, their positions, and the connectivity are varied during the optimization process. Although there is currently no provision that guarantees a global minimum, the method is superior to the aforementioned techniques in that the objective function directly measures deviation of the final mesh from the original. Furthermore the procedure is reported to concentrate vertices in regions of high Gaussian curvature, align edges along directions of low curvature, and recover sharp features.

Unstructured Mesh Generation

The CFD and FEA communities have published extensively on the general topics of structured and unstructured mesh generation [1, 2, 3, 4, 5]. The term decimation, however, is not widely used in these fields. Rather, the process of removing, repositioning, or adding points to refine domains of interest is generally described as *adaption*. As the cost of computer memory and processing time decreased, researchers modeled successively larger problems with respect

to both size and complexity. The mesh generation problem is typically approached from an increasing density perspective, i.e., an initial nominal vertex distribution is iteratively redistributed or augmented with additional points to resolve high gradient features of interest. The power of unstructured algorithms which intelligently add or remove points to resolve relevant physical features has been well established. However, current three-dimensional unstructured techniques require history information to redistribute or remove points. Since this research focuses on unstructured meshes, the scope of the literature reviewed is limited accordingly.

Briggs [8] provides an informative introduction to the multigrid strategy, which is used to solve a system of governing equations (for fluids, typically the mass, momentum, and energy equations) by successively iterating across a series of meshes of varying vertex density. Vertex densities vary by a constant factor, usually 4 for 2D and 8 for 3D. Starting the solution on a coarse grid is an inexpensive means to obtain a good initial guess for the next finer mesh. In addition, when smooth error modes dominate the fine grid solution (making the relaxation scheme ineffective), the current solution is simply interpolated onto a coarser grid. Consequently, smooth modes appear more oscillatory, yielding a more effective relaxation scheme. Concurrently, when the fine grid convergence rate deteriorates, the residual equation is used on a coarser grid to compute an approximation for the error that is subsequently used to correct the fine grid solution.

Mavriplis and Venkatakrishnan [25] present a timely review of unstructured multigrid methods in which "the generation of coarser (mesh) levels" is cited as the primary obstacle in the use of multigrid algorithms for unstructured flow solver applications. While structured multigrid methods use a sub-sampling approach to generate the required coarser structured grids from an initial fine grid, no trivial extension of this concept exists for unstructured methods. Consequently, unstructured multigrid researchers have attempted a variety of solutions to the coarse mesh generation problem. One such approach within the computational fluids discipline is *agglomeration*, which refers to the grouping of fine mesh control volumes to form larger coarse mesh cells.

Constrained Triangulations

Preparata and Shamos [28] state that, "In many cases the triangulation problem may be of a constrained nature, that is, a set of triangulation edges may be prespecified in the problem statement. Typically, this is the case when we are asked to triangulate the interior of a

simple polygon. The greedy method will succeed for such constrained problems, but it has the drawback that its performance is substantially far from optimal. On the other hand, there is no immediate way to adapt to this case the Delaunay triangulation method. So, a new technique has to be found." Although advances have been made in constrained Delaunay triangulation algorithms since this statement was published in 1985, it remains true for dimensions greater than two.

The greedy triangulation of a set of N points in two dimensions can be constructed in time $O(N^2 \log N)$. The algorithm constructs a list of all the candidate edges and subsequently inserts edges into the triangulation one-by-one, under the constraint that a new edge never crosses an existing one. The term "greedy" refers to the fact that the method never undoes what it did earlier. While the method is useful for solving constrained planar triangulation problems, it neither guarantees a unique solution nor naturally generalizes to higher dimensions.

A constrained Delaunay triangulation (CDT) forces specified edges into the triangulation and adds new edges based on specific visibility and Delaunay conditions. According to Bern and Eppstein [7], "the CDT contains the edge $\{a, b\}$ between two input vertices, if and only if a is visible to b and some circle through a and b contains no input point c, visible to segment ab, in its interior. Point a is said to be *visible* to point b if line segment ab does not cross any existing edge of the triangulation. Line segment ab may intersect an edge without crossing it, however. The concept of a constrained Delaunay triangulation is currently meaningful for two dimensions but ill-defined for higher ones.

A scheme to generalize the concept of constrained Delaunay triangulation to 3D surfaces is presented by Chew [11]. The method occasionally requires vertex deletion of non-source (initial constrained vertices or edge endpoints) vertices. The resulting hole is filled using a star-shaped triangulation algorithm. Chew has acknowledged the goal of performing 3D unstructured volume decimation independent of connectivity history via private communication [12]. Work toward this end has been focused on using an unconstrained algorithm to generate the required candidate void-filling tessellation. A proposed scheme to determine which tetrahedra to insert and in what order to insert them is based on projecting the candidate decimation vertex to the fourth dimension. An algorithm has yet to be implemented or rigorously tested.

Given an unstructured surface definition, the problem of generating a valid volume tessellation of the interior domain is non-trivial. The concept of a constrained triangulation is natural since some faces are specified and general nonconvex surfaces must be accomodated. The volume tessellation can be generated using the Tanemura/Merriam algorithm, which propagates

a front from the defined surfaces such that new tetrahedra satisfy the Delaunay circumsphere criteria. However, Barth [3, 4] cites two key problems associated with this method. First, the final mesh is dependent on the order of traversal of the surface faces, since the concept of a constrained 3D Delaunay tessellation is ill-defined [7]. Second, a nonconvex volume defined by a constrained face set is not guaranteed to be tetrahedralizable, unlike the planar counterpart [10]. An example of an untetrahedralizable polyhedron [6] is depicted in Figure 1.2. The final geometry is created from an initial triangular prism by fixing the base and twisting the top face with respect to it. This deformation creates three reflex edges and renders the resulting volume untetrahedralizable without the addition of Steiner points (additional interior vertices). Clearly the original triangular prism is tetrahedralizable, as illustrated in Figure 1.3.

Mesh generation methods are generally constrained to remove vertices, edges, or faces based on the insertion history, which is typically recorded in a tree or list structure. From the volume modeling and mesh generation arena, apparently no unstructured three-dimensional decimation algorithms exist that are independent of connectivity history information.

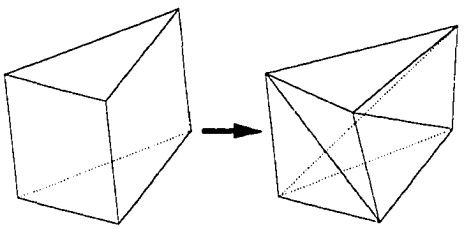


Figure 1.2: Construction of a twisted triangular prism (untetrahedralizable)

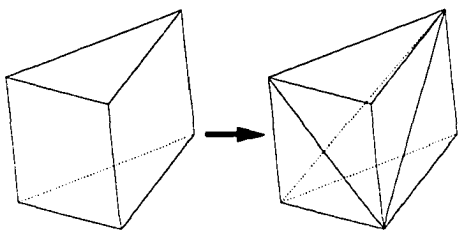


Figure 1.3: Initial triangular prism (tetrahedralizable)

Overview

The formulation, implemention, and testing of a general algorithm for decimating unstructured discretized data sets is presented. The discretized space may be a planar triangulation, a

general 3D surface triangulation, or a 3D tetrahedrization. The decimation algorithm enforces Dirichlet boundary conditions, uses only existing vertices, and assumes manifold geometry. Local dynamic vertex removal is performed without history information while preserving the initial topology and boundary geometry. The tessellation at each step of the algorithm is preserved and, in the pathological case, every interior vertex is a candidate for removal. The focus of this research is how to remove a vertex from an existing unstructured n-dimensional tessellation, not on the formulation of decimation criteria. The criteria for removing a candidate vertex may be based on geometric properties or any scalar governing function specific to the application.

A review of basic tessellation concepts and algorithms follows in Chapter Two. Chapter Three presents the core algorithm development for unconstrained planar tessellations. The development of the local tessellation and topological classification scheme to handle nonconvex local boundary regions is discussed. Chapter Four addresses the conservation of Delaunay properties. The projection and rotation computations used to extend the planar decimation algorithm to general three-dimensional surfaces are introduced in Chapter Five. The generality of the decimation algorithm is demonstrated in Chapter Six, which explores unconstrained volume decimation. Performance and implementation issues are presented in Chapter Seven. A discussion of the merits of the current research and avenues for further exploration is documented in Chapter Eight.

CHAPTER 2. BACKGROUND

Tessellation

A comprehensive introduction to the mathematical basis and diverse applications of spatial tessellations is presented by Okabe et al. [27], from which the following formal definition is provided. Let S be a closed subset of $\Re^m$ and $\mathcal{T} = \{S_1, \ldots, S_n\}$, where S_i is a closed subset of S. If elements in the set $\mathcal{T}$ satisfy

$$[S_i \setminus \partial S_i] \cap [S_j \setminus \partial S_j] = \emptyset, \qquad i \neq j, \qquad i, j \in I_n \ , \tag{2.1}$$

and

$$\bigcup_{i=1}^{n} S_i = S, \tag{2.2}$$

then the set $\mathcal{T}$ is called a *tessellation* of S.

Connectivity Scheme

Given an arbitrary set of points, a connectivity scheme introduces the concepts of edges, faces, and adjacency. An edge is created by connecting any two points; a face is defined by a closed loop of at least three edges; and adjacency refers to the immediate neighbors or incident elements at a given vertex. Most visual applications and conservative formulations impose the additional constraint that faces must be composed of simple non-overlapping polygons.

In general, data may be organized in a structured or unstructured sense. Contrary to an unstructured scheme, the vertex *degree* (refer to the **DEFINITIONS** section in the prelude for definitions of common terms) for a structured organization is predictable. Regardless, the points adjacent to a given point must be accessible. Adjacency facilitates the formulation and evaluation of integrals and derivatives required to model and solve the system governing equations. Structured data schemes are typically organized and stored in arrays. In one dimension, the adjacent points of the i^{th} point are simply the $(i+1)^{th}$ and $(i-1)^{th}$ positions. Unstructured data schemes are typically stored in linked list or tree structures. A pointer to the root node

is used to access the first value and a pointer(s) to the subsequent node, which in turn stores a value and a pointer(s) to the next node in the structure. The list or tree is terminated with a null pointer.

Triangulation

A triangulation is a connectivity scheme composed of triangular elements which tessellate a planar region or perhaps a three-dimensional surface. A tessellation refers to a general subdivision of a spatial domain, with no implied maximum dimension. Therefore a triangulation is also a tessellation. According to Preparata and Shamos [28], "a triangulation of a finite set S of points is obtained by joining the points of S by nonintersecting straight line segments so that every region internal to the convex hull of S is a triangle." Two examples of planar triangulations are shown in Figures 2.1 and 2.2. The term *uniform* unstructured tessellation is used in Figure 2.1 because each interior vertex is separated by a minimal distance, d, from every other vertex in the mesh. This constraint is not imposed in the random unstructured tessellation illustrated in Figure 2.2.

Delaunay Triangulation

A Delaunay triangulation (DT) enforces the additional constraint that the circumcircle uniquely defined by the three points of a face will contain no other points in the domain. Thus a Delaunay triangulation of an arbitrary point set guarantees a unique connectivity scheme, assuming 4 or more points are not co-circular. Added benefits from numerical, visual, and aesthetic perspectives are that the DT maximizes the minimum angle of the triangulation. Many of the mathematical properties of the planar DT extend naturally to higher-dimensional tessellations. Barth [5], Okabe et al. [27], and Bern and Eppstein [7] provide excellent reviews of Delaunay properties. An example of an unstructured planar Delaunay triangulation is shown in Figure 2.3.

The Concept of a Simplex

A simplex is a special type of convex hull. According to Okabe et al. [27], "A *simplex* in $\Re^m$ is the convex hull of any set of $m+1$ points which do not all lie on one hyperplane in $\Re^m$. For $m = 0$, the simplex is a point itself, called the zeroth-order simplex; for $m = 1$, the simplex is the straight line segment connecting the two points, called the first-order simplex;

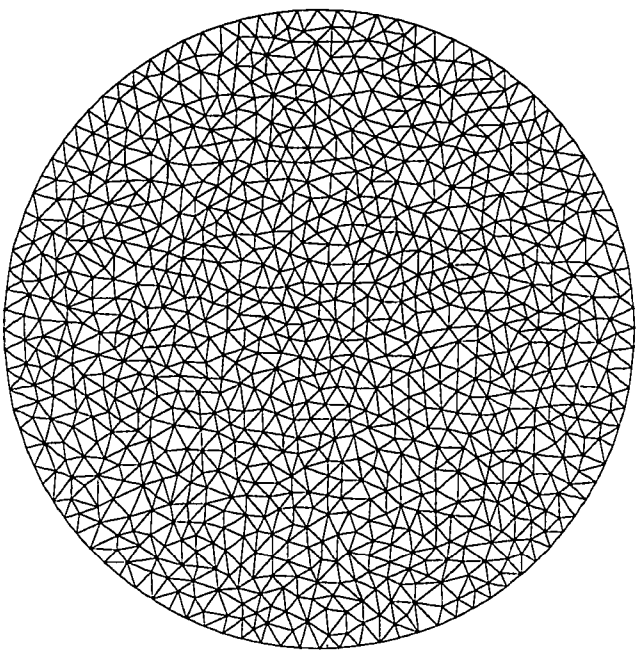


Figure 2.1: Uniform unstructured planar tessellation (1000 vertices, 1898 elements)

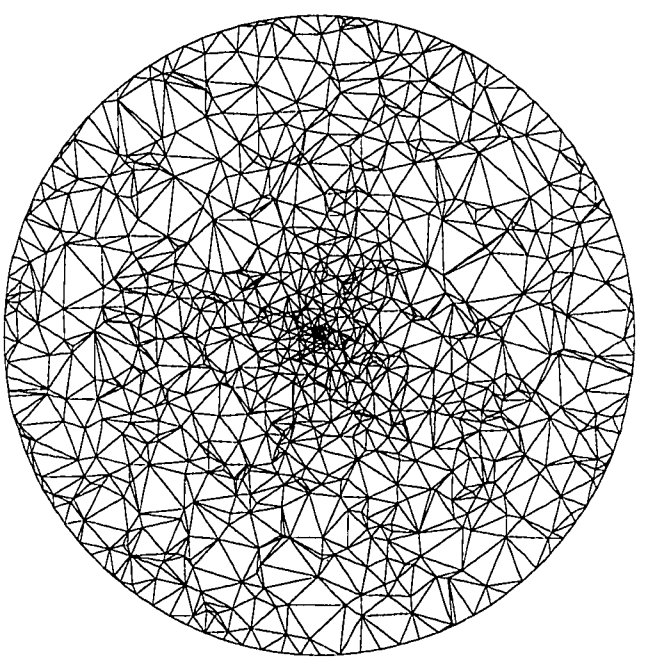


Figure 2.2: Random unstructured planar tessellation (1000 vertices, 1898 elements)

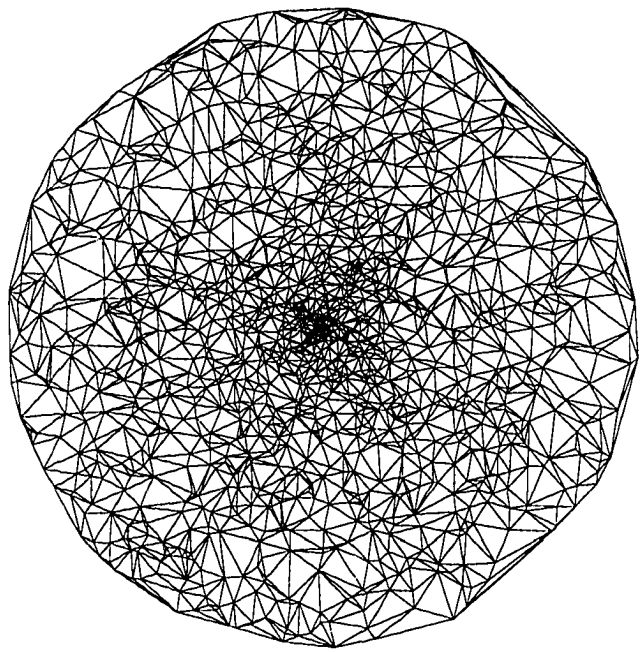


Figure 2.3: Unstructured planar Delaunay triangulation (2000 vertices)

for $m = 2$, the simplex is a triangle, called the second-order simplex; for $m = 3$, the simplex is a tetrahedron, called the third-order simplex, and so forth."

A first-order simplex may be expressed formally as

$$L = \{\mathbf{x} |\ \mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2, \quad 0 \leq \lambda \leq 1\} \tag{2.3}$$

where L is a straight line segment connecting two points p_1 and p_2, inclusive; $\mathbf{x}$ is a column vector; and λ is a scalar. The mth-order simplex is written using Equation 2.4, which defines a convex polygon A with vertices $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as

$$A = \left\{ \mathbf{x} | \sum_{i=1}^{n} \lambda_i \mathbf{x}_i \quad \text{where} \quad \sum_{i=1}^{n} \lambda_i = 1, \quad \lambda_i \geq 0, \quad i \in I_n \right\} \tag{2.4}$$

where $\mathbf{x}_2 - \mathbf{x}_m, \ldots, \mathbf{x}_{m+1} - \mathbf{x}_1$ are linearly independent.

Watson's Simplex Algorithm

The general tessellation algorithm used in this research was developed by Watson [33]. It finds the DT of a list of two- or three-component vertices and returns a list of simplex vertex indices with the corresponding circumcenter and squared radius. The method can also be used to

find the ordered convex hull of a set of two- or three-component points. The algorithm therefore is suited to generate planar triangulations or volume tessellations (tetrahedrizations). Consistent with Watson's terminology, an n-simplex defines a triangle for $n = 2$ and a tetrahedron for $n = 3$.

Polygon Classification

The decimation algorithm development depends on familiarity with elementary computational geometry definitions for polygons and polyhedra. In particular, the concepts of a simple polygon, monotonicity, and convexity are critical.

Simple Monotone Polygon

A polygon is *simple* if there is no pair of nonconsecutive edges sharing a point. Conversely, a non-simple polygon contains self-intersecting edges. A simple polygon is said to be *monotone* if its boundary can be decomposed into two chains monotone with respect to the same straight line. Trivial examples of a simple monotone and a non-simple polygon are presented in Figure 2.4.

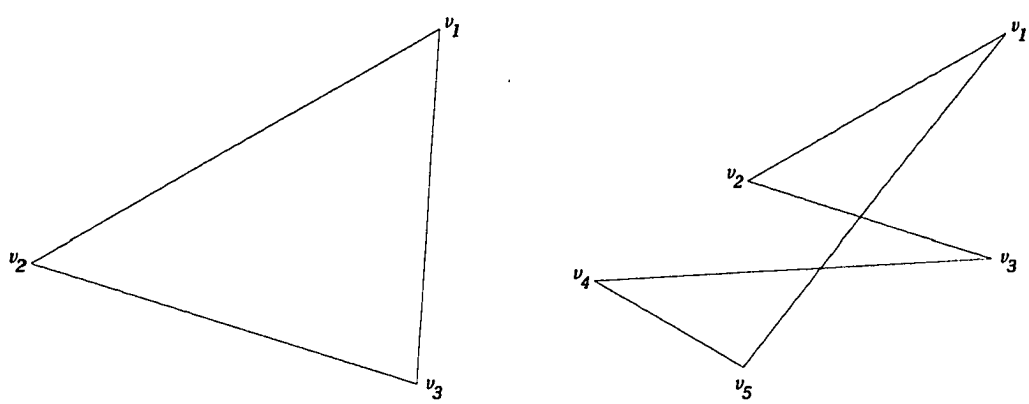


Figure 2.4: Simple monotone polygon (left) and non-simple polygon (right)

Convex and Monotone Polygon

According to Preparata and Shamos [28], a domain D in E^d is *convex* if, for any two points q_1 and q_2 in D, the segment $\overline{q_1 q_2}$ is completely contained in D. From an informal viewpoint in two dimensions, a convex polygon has a boundary which could be traversed in an orderly

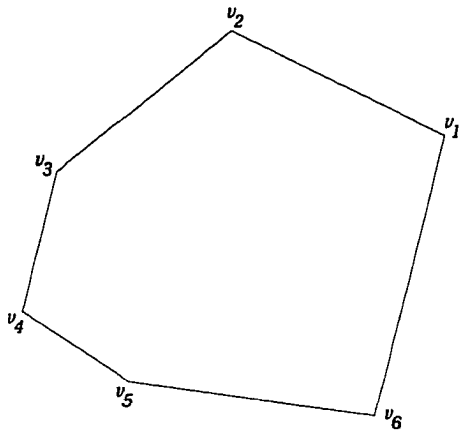


Figure 2.5: Simple convex, monotone polygon

counterclockwise direction by taking exclusive left-hand turns. A polygon is defined to be *convex* if its interior is a *convex set.* A simple, convex, monotone polygon is depicted in Figure 2.5.

Nonconvex and Monotone Polygon

A simple, nonconvex, monotone polygon is shown in Figure 2.6.

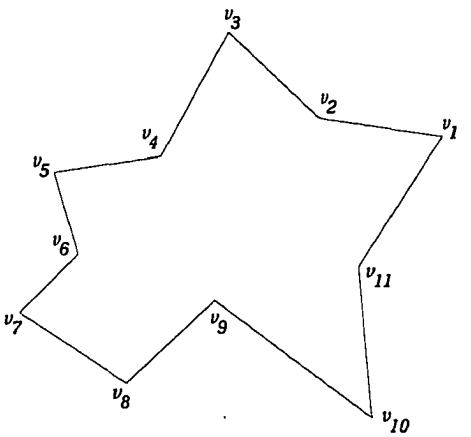


Figure 2.6: Simple nonconvex, monotone polygon

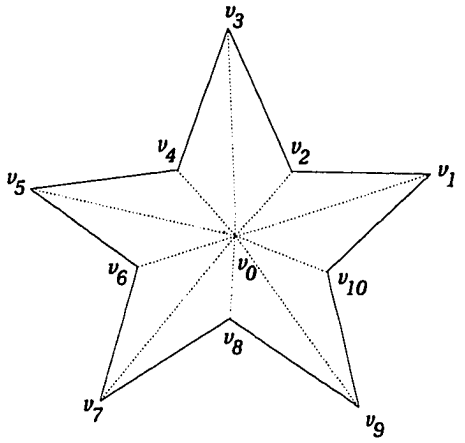


Figure 2.7: Simple star-shaped polygon

Star-Shaped Polygon

A simple polygon P is *star-shaped* if there exists a point z not external to P such that for all points p of P the line segment $\overline{zp}$ lies entirely within P. Figure 2.7 illustrates a simple star-shaped polygon.

Nonconvex and Non-Monotone Polygon

An example of a simple, nonconvex, non-monotone polygon appears in Figure 2.8.

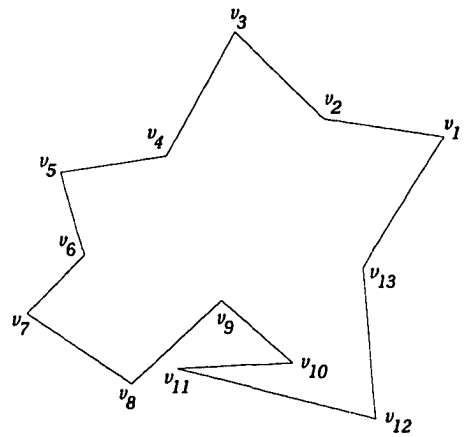


Figure 2.8: Simple nonconvex, non-monotone polygon

Surface Boundary Vertex Identification

For general surface triangulations, interior vertices can be distinguished from boundary vertices by comparing the number of triangles incident at a vertex to the number of adjacent vertices. For interior vertices, the number of incident triangles equals the number of adjacent vertices. In contrast, the number of adjacent vertices corresponding to a given boundary vertex will be one greater than the number of incident triangles. An illustration of this simple classification scheme is provided in Figure 2.9. Clearly vertex v_0 has 6 adjacent vertices (v_1–v_6), and 6 incident triangles (T_0–T_5). However, vertex v_1 has 3 adjacent vertices (v_2, v_0, and v_6) but only two incident triangles (T_0 and T_1).

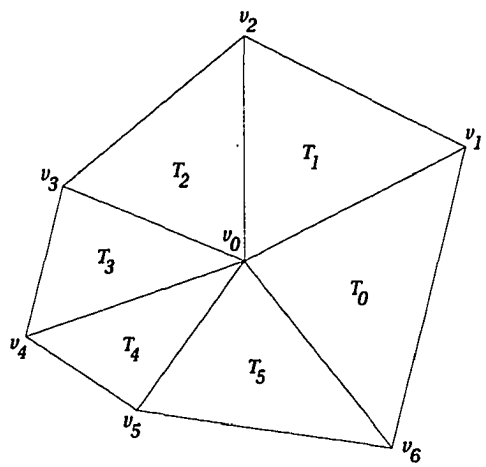


Figure 2.9: Automatic classification of interior and boundary vertices

Degenerate Geometry and Topology

Problems may arise in a given tessellation due to the presence of coincident physical vertices, repeated vertex indices, non-simple geometry, or inconsistent topology (e.g., dangling edges or faces). An example of inconsistent topology is a surface that contains an edge with more than two incident faces. Degenerate volumes, faces, or edges may be manifest by the existence of zero volume, area, or length values, respectively.

All geometry for the current research is preprocessed in an attempt to eliminate input degeneracies. Degenerate cases created by coincident physical vertices and repeated vertex indices are recognized and simplified. Vertices defining colinear or coplanar degenerate geometries are retained. Existing inconsistent topology is also preserved in the final decimation result.

Degenerate triangles are composed of 3 coincident vertices, 2 coincident vertices (collapsed edge), or 3 colinear vertices. Degenerate tetrahedra consist of 4 coincident vertices, 3 coincident vertices, 2 coincident vertices (collapsed edge), 2 pairs of coincident vertices (collapsed edge), 4 colinear vertices, 3 colinear vertices, or 4 coplanar vertices.

Pathological Decimation Constraint

For a given tessellation, the vertex decimation criteria are defined to enable pathological extremes for robustness evaluation. The conditions are:

1. Each boundary vertex is retained
2. Each interior vertex is a candidate for removal

Thus all non-boundary vertices may be deleted while the tessellation is maintained at each step. The severity of the pathological decimation criteria will be illustrated in the application sections that follow. Less severe decimation criteria are typically employed in practical applications.

The current research focuses on how to remove a vertex from an existing unstructured n-dimensional tessellation, not on the formulation of decimation criteria. Criteria for removing a candidate vertex may be based on geometric properties or any scalar governing function specific to the application. For example, the criteria for removing a candidate planar vertex might be a function of area, aspect ratio, or gradient information. General surface decimation rules might be based on surface curvature, area, aspect ratio, or gradient information. Aspect ratio, solid angles, gradient information, or volume could be used to formulate volume decimation constraints.

CHAPTER 3. PLANAR DECIMATION

The general decimation algorithm is conceptually simple. First, identify a candidate vertex and its adjacent points. Second, tessellate the polygonal region defined by the adjacent vertices (local boundary loop) using any suitable algorithm. Third, replace the original n-simplices with the new local tessellation and delete the candidate vertex. Since deletion of a vertex removes all of its incident n-simplices, a hole is created that must be tessellated. In two dimensions, this hole defines a planar polygon which may be classified as either convex and monotone, nonconvex and monotone, or nonconvex and non-monotone.

Planar Hole Tessellation

The number of triangles remaining after the removal of an interior vertex and insertion of the new local triangulation may be computed by means of Euler's formula [28]. For a planar subdivision,

$$V - E + F = 2 \tag{3.1}$$

where V is the number of vertices, E is the number of edges, and F is the number of faces. The simplest possible polygon is a triangle, where $V = 3$, $E = 3$, and $F = 2$. Note that the number of faces must include the face exterior to the point hull of interest.

In general, when an interior vertex is removed, the number of vertices is reduced by one and the number of edges must decrease by at least three. Using the $'$-notation to denote the state after an interior vertex is removed,

$$V' - E' + F' = 2 \tag{3.2}$$

where

$$V' = V - 1 \qquad E' = E - 3 \tag{3.3}$$

Substitution of Eqn. 3.3 into Eqn. 3.2 yields

$$(V - 1) - (E - 3) + F' = 2 \tag{3.4}$$

Making use of Eqn. 3.1, the new number of faces, F' is

$$F' = E - V = F - 2 \tag{3.5}$$

Therefore the deletion of an interior vertex must remove exactly two triangles. Figure 3.1 illustrates the initial and final triangulations, respectively, corresponding to the removal of interior vertex v_0.

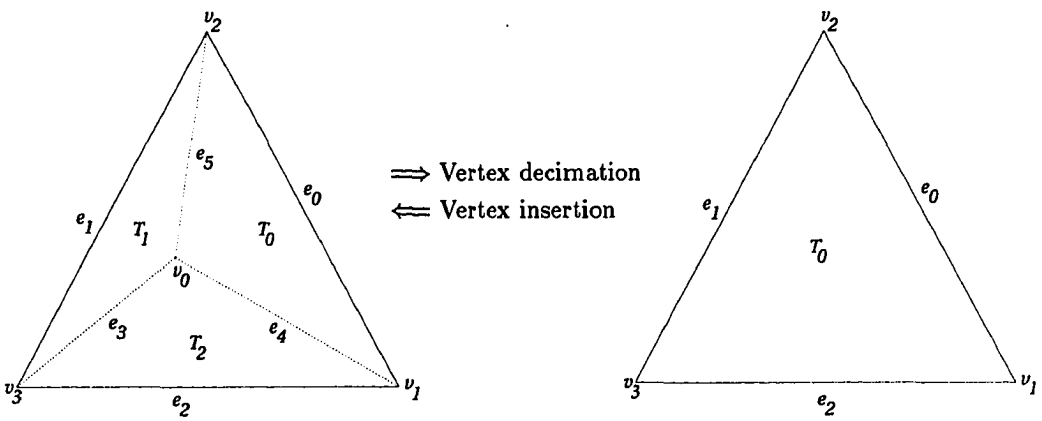


Figure 3.1: Removal of interior vertex, v_0

Local Triangulation

If restricted to the class of planar or general surface problems, the new local triangulation may be computed by several methods. For strictly convex polygons, the problem is trivial. Triangles may be formed by choosing any boundary vertex and connecting it to every other boundary vertex, excluding its two adjacent neighbors. Thus each new triangle is incident at the starting node. Examples of two distinct valid local triangulations for candidate decimation vertex, v_0, are illustrated in Figure 3.2. This method requires the boundary vertices to be ordered and yields a non-unique solution for the given vertex set (i.e., the set of triangles is dependent on the starting node).

Preparata and Shamos [28] present a general algorithm to triangulate a simple, monotone polygon in $O(N)$ time. A sorted list of vertices is required, which requires $O(NlogN)$ time. Furthermore, the resulting triangulation is not guaranteed to be Delaunay.

Watson [33] developed an incremental unconstrained Delaunay tessellation (DT) scheme based on the observation that every vertex inserted into an existing DT must be visible to the adjacent vertices defining the local boundary loop. Removal of a vertex from a planar

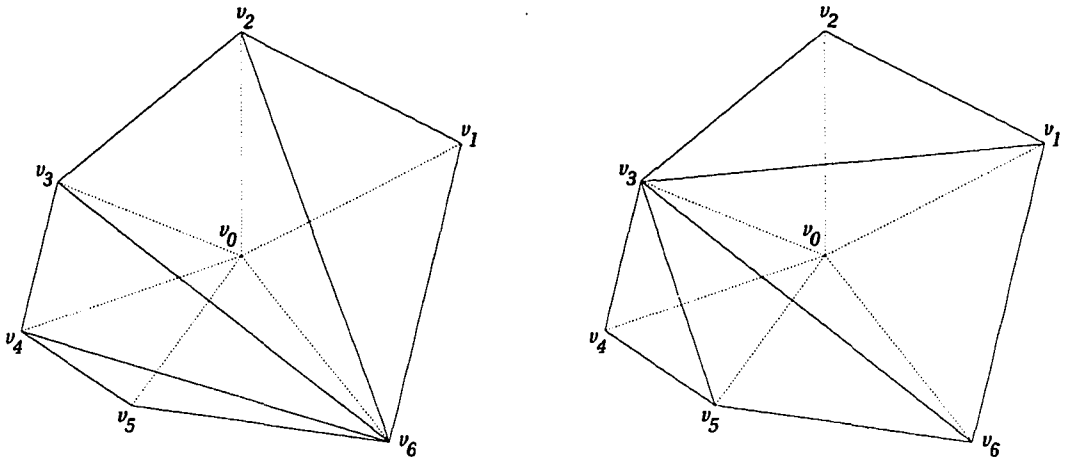


Figure 3.2: Two simple convex local triangulations

DT therefore guarantees a star-shaped (nonconvex, monotone) polygon triangulation problem. The selection of any existing algorithm for triangulating star-shaped polygons, or perhaps a constrained triangulation method (e.g., a greedy triangulation algorithm), would be sufficient to solve the 2D local tessellation problem. However, none of these methods generalize to higher dimensions.

So why not simply use a constrained Delaunay triangulation (CDT) algorithm? Bern and Eppstein [7] state that, "There does not seem to be a reasonable definition of a constrained Delaunay tessellation in three dimensions."

The choice of a general unconstrained tessellation algorithm (preferably Delaunay) is motivated by the goal of solving the volume decimation problem. No three-dimensional equivalent to the two-dimensional concept of clockwise ordering exists. Consequently, a simple algorithm to tessellate a star-shaped polyhedron does not exist.

This obstacle is overcome by applying a general algorithm to generate a local unconstrained Delaunay tessellation. Although this approach generalizes to higher dimensions, the resulting local tessellation is always convex. If the initial local boundary loop is nonconvex, some of the resulting n-simplices will intersect valid n-simplices external to the local boundary loop, violating the global tessellation. As illustrated in Figure 3.3, inserting all triangles returned by the local unconstrained Delaunay tessellation algorithm for the local boundary loop defined by vertices 2, 10, 9, 13, 17, 20, and 5 would cause triangle 2-9-10 to violate the global tessellation properties. This introduces an n-simplex classification problem to address nonconvex regions.

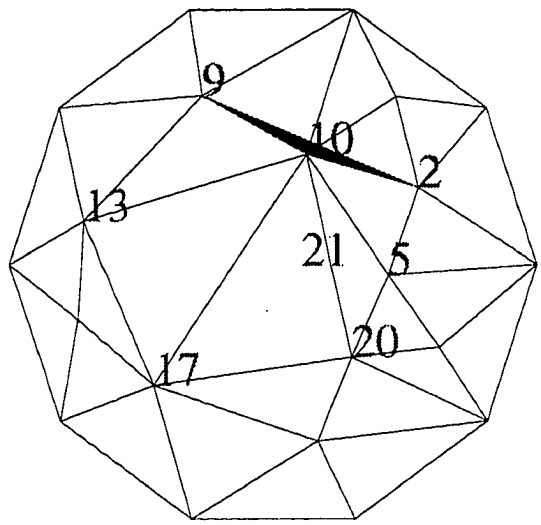


Figure 3.3: Unconstrained Delaunay tessellation local classification problem: triangle 2-9-10 violates the global tessellation

n-Simplex Classification

This general class of problems is referred to as the point-location problem. Preparata and Shamos [28] present Theorem 2.1 which states: "Whether a point z is internal to a simple N-gon P can be determined in $O(N)$ time, without preprocessing."

The polygon inclusion problem (i.e., classification of a point as interior or exterior with respect to a nonconvex polygonal boundary) may be solved by one of two methods [28]: angle summation or ray-intersection. Angle summation accumulates the angles subtended between the point in question and all sets of two consecutive boundary vertices. This method requires an ordered boundary and may be restricted to 2D problems.

A more general point location algorithm is based on the number of times a ray originating at the point intersects the polygon. An odd number of intersections indicates that the point is inside the loop while an even number corresponds to a position external to the loop. This method does not require an ordered boundary and is applicable to both 2D and 3D. However, degenerate cases exist (e.g., if a ray is tangent to a boundary edge).

The alternative method proposed utilizes a general unconstrained local Delaunay tessellation. A topology preserving post-processing scheme is developed to classify valid n-simplices.

Data Structures

An arbitrary collection of vertices is the minimal input required by the decimation algorithm. If not specified, the connectivity may be computed using any general tessellation algorithm. In the current implementation, an unconstrained Delaunay tessellation algorithm will return the connectivity corresponding to a convex tessellated domain. Nonconvex geometry and topology with holes can be specified with explicit vertex and initial connectivity information.

Input Requirements:

1. An unordered list of vertex coordinates, **or**
2. An IGES NURBS surface definition, **or**
3. An unordered list of vertex coordinates **and** an unordered list of triangle/tetrahedron vertex indices. In this case, the connectivity data is generated by an external source (e.g., a CAD/CAM program).

Storage Requirements:

1. Vertex coordinate data requires a vector of length $3N$ (N vertices, 3-components per vertex)
2. Connectivity information requires a vector of length $(M \cdot m)$ (M triangles, m vertex indices per element)
3. Internal data structures:
   (a) For each vertex, the corresponding list of adjacent vertices (N^2 maximum)
   (b) For each vertex, a list of incident elements ($N \cdot M$ maximum)
   (c) A list of boundary vertices (N maximum)

The following data structures are currently used in the implementation of the algorithm: binary tree, simple linked list, stack, queue, and mathematical set. These structures were designed to foster intuitive access to tessellation information, with the explicit mission of demonstrating the feasibility of the proposed decimation algorithm—not necessarily to provide minimal storage or most efficient access.

Planar Decimation Algorithm

The general decimation algorithm for planar tessellations is based on the following constraints:

1. Boundary vertices and vertices of degenerate elements are retained. Interior vertices are candidates for removal
2. Pathological constraint to demonstrate method and numerical robustness

Algorithm:

Given a planar tessellation, for each vertex, v, in the candidate decimation list:

1. Remove v
2. Apply an unconstrained tessellation algorithm to the adjacent vertices defining the generally nonconvex local boundary loop
3. Classify the new triangles as either interior or exterior with respect to the original local boundary loop
4. Remove the original triangles incident to vertex v
5. Insert the valid triangles identified in Step 3

Begin by creating an unconstrained Delaunay triangulation over the set of adjacent vertices that define the local boundary loop (hole boundary). Each adjacent vertex is initially connected to the candidate decimation vertex, v, by an edge. Since the convex hull of the local triangulation may not coincide with the local boundary loop, determine which candidate triangles lie inside this boundary. Begin by identifying interior triangles that are adjacent to portions of the convex hull and coincident with the original local boundary loop. From this set, determine the remaining interior triangles by successively marking triangles that neighbor triangles know to be interior.

Planar Local Tessellation Algorithm

Given that the decimation criteria have been satisfied for the candidate vertex, the following algorithm is applied to tessellate the hole created by its removal.

1. Identify and store the local boundary loop edges defined by the vertices adjacent to the candidate vertex.

2. Input the list of adjacent vertices into an unconstrained tessellation algorithm and return the new local connectivity.
3. For each new triangle
   - Assign a unique identifier to each of its edges
   - Record the number of times a boundary loop edge is shared by the triangle
4. Test for the failure case—at least one original boundary edge does not exist in the new local tessellation. If true, the candidate vertex cannot be removed without violating the tessellation topology. In this case exit and proceed to the next candidate vertex.
5. Initialize the set of valid edges, which contains the original boundary edge set and/or edges belonging to valid interior triangles.
6. **Classification—Phase 1:** Sort the new triangles onto one of three stacks (**Valid, Interior**, or **Exterior**) based on their unique edge identifiers and the original boundary edge set.

   If the triangle exclusively shares an original boundary edge, the triangle must exist in the interior of the local boundary loop. Therefore remove all common edges from the set of valid edges. Subsequently add remaining edges to the set of valid edges. Then push the triangle index onto the **Valid Stack.** Otherwise push the triangle onto one of two stacks:

   - **Interior Stack:** no triangle edges match an original boundary edge.
   - **Exterior Stack:** one or more triangle edges match an original boundary edge.
7. Construct the set of valid interior edges by extracting the set of boundary edges from the set of valid edges.
8. **Classification—Phase 2:** Process the **Interior** and **Exterior** stacks in order while either of these stack dimensions remain dynamic.

   Pop a triangle from the current stack. If the intersection of its edge set and the valid interior edge set exists, remove all common edges from the set of valid interior edges. Subsequently add remaining edges to the set of valid interior edges. Then push the triangle onto the **Valid Stack.** Otherwise queue (insert) the triangle on the bottom of the stack. Upon convergence, exit.

9. Pop each triangle off the **Valid Stack** and insert it into the hole to preserve the original topology and boundary geometry.

Illustrations of the **Valid, Interior,** and **Exterior** stack contents corresponding to Phase 1 and Phase 2 of the classification algorithm are presented in Figure 3.4 for a nonconvex planar local boundary loop. In this example the local boundary loop encloses triangles labeled 0, 1, 2, 4, and 5. Unshaded triangles correspond to membership on the **Valid Stack**, lightly shaded triangles denote placement on the **Interior Stack**, and darker triangles indicate **Exterior Stack** membership.

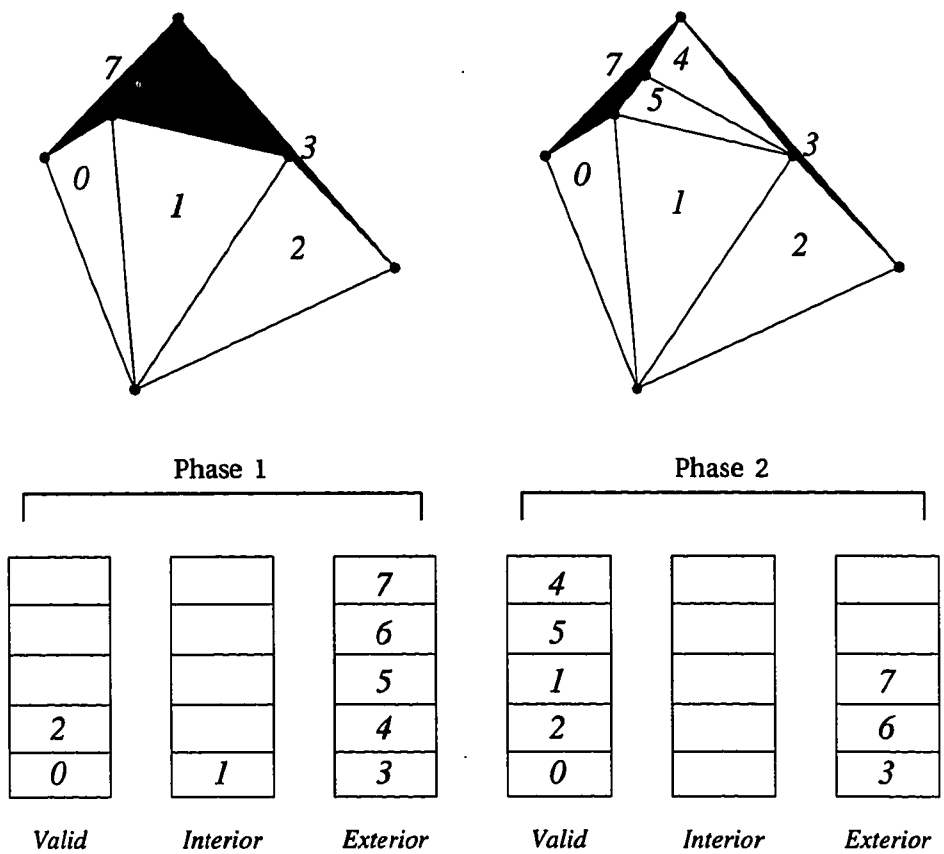


Figure 3.4: Stack contents corresponding to classification Phase 1 and Phase 2

Sample planar geometry and the resulting Phase 1 and Phase 2 classification results follow in Figures 3.5–3.9. Shaded triangles denote rejected elements for the given stage. Examples of four successful Phase 1 classification cases are illustrated in Figures 3.5 and 3.6 for convex

and nonconvex local boundary loops, respectively. The common denominator among these geometries is the fact that each interior triangle exclusively shares at least one original boundary loop edge. Although the Phase 1 algorithm is sufficient to classify these cases, it does not generalize to more complex cases.

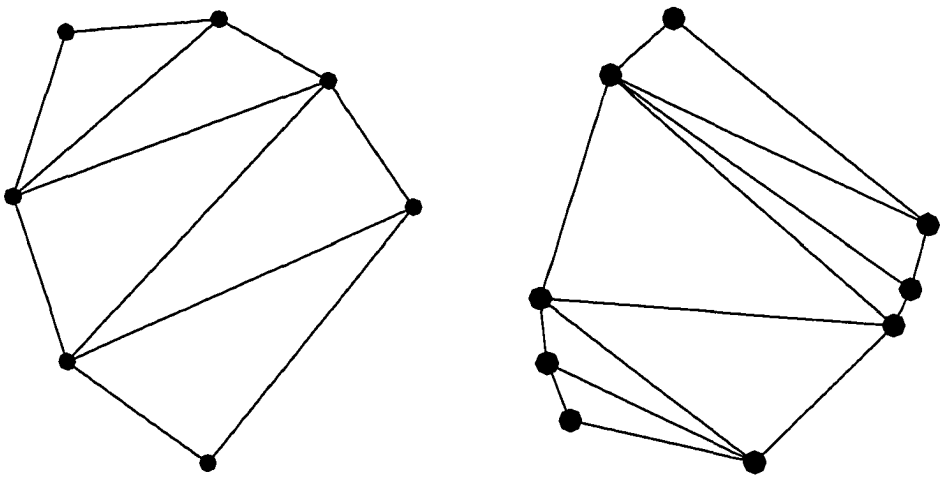


Figure 3.5: Two distinct convex loops: Phase 1 successful

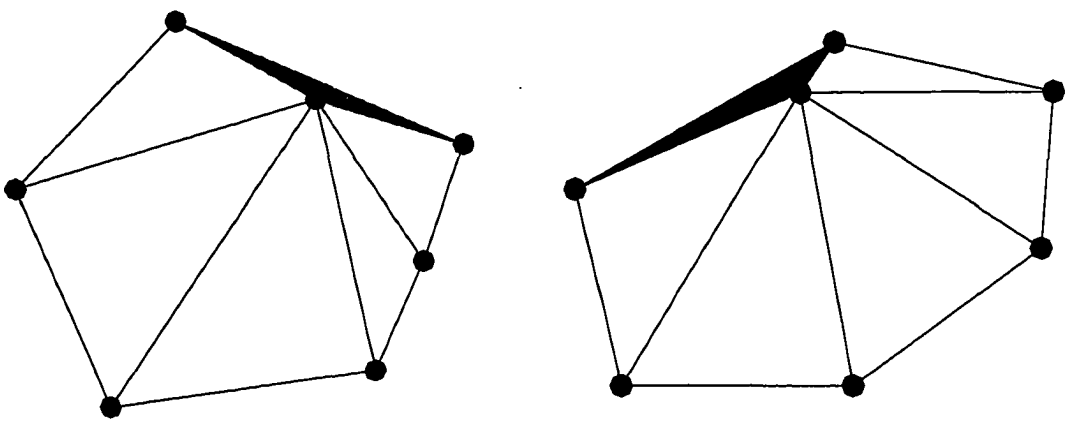


Figure 3.6: Two distinct nonconvex loops: Phase 1 successful

Complex convex and nonconvex geometry requires the use of Phase 2 of the classification algorithm. The general geometries illustrated in Figures 3.7–3.9 document the robustness of the two-phase classification algorithm. The intermediate results following the Phase 1 classification are presented on the left and the corresponding final Phase 2 solution appears on the right.

The two configurations shown in Figure 3.7 demonstrate one limitation of the Phase 1 classification. Triangles that share no original boundary edges are indeterminate with respect

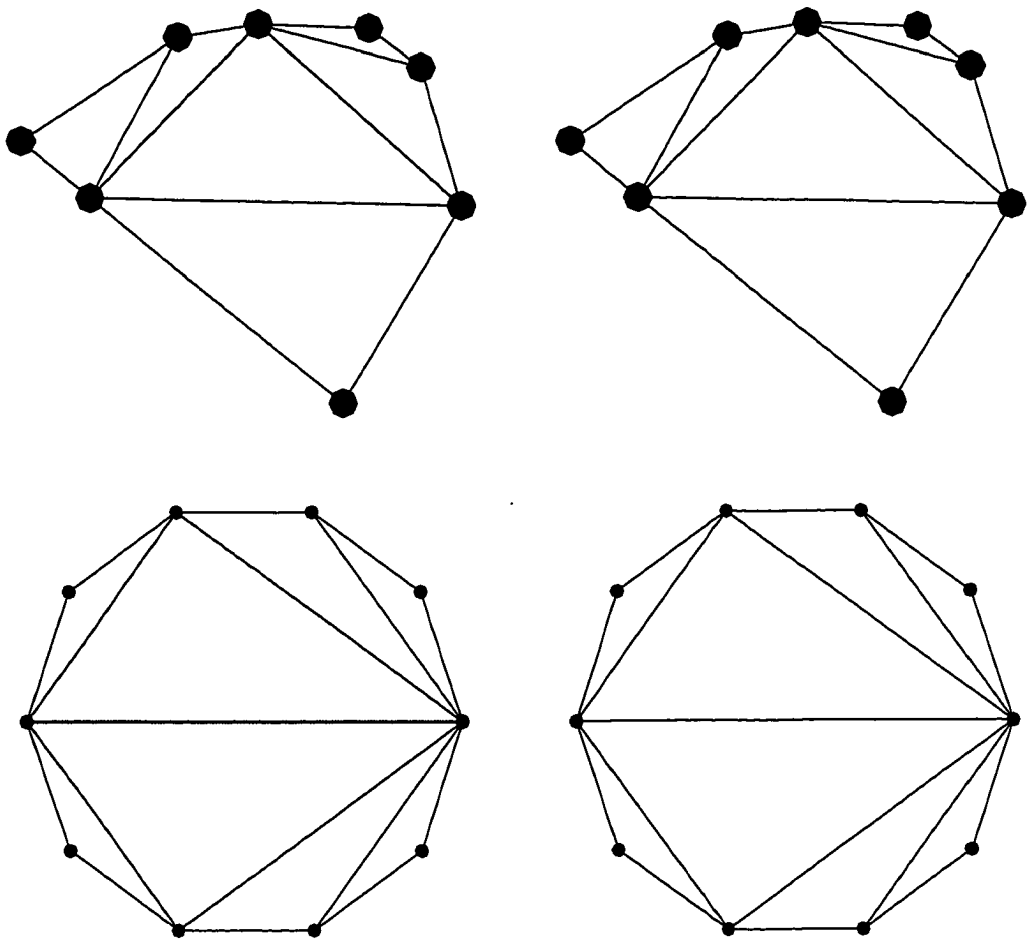


Figure 3.7: Two complex convex loops: Phase 1 (left), Phase 2 successful (right)

to **Valid Stack** membership at this stage. Placement of these elements on the **Interior Stack** is optimistic in the sense that actual membership is probably on the **Valid Stack**. The assumption that **Interior Stack** elements lie in the interior of the boundary is leveraged by processing the **Interior Stack** prior to the **Exterior Stack** in Phase 2 of the algorithm. However, it is possible for a triangle to lie outside the original boundary loop and not contain any boundary edges. An example of this infrequent result is presented in Figure 3.8. The multitude of long, narrow triangles conforms to the nonconvex boundary defined by a portion of a circular arc. Lightly-shaded triangles to the right of the arc are exterior to the original boundary loop, and therefore rejected.

Another shortcoming of the Phase 1 classification is that triangles that do not exclusively share at least one original boundary edge are indeterminate. These darkly-shaded elements are pushed on the **Exterior Stack** for Phase 2 consideration. Two complex nonconvex local boundary loop examples are presented in Figures 3.8 and 3.9.

Failure Cases

The local tessellation algorithm fails if an *exclusively shared* original boundary loop edge does not exist in the candidate local tessellation. This is not an anomaly! However, it is not a critical problem even for the pathological decimation scheme. The candidate decimation vertex is simply appended to the end of the list and processed later. This is advantageous because the adjacent vertex and incident triangle lists for the rejected vertex change the instant one of its adjacent vertices is decimated, automatically renewing its decimation candidacy.

A trivial example of a simple nonconvex failure case is depicted in Figure 3.10. Since the Delaunay tessellation algorithm is guaranteed to return a candidate local tessellation bounded by its convex hull, Phase 1 of the classification algorithm will fail. In particular, the **Valid Stack** will be empty after Phase 1 because no exclusively shared original boundary edges can exist in the new local tessellation.

The solution of the trivial star-shaped failure case could be explicitly handled by a subprocess check. If the **Valid Stack** is empty after completion of Phase 1 of the classification algorithm, compute each candidate triangle centroid location and use a general ray-intersection algorithm to determine if the triangle lies inside or outside the original local boundary loop. The amount of effort introduced is probably not justified, since the observed frequency of this failure case is minute. Moreover, the additional topological checks required to combat ray-intersection degenerate cases are not warranted.

By definition, the candidate decimation vertex must be visible to each of its adjacent vertices. This conclusion is trivial because the candidate decimation vertex lies within the local boundary loop, which is always star-shaped. This initial tessellation constraint is enforced at every step of the decimation algorithm.

Conditions that exist to produce a failure case may, however, be due to degenerate geometry. This scenario exists in two dimensions when the candidate decimation vertex is virtually colinear with a boundary loop edge. An example of this condition for the 2,000 vertex mesh is shown in Figure 3.11. The candidate decimation vertex is vertex 864. The corresponding local boundary loop is defined by vertices 1628, 1056, 827, 1089, 723, 1950, and 1641. Vertices 864,

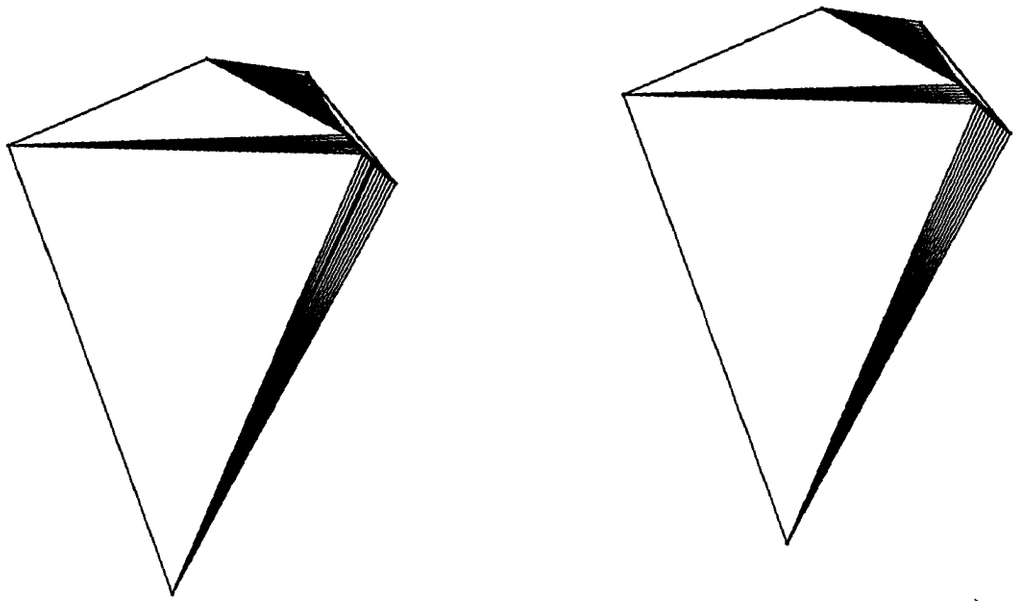


Figure 3.8: Complex nonconvex: Phase 1 (left), Phase 2 successful (right)

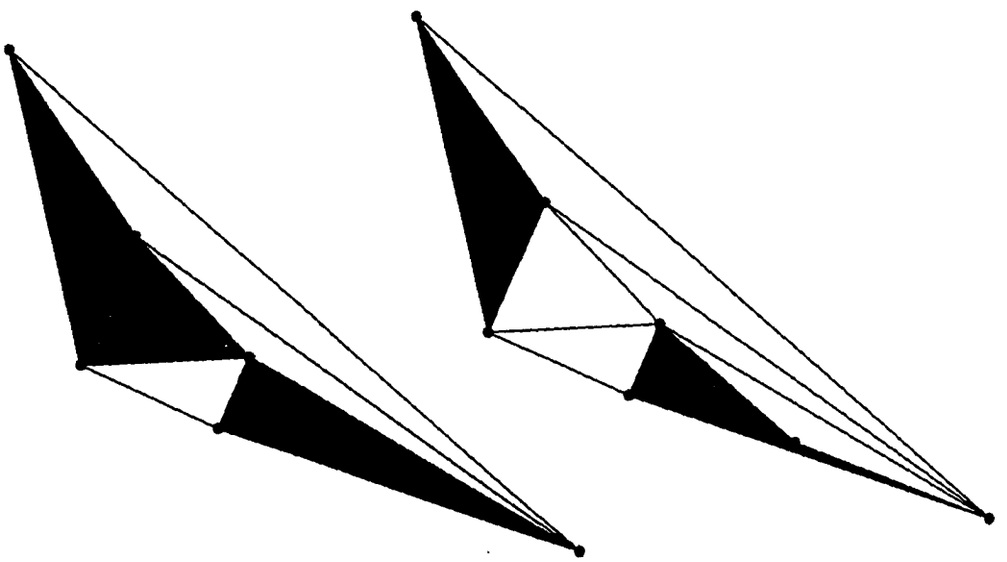


Figure 3.9: Complex nonconvex: Phase 1 (left), Phase 2 successful (right)

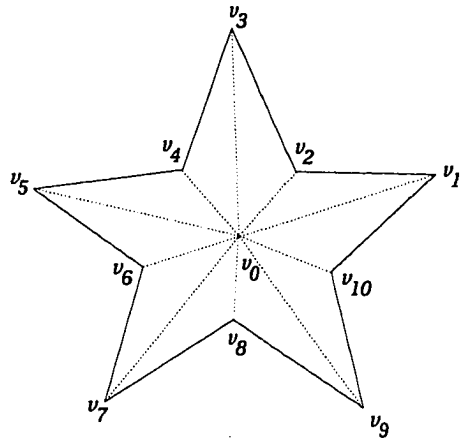


Figure 3.10: Simple star-shaped polygon failure case

1628, and 1641 define a nearly colinear (degenerate) triangle. In this example, the numerical tolerance associated with the Delaunay tessellation algorithm floating point computations is large enough to return a nonconvex tessellation. That is, the nearly colinear triangle defined by vertices 1950, 1628, and 1641 is not defined in the new local tessellation. As a result, the original local boundary loop is not preserved and candidate vertex 864 must be rejected at this step.

Unique Hashing Function

The computational efficiency of the current algorithm relies on the notion of a unique hashing function. A hashing function accepts an input variable(s) and returns a key which may or may not be unique. Construction of a hashing function which returns a unique key is, in general, a non-trivial task.

For planar or general surface applications, the algorithm utilizes a unique key corresponding to each local boundary edge to enable topological classification with respect to the new set of triangle edges.

> **Problem 1** *For any combination of two unique integer indices, return a unique integer key.*

Intuition promised that a one-to-one mapping could be constructed with the aid of a list of prime numbers. A unique prime number was associated with each vertex in the current local

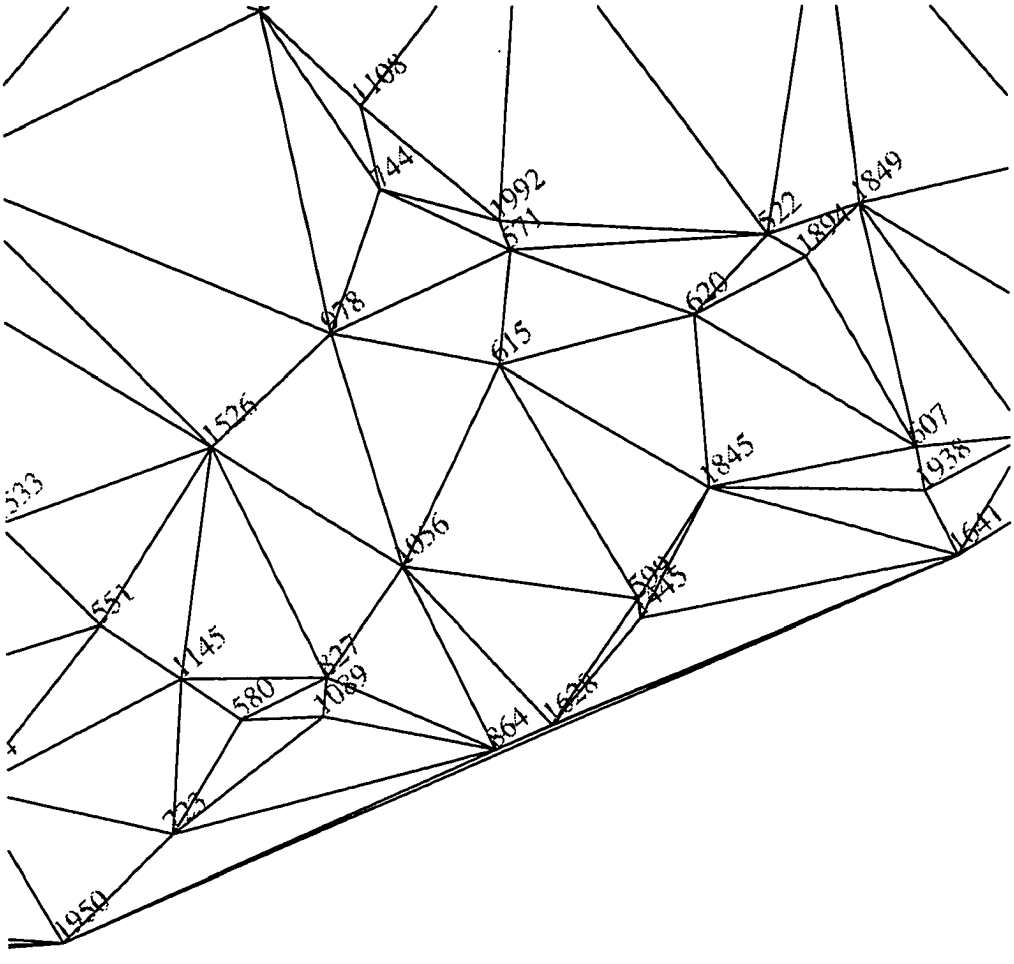


Figure 3.11: Nearly colinear boundary and candidate decimation vertex geometry

boundary loop as

$$P(v_i) = Prime\ Number[i] \tag{3.6}$$

Given a unique set of vertex indices and a one-to-one mapping to a list of prime numbers, a unique edge index corresponding to any two vertices results from the product of the two prime numbers. Thus,

$$H_2(v_1, v_2) = P(v_1) * P(v_2) \tag{3.7}$$

Implementation of the prime number-based hashing function is trivial. However, artificial constraints on the local decimation problem size are imposed by this technique. The problem stems from the fact that the size of an unsigned integer is restricted by the physical hardware design. Assume that the largest integer that can be represented is $x = 2^{32}$. Then $x^{\frac{1}{2}}$ and $x^{\frac{1}{3}}$

represent the largest prime numbers allowed in the hashing function table for general surface and volume decimation applications, respectively. This is true because each edge is uniquely identified by the product of two prime numbers. Similarly, each face is uniquely represented by the product of three prime numbers. Table 3.1 documents constraints on the maximum number of adjacent vertices permitted in the local boundary loop for planar, surface, and volume decimation applications.

Table 3.1: Artificial constraints imposed by prime number-based hashing function

Decimation Type	Largest Prime Number	Maximum Adjacent Vertices
Planar	$< 65,536$	6,544
Surface	$< 65,536$	6,544
Volume	$< 1,625$	259

A general solution to the hashing function problem can be implemented using a bit shifting method. An edge can be identified by concatenating its two unique vertex indices. The problem of presentation order of the vertex indices is overcome by sorting them in ascending order. That is, given edge, $e(v_1, v_2)$, where v_1 and v_2 are the vertex indices defining edge e, construct a unique edge identifier $H_3(v_1, v_2)$ as

$$H_3(v_1, v_2) = \begin{cases} v_1 \parallel v_2 & \text{if } v_1 \leq v_2 \\ v_2 \parallel v_1 & \text{otherwise} \end{cases}$$

Robust Example

A zoomed-in view of an unstructured mesh (from Figure 2.3) useful for testing the robustness of the planar decimation algorithm is illustrated in Figure 3.12. The mesh was generated from a random collection of planar points with an imposed dense, circular array of vertices near the center. The initial tessellation is composed of 2000 total vertices, where 34 vertices lie on the boundary, and 500 vertices reside on an interior circular feature. The maximum number of triangles incident at a given vertex in the initial tessellation is 72. The intermediate and final pathological decimation results are presented in Figures 3.13 and 3.14.

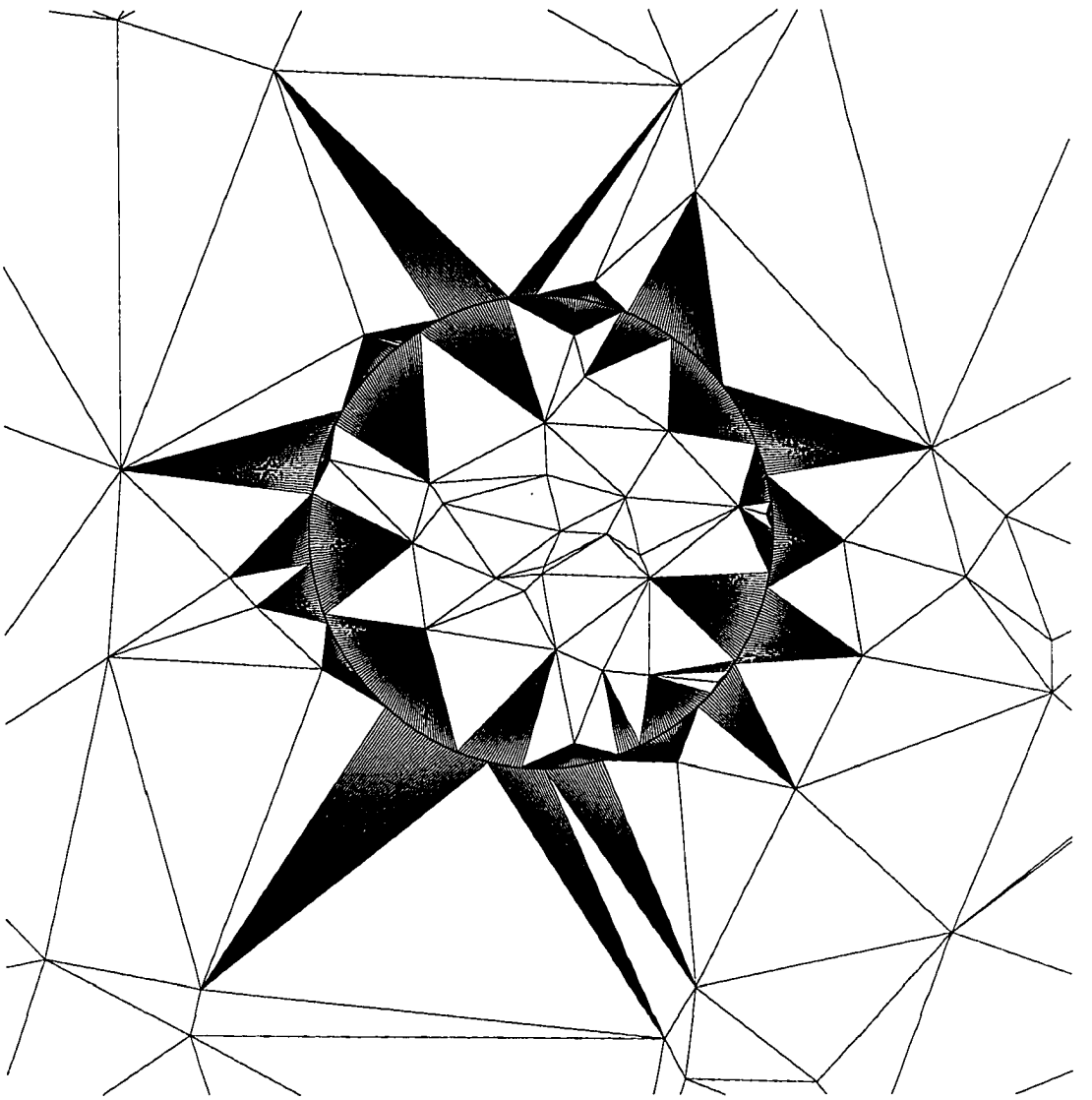


Figure 3.12: Initial robust planar unstructured mesh (partial view, 2000 vertices)

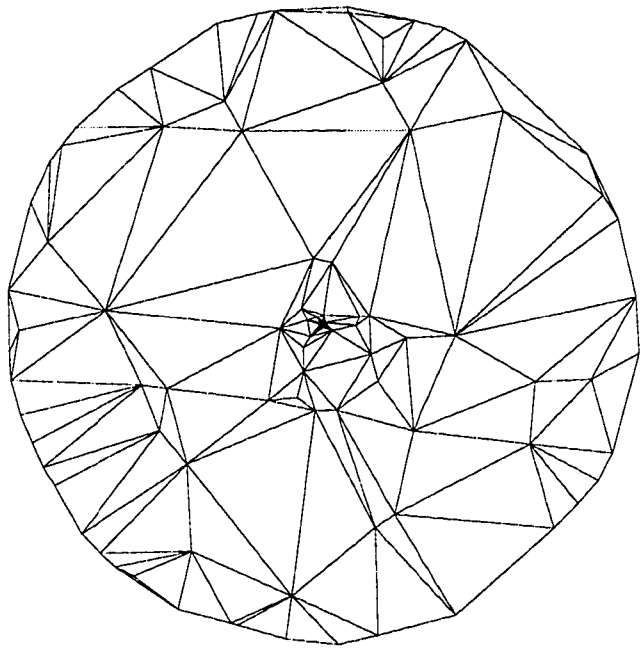


Figure 3.13: Robust planar unstructured mesh: 75% decimated

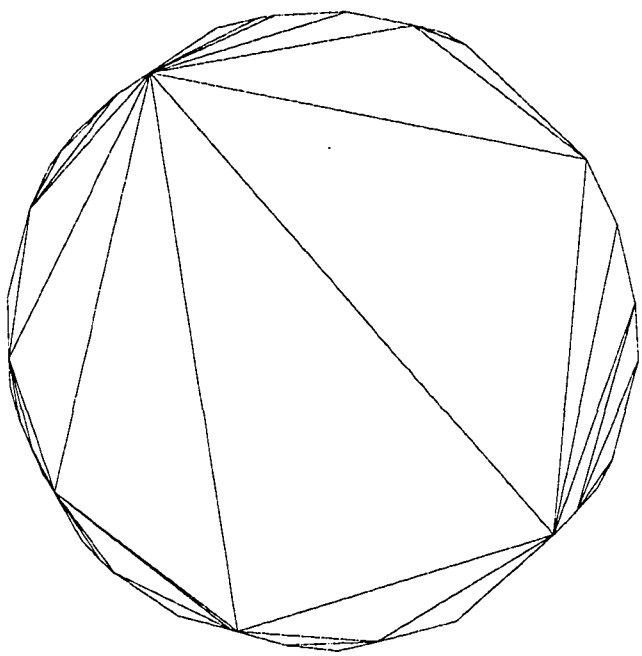


Figure 3.14: Robust planar unstructured mesh: interior 100% decimated

CHAPTER 4. CONSERVATION OF DELAUNAY PROPERTIES

Background

The hole left by vertex removal is filled using a general unconstrained Delaunay tessellation algorithm. The choice of Delaunay algorithms is arbitrary. It may be based on an edge-swapping [4], incremental [9, 33], advancing front [31], or divide and conquer [15, 28] scheme. However, the Delaunay scheme should be general enough to yield n-dimensional tessellations of the input vertex set to permit widespread decimation applications.

This basic procedure begs the following question: *"What is the state of the mesh with respect to global Delaunay properties as each vertex is removed if the initial tessellation is Delaunay?"* If the mesh remains globally Delaunay at every decimation step, the ramifications are clear. A decimation algorithm could be designed to remove multiple vertices per step. In theory, if the tessellation of an interior piecewise-linear boundary containing no embedded points depends only on the boundary vertices, all adjacent points within a closed boundary could be removed simultaneously. Furthermore, if the Delaunay properties are preserved, bi-directional mesh adaption (i.e., more coarse or refined) could be performed using the current decimation algorithm in conjunction with an incremental Delaunay insertion algorithm. Criteria to optimize the number of triangles or tetrahedra could then be formulated.

Conjecture

"Given an initial Delaunay tessellation, the global Delaunay properties are maintained throughout the series of local tessellations since no new points are added during the decimation process."

Experimental and intuitive evidence supports this conjecture. A graphic proof that utilizes a Voronoi diagram (VD), Delaunay tessellation (DT), or convex hull (CH) seems feasible. This problem might be approached from a geometric viewpoint by observing changes in the Voronoi diagram when a vertex is removed. The Voronoi diagram and the Delaunay tessel-

lation are duals. Simple examples for regular geometries—an equilateral triangle, a square, and a hexagon—are illustrated in Figures 4.1–4.3. The decimated configuration appears on the right in each case. Solid lines represent the Delaunay triangulation, dotted lines denote the corresponding triangle circumcircles, and dashed lines identify the Voronoi diagram. The dot-dashed lines demark the original regular polygon circumscribing circle, which is coincident with the set of overlapping Delaunay circumcircles in the decimated figure.

Visualize the change in the Voronoi diagram (dashed line) between the left and right images caused by the removal of the center vertex in the left figure. When the vertex is deleted, changes in the Voronoi diagram are local. That is, the Voronoi cell corresponding to the removed vertex simply collapses to a point or to an edge. The dual of the modified Voronoi diagram yields the new local Delaunay triangulation.

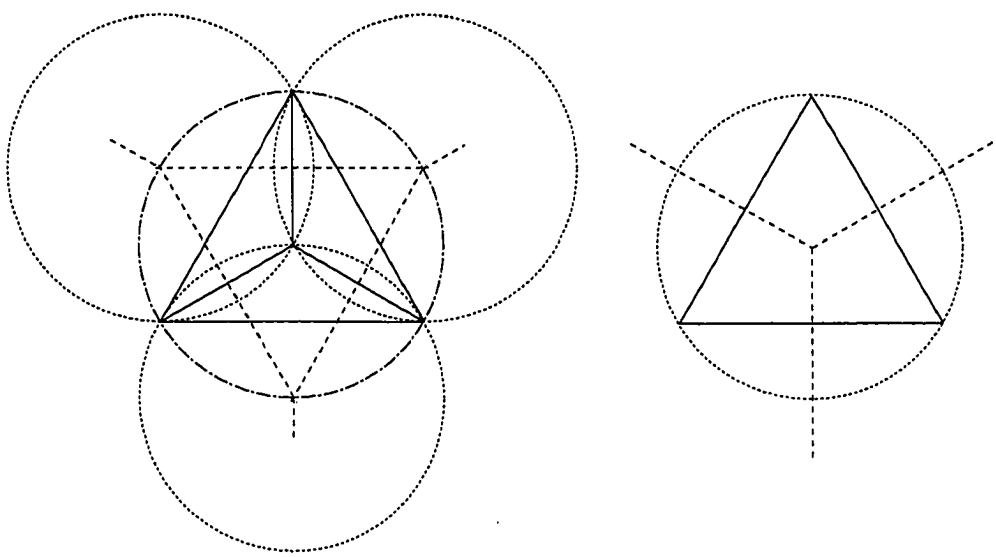


Figure 4.1: Voronoi diagram change for a trivial decimation step (equilateral triangle)

It is difficult to imagine local vertex insertion and deletion in terms of the Delaunay triangulation or the Voronoi diagram. Perhaps it is easier to see how the convex hull should be dynamically updated after vertex insertion or deletion. The Delaunay triangulation in n dimensions is closely related to the convex hull computation in $(n+1)$ dimensions. In fact the latter algorithm can be used to compute the former [28]. Given the existence of algorithms for the online update of a convex hull [28], a proof should be straightforward either through the convex hull in 4D or a direct proof in 3D.

Although Preparata and Shamos detail a 2D algorithm for dynamic maintenance of a convex hull and outline an n-dimensional algorithm for on-line construction of convex hulls, no

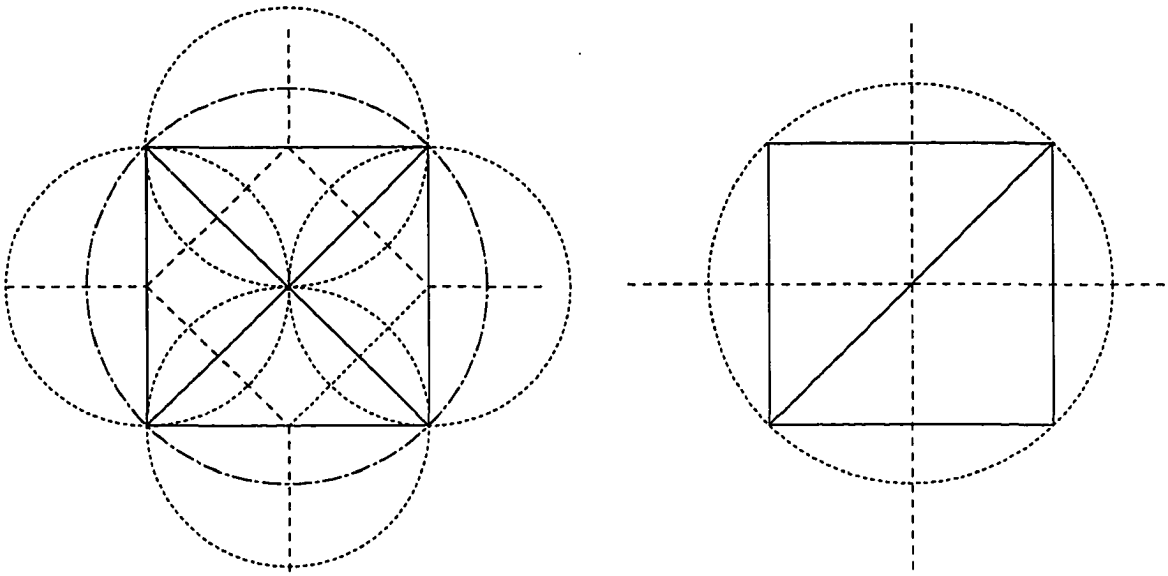


Figure 4.2: Voronoi diagram change for a simple decimation step (square)

general method for dynamic maintenance is presented. In this context, on-line refers to insertion only. There are many papers dealing with insertion, but deletion seems to be ill-documented.

Problem Statement: *Given a Delaunay tessellation, for an arbitrary interior vertex v, show that the global tessellation remains Delaunay if the points adjacent to v are used to generate a local Delaunay tessellation whose triangles are inserted into the hole generated by removing v.* Assume that both convex and nonconvex local boundary loops are tessellated using an unconstrained Delaunay tessellation algorithm. External $(n-1)$-simplices for local nonconvex boundaries are discarded via the local tessellation classification algorithm.

Incremental Insertion Perspective

From the viewpoint of local insertion inversion, observe that insertion is localized to triangles having circumcircles that contain the insertion point. In addition, the new connectivity affects only edges within the boundary defined by the triangles with non-empty circumcircles. Therefore, deletion of a vertex, v, should constrain connectivity changes to the interior region defined by v's adjacent vertices.

Experimental Perspective

Experimental evidence—obtained by checking if any existing vertices lie within the circumcircle of any of the insertion triangles (an equivalent Delaunay generation criterion)—supports

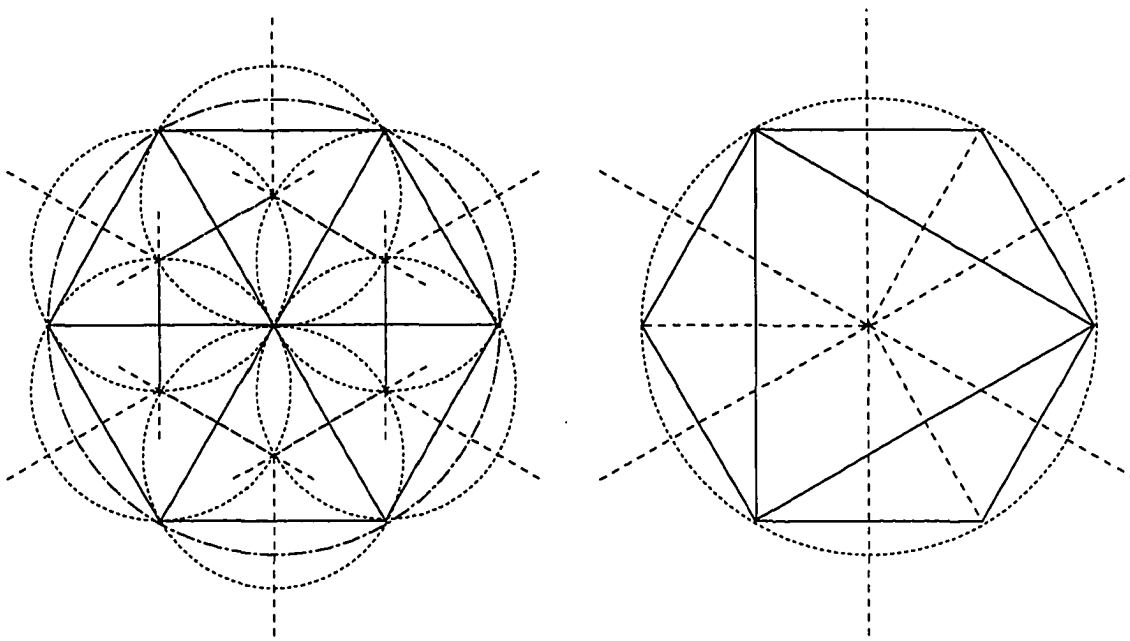


Figure 4.3: Voronoi diagram change for a simple decimation step (hexagon)

the original conjecture. The existence of an experimental failure case makes the pursuit of a proof useless, unless the geometry is degenerate. An experimental failure case was not discovered after deleting more than $185k$ vertices from randomly generated initial unstructured meshes. Both planar and volume meshes were tested, with the largest mesh size consisting of $100k$ points.

Graphical inspection of the original and replacement local connectivity was used to check the algorithm. Examples of connectivity interrogation are illustrated in Figures 4.4 and 4.5. The overall unstructured mesh is depicted as a black wireframe. The circumcircle and corresponding circumcenter for each new local tessellation triangle is denoted by a lightly shaded circle and small shaded sphere, respectively. Observe that if a circumcircle contains at least one vertex, the associated local insertion triangle is marked for discard (shaded gray), as expected.

Proof

First consider the set of triangles external to the local boundary loop. Because each triangle is a member of the initial DT, each must satisfy the well-known circumcircle criteria [6]. Bern and Eppstein [7] describe the edge circle property and circumcircle criteria: "If a and b are input points, the DT contains the edge $\{a, b\}$ if and only if there is a circle through a and b that intersects no other input points and contains no input points on its interior. Moreover, each

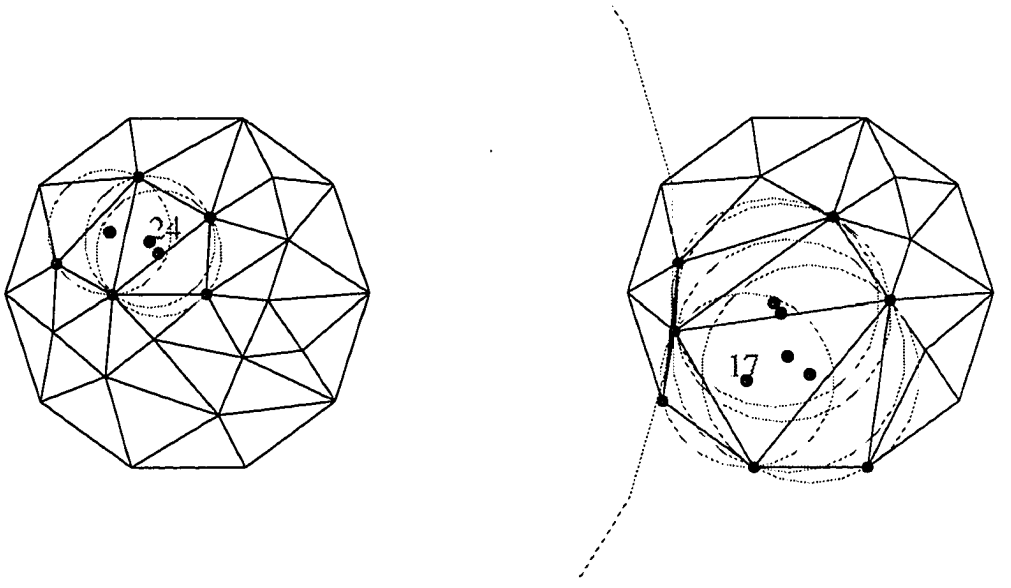


Figure 4.4: Delaunay circumcircles for local tessellation insertion check

circumscribing circle (*circumcircle*) of a DT triangle contains no input points on its interior."

The deletion of v does not impact the connectivity of any triangles exterior to the local boundary loop. Each triangle circumcircle was originally empty (by definition) and must remain empty since no new points are introduced. Therefore the set of triangles external to the local boundary loop exist in the global DT for the reduced set of vertices.

Next examine the local boundary loop edges with respect to the edge circle property. For each boundary edge, a circle exists that passes through the edge endpoints and neither intersects nor contains any input points. This circle is simply the circumcircle of the triangle external to the local boundary loop that shares the edge.

Now consider the valid triangles (triangles which tessellate the local boundary loop interior) returned from the local unconstrained DT algorithm. Each triangle is defined by exactly three local boundary loop vertices. No general adjacency conditions relating these three vertices can be formulated. In a two-dimensional context, adjacency refers to vertices positioned topologically "next" to one another when marching along the boundary in an ordered counterclockwise direction. Cases exist when all three triangle vertices are adjacent, when two vertices are adjacent, and when none of the vertices are adjacent.

With respect to the local boundary vertices, the valid set of triangles returned from the tessellation classificaton algorithm is Delaunay. That is, the circumcircle corresponding to each valid local triangle neither intersects nor contains any local boundary vertex. Furthermore,

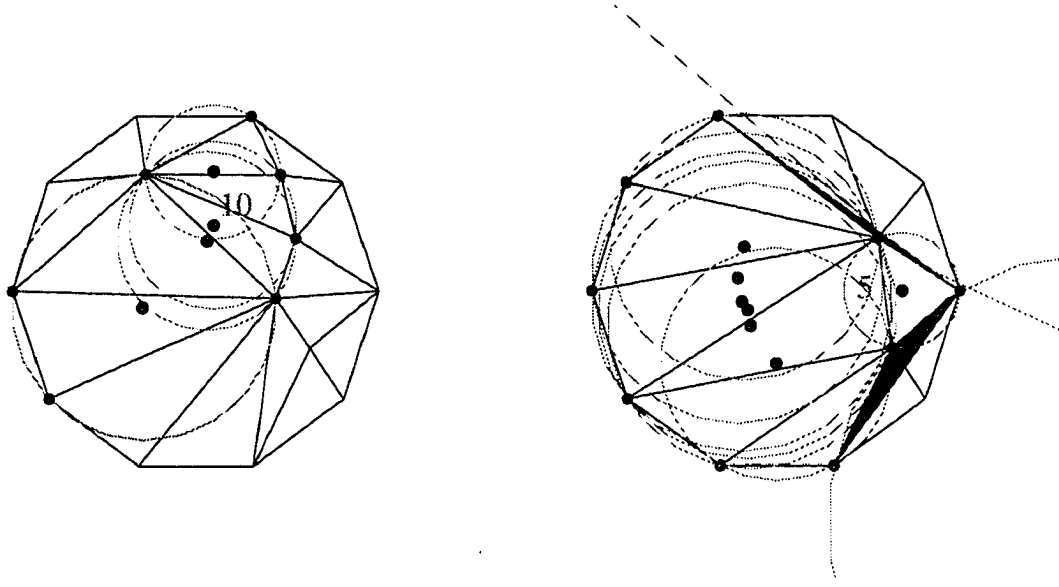


Figure 4.5: Delaunay circumcircles for local tessellation insertion check

since no vertices exist within the local boundary loop or on the local boundary edges, only vertices exterior to the local boundary loop must be considered to complete the proof.

Critical Question: *"Is it possible for a circumcircle of a valid local triangle to contain a vertex exterior to the local boundary loop?"*

Suppose the local tessellation algorithm generates a triangle with a circumcircle that encloses a vertex external to the local boundary loop. Then a new local tessellation of the boundary loop vertices must exist since the current tessellation is not globally Delaunay. However, the existence of such a tessellation violates the uniqueness condition of the initial local insertion connectivity. Recall that the initial local tessellation was generated using an unconstrained Delaunay algorithm. Producing a unique tessellation of the interior from global considerations or local considerations must yield the same result since no triangle outside the local boundary loop is permitted to change. This is sufficient to guarantee that the global mesh remains Delaunay after each decimation step, assuming the initial mesh is Delaunay and degenerate cases (e.g., four co-circular vertices in 2D or five co-spherical vertices in 3D) do not exist.

CHAPTER 5. GENERAL 3D SURFACE DECIMATION

A general 3D surface decimation algorithm is developed based on the new planar decimation algorithm. Area-weighted normals of the triangles incident at the candidate decimation vertex are used to compute a local average plane. Then the 3D surface points defining the local boundary loop are projected onto the average plane allowing use of the local planar decimation algorithm described earlier. Since only the connectivity is of interest and vertex positions on the local boundary loop remain fixed, the inverse transformation is unnecessary.

Projection Overview

The general surface decimation algorithm is based on a local projection transformation from three-dimensional space to a two-dimensional plane. Subsequent application of the planar decimation algorithm yields the desired candidate tessellation for the local surface patch.

Projection Plane Definition: The local projection plane and the distance to plane criterion were defined according to the equations described by Schroeder et al. [30]. An average plane with normal, $\bar{n}$, is constructed using information from vertices adjacent to the candidate decimation vertex. Specifically, triangle normals, $\vec{n}_i$, centers, $\vec{x}_i$, and areas, A_i , are used to define

$$\vec{N} = \frac{\sum \vec{n}_i A_i}{\sum A_i} \quad , \quad \bar{n} = \frac{\vec{N}}{\|\vec{N}\|} \quad , \quad \bar{x} = \frac{\sum \vec{x}_i A_i}{\sum A_i} \tag{5.1}$$

where the summation is over all triangles in the local boundary loop.

Decimation Criterion for Interior Vertices

The distance from the candidate decimation vertex, $\vec{v}$, to the average plane is

$$d = |\bar{n} \cdot (\vec{v} - \vec{x})| \tag{5.2}$$

If the distance, d, is less than a user-specified constraint, the vertex, $\vec{v}$, will be removed. Otherwise the vertex will be retained.

Point Projection

Begin with the normal form of the general plane equation,

$$Ax + By + Cz = D \tag{5.3}$$

where A, B, and C are the respective components of the plane unit normal and D is a constant determined by the plane unit normal, $\bar{n}$, and any point that lies on the plane. Next describe a general vector in parametric form,

$$\begin{aligned} x(t) &= x_0 + At \\ y(t) &= y_0 + Bt \\ z(t) &= z_0 + Ct \end{aligned} \tag{5.4}$$

Substitution of Eqn. 5.4 in Eqn. 5.3 yields

$$A(x_0 + At) + B(y_0 + Bt) + C(z_0 + Ct) = D \tag{5.5}$$

Solving for the general parameter, t, and simplifying yields

$$(Ax_0 + By_0 + Cz_0) + t(A^2 + B^2 + C^2) = D \tag{5.6}$$

so that

$$t = \frac{D - (Ax_0 + By_0 + Cz_0)}{(A^2 + B^2 + C^2)} \tag{5.7}$$

In vector form, Eqn. 5.7 is equivalent to

$$t = \frac{D - (\bar{n} \cdot \vec{Q}_0)}{\|\bar{n}\|} \tag{5.8}$$

where $\vec{Q}(t) = \vec{Q}_0 + \bar{n}t$. The value of t corresponding to each vertex $\vec{v}_i$, is used to project $\vec{v}_i$ along the average plane normal $\bar{n}$, onto the intermediate projection plane P.

Geometric Transformation

A translation, $[T]$, and a general rotation, $[R]$, are required. The average plane is first translated to the origin using the area-weighted centroid, $\bar{x}$, as the reference. The plane is subsequently rotated to the X-Y plane. The projected, transformed points corresponding to the original local boundary loop are then input to the planar Delaunay tessellation algorithm. The result is the candidate local surface tessellation connectivity. Projection plane transformation nomenclature appears in Figure 5.1.

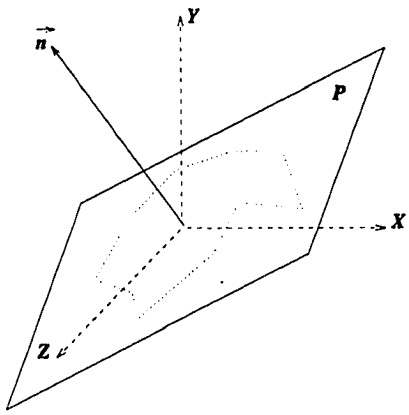


Figure 5.1: Projection plane orientation

First, apply a translation to each loop vertex that moves the area-weighted centroid, $\bar{x}$, to the origin.

$$[T] = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

Next construct the generalized rotation matrix

$$[R] = \begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} & 0 \\ r_{1_y} & r_{2_y} & r_{3_y} & 0 \\ r_{1_z} & r_{2_z} & r_{3_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.10}$$

where R_z is the unit vector that will rotate into the positive z-axis. Here R_z is simply the average plane unit normal, $\bar{n}$.

$$R_z = \bar{n}_x\hat{\imath} + \bar{n}_y\hat{\jmath} + \bar{n}_z\hat{k} = \bar{n} \tag{5.11}$$

The R_x unit vector is formed by the candidate decimation vertex and any point in the adjacent vertex list. Knowing R_x and R_z allows R_y to be defined based on a right-handed coordinate system as

$$R_y = R_z \times R_x \tag{5.12}$$

Thus the composite transformation can be expressed as

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} r_{1_x} & r_{2_x} & r_{3_x} & 0 \\ r_{1_y} & r_{2_y} & r_{3_y} & 0 \\ r_{1_z} & r_{2_z} & r_{3_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.13}$$

where the subscript, p, denotes projected coordinate values. This well-documented projection and coordinate transformation scheme [18, 34] was utilized in an independent effort by Hamann [19]. An example of local boundary loop projection and transformation for a simple 3D surface is presented in Figure 5.2. The adjacent vertices, $\bar{v}_i$, are projected to the average plane along unit normal $\bar{n}$ a distance t_i, and subsequently transformed to the X-Y plane (indicated by the point cluster in the upper right of Figure 5.2).

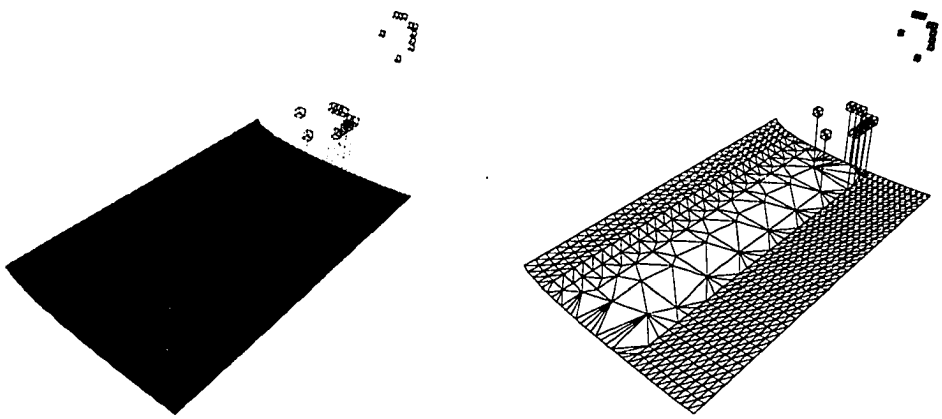


Figure 5.2: Local projection plane for a simple 3D surface: shaded (left), wireframe (right)

Projection Failure

Schroeder et al. [30] observed that, in general, it is not possible to employ a two-dimensional triangulation algorithm to 3D surface decimation problems. The definition and use of a local projection plane may permit projection of a simple polygon on the 3D surface to a non-simple polygon on the projection plane. This non-simple polygon projection "failure" case will not cause the current decimation algorithm to fail. Rather, the candidate vertex will simply be rejected due to the existence of a topological violation. Failure of the original local boundary loop to be preserved is guaranteed in this case since a Delaunay algorithm cannot possibly return self-intersecting edges (connectivity). In any case, it is doubtful whether a vertex with this connectivity should be deleted.

Surface Decimation Algorithm

The general decimation algorithm for surface tessellations is based on the following constraints:

1. Boundary vertices and vertices of degenerate elements are retained. Interior vertices are candidates for removal
2. Average plane tolerance implementation or pathological condition to demonstrate method and numerical robustness

Algorithm:

Given a surface tessellation, for each vertex, v, in the candidate decimation list:

1. Evaluate the vertex decimation criteria (e.g., Schroeder's distance-to-plane criterion)
2. If the decimation criteria are satisfied, remove v. Otherwise exit and proceed to the next vertex in the candidate decimation list
3. Compute the local, area-weighted, average plane
4. Project the adjacent vertices onto the average plane
5. Rotate the average plane to the X-Y plane
6. Apply an unconstrained tessellation algorithm to the projected, transformed vertices defining the nonconvex local boundary loop
7. Classify the new triangles as either interior or exterior with respect to the original local boundary loop
8. Remove the original triangles incident to vertex v
9. Insert the valid triangles identified in Step 7

Boundary Decimation Algorithm

A boundary vertex decimation algorithm may be implemented by processing the list of boundary vertices prior to the interior vertices. At each candidate boundary vertex, identify the two adjacent boundary vertices. Perform a closest point (the candidate boundary vertex) to edge (defined by the two adjacent boundary vertices) distance calculation. Delete the boundary vertex if the distance is less than a prescribed tolerance. In this case, remove the candidate boundary decimation vertex from the local boundary loop. Input the remaining local boundary loop vertices into the local surface decimation algorithm beginning at Step 3.

Applications

Rigorous testing of the surface decimation algorithm is documented in Figures 5.3–5.23. The distance-to-plane criterion is used to identify candidate decimation vertices in each case. Boundary vertices and degenerate topology are explicitly retained. Curvature-based decimation criteria such as those reported by Hamann [19] and Turk [32] were considered, but not implemented. The following results simply demonstrate the generality of the unstructured tessellation algorithm, which was explicitly designed for volume decimation. All shaded figures are rendered using flat triangle shading. Hidden line removal rendering difficulties dictated the presentation of some wireframe pictures. With one exception, all such figures include the shaded image complement.

Decimation results for three surfaces reconstructed from three-dimensional anatomical data appear in Figures 5.3–5.7. Two views of a human pelvis are shown in Figures 5.3 and 5.4, respectively. In each case, the original geometry is located on the left and the decimated surface appears on the right. Approximately 85% of the original 34,939 vertices were removed from the source. The number of triangular faces was reduced from approximately $70k$ to $10k$. The initial geometry contains artificial internal voids (surface clouds) in addition to the obvious topological hole.

A similar presentation format is used to display a human femur surface in Figures 5.5–5.6. Each view depicts the original geometry on the left and the decimated surface on the right. Nearly 82% of the initial 29,820 vertices were deleted. The number of triangular faces was reduced from approximately $60k$ to $10.7k$. Features remain reasonably crisp despite the sparse representation.

A detailed decimation history for a reconstructed horse leg surface is presented in Figure 5.7. The horse leg geometry was reconstructed from a series of slices by using Marching Cubes [23] to generate initial connectivity information. Four decimation steps are shown with both shaded surface and wireframe images. The initial surface consisted of 6,450 vertices and 12,600 triangular faces. Views corresponding to 25%, 50%, 75%, and 85% vertex decimation are illustrated. Recall that each interior vertex removed equates to the deletion of two faces. This complex surface geometry is a rigorous test for the local projection and local tessellation algorithms. The appearance of surface creases as the decimation level increases is expected. The final surface is composed of approximately 1,600 faces.

The wireframe decimation evolution depicted in Figure 5.7 indicates that many of the vertices remaining on the upper horse leg boundary could be removed by applying a boundary

Figure 5.3: Initial pelvis surface (left), 34,939 vertices; 85% decimated (right)

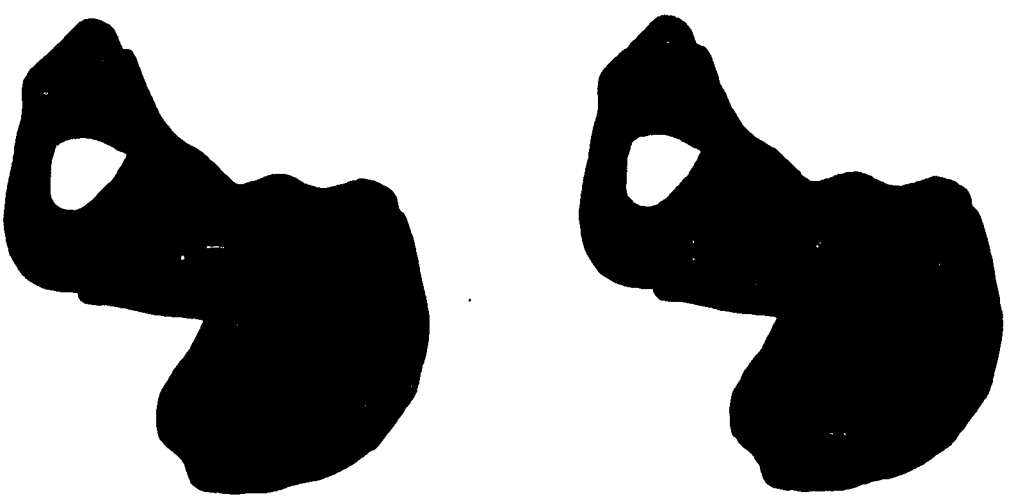


Figure 5.4: Initial pelvis surface (left), 34,939 vertices; 85% decimated (right)

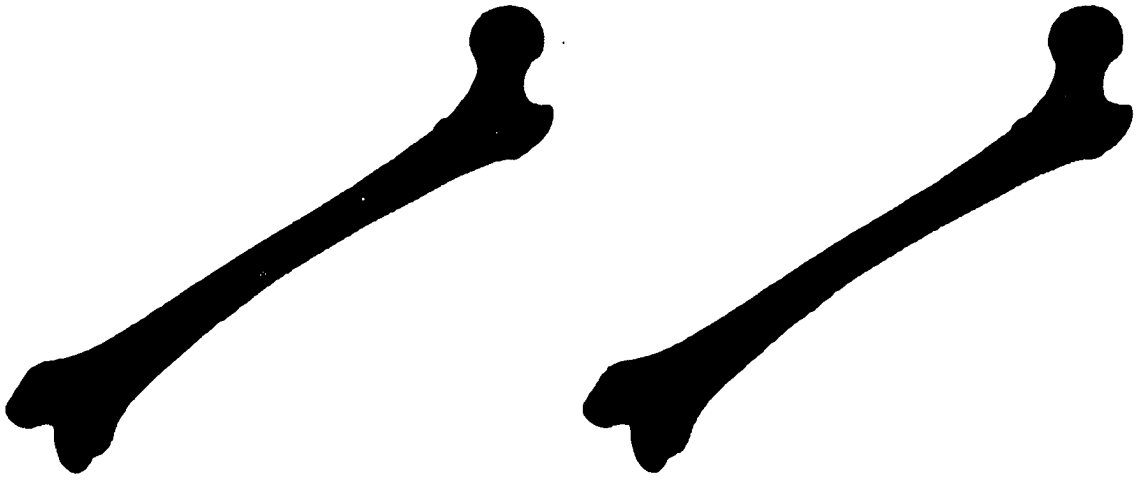


Figure 5.5: Initial femur surface (left), 29,820 vertices; 82% decimated (right)

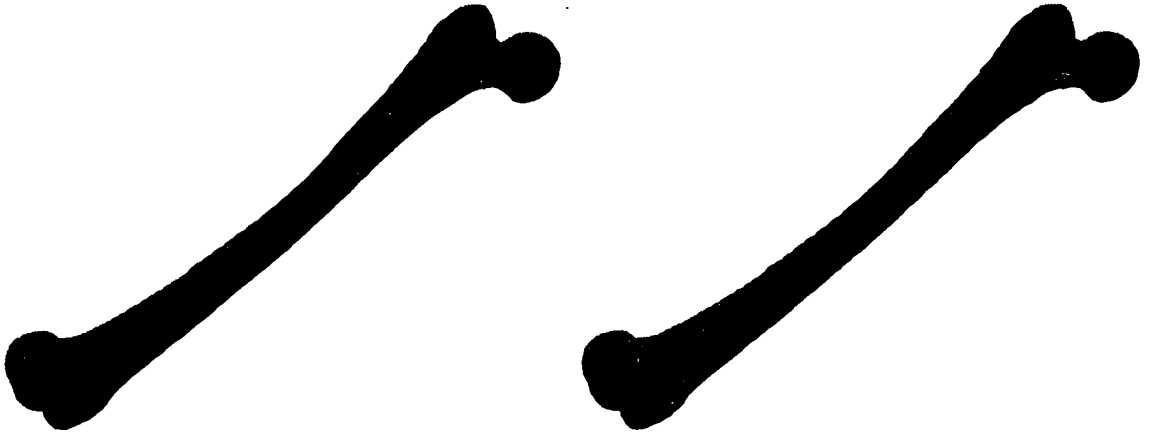


Figure 5.6: Initial femur surface (left), 29,820 vertices; 82% decimated (right)

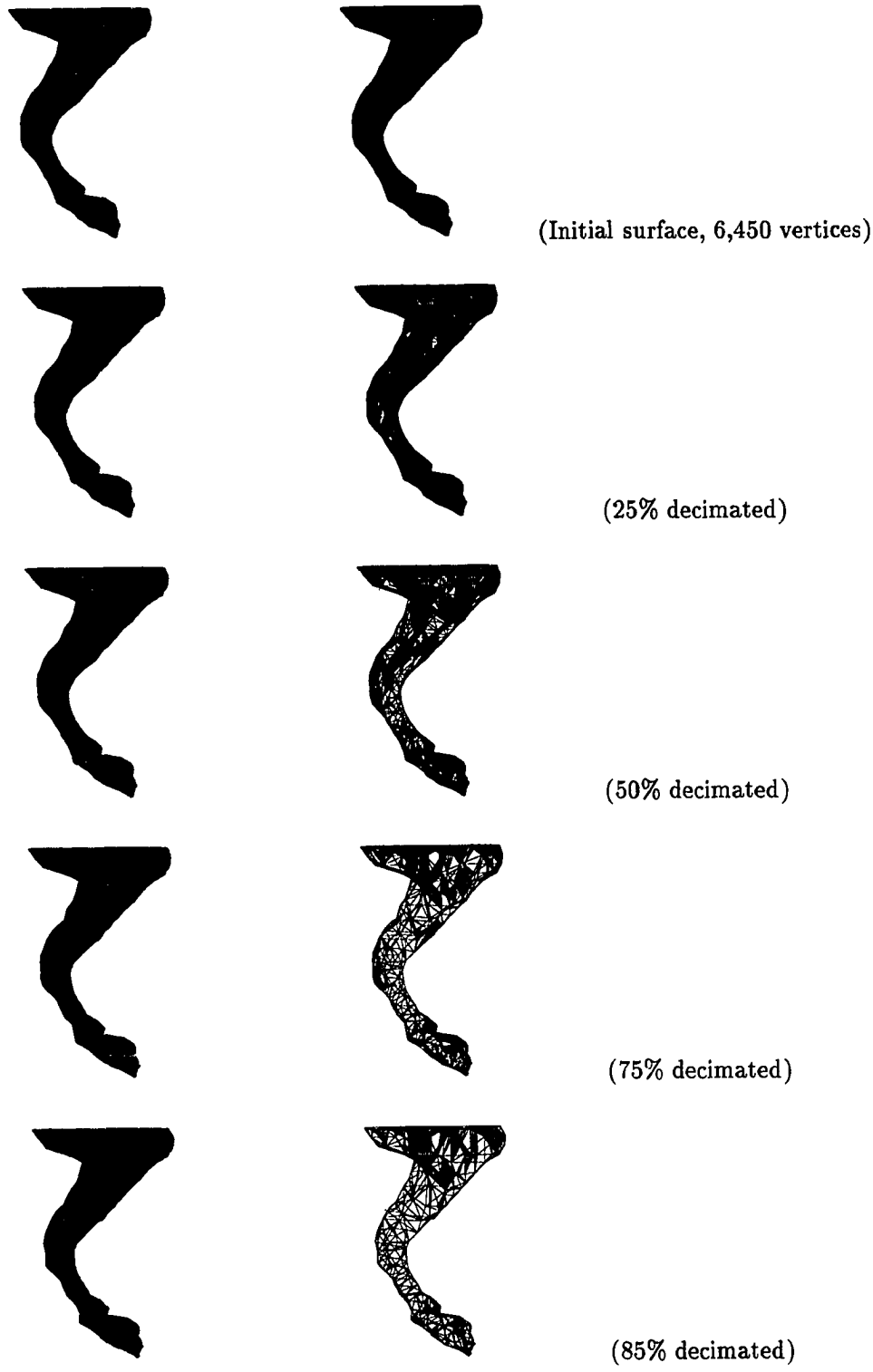


Figure 5.7: Decimation of a horse front leg, five snapshots: shaded (left), wireframe (right)

decimation algorithm. The utility of a boundary decimation scheme (such as the one outlined above) is especially evident for decimation of non-closed surfaces. A discrete model of a vase was constructed by evaluating a non-uniform rational B-spline (NURBS) surface representation. Subsequent decimation of the surface yielded the undesirable boundary seams evident in Figure 5.8.

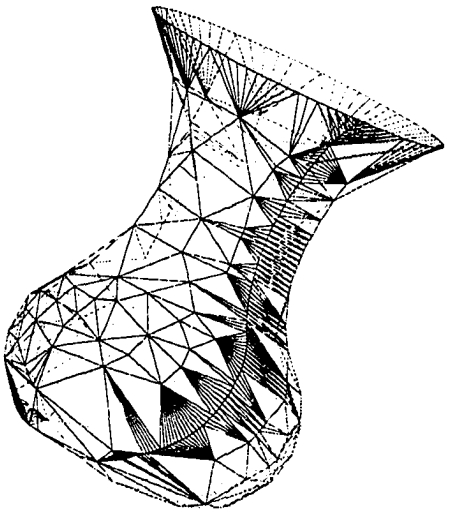


Figure 5.8: Boundary seams remain after decimation of a vase NURBS surface

A general three-dimensional surface decimation histogram is detailed in Figures 5.9–5.14 for a hypothetical sports car. The initial polygonal model, comprised of 13,161 vertices and 25,864 faces, was constructed from a parametric definition. Intermediate shaded surface and wireframe images in Figures 5.10–5.14 represent 37%, 57%, 77%, 90%, and 95% vertex decimation, respectively.

Observe that vertices tend to remain in regions of high surface curvature, but that gradual features fade out as vertices are removed. The fact that the decimation criterion is evaluated based on the current, not the original surface configuration, accounts for the latter negative characteristic. Furthermore, the local nature of the decimation algorithm provides no attempt to guarantee solution symmetry.

The current distance-to-plane decimation criterion is, however, very useful for maintaining sharp features. Crisp edge resolution is required for the simple bracket (Figures 5.15 and 5.16), the driven cavity model (Figures 5.17–5.19), and the microprocessor heat sink (Figures 5.20–5.22) geometry. Each figure set depicts the resulting shaded and wireframe surfaces as the

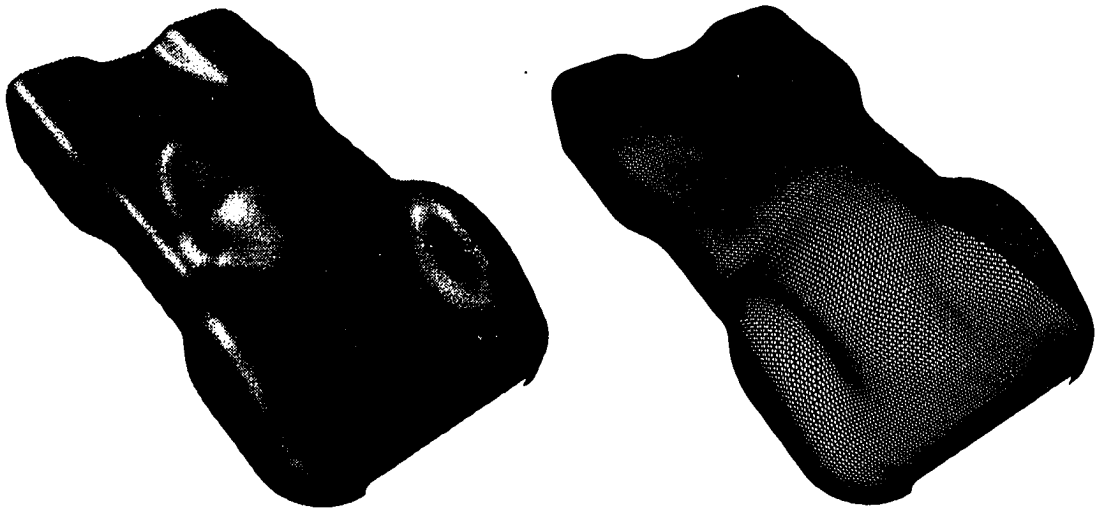


Figure 5.9: Initial sports car surface, 13,161 vertices: shaded (left), wireframe (right)

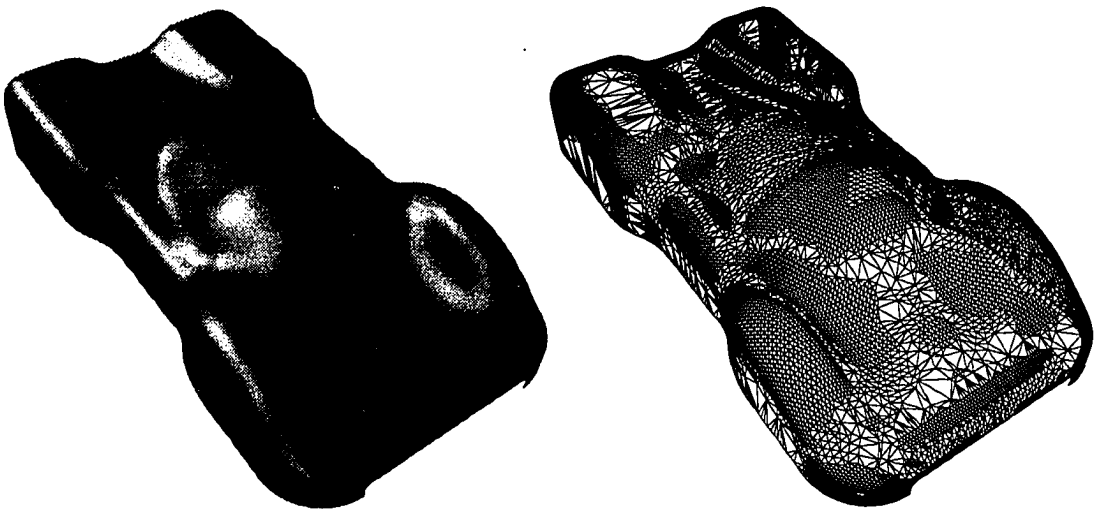


Figure 5.10: Sports car surface, 37% decimated: shaded (left), wireframe (right)

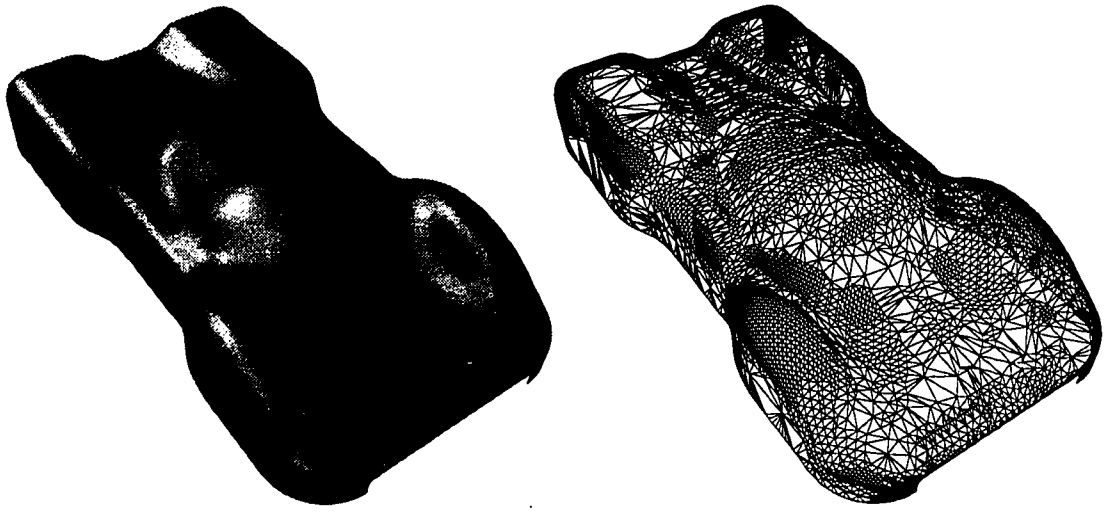


Figure 5.11: Sports car surface, 57% decimated: shaded (left), wireframe (right)

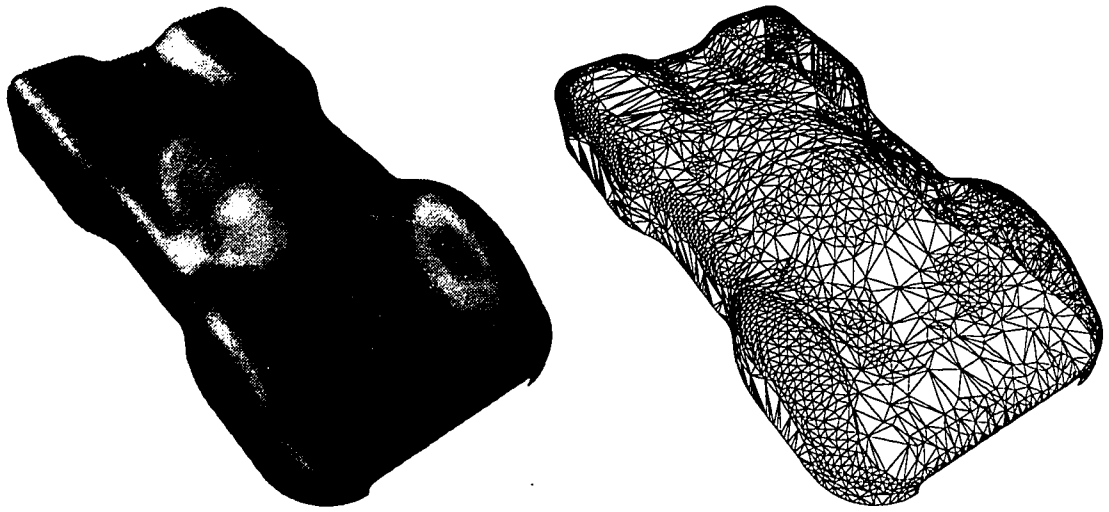


Figure 5.12: Sports car surface, 77% decimated: shaded (left), wireframe (right)

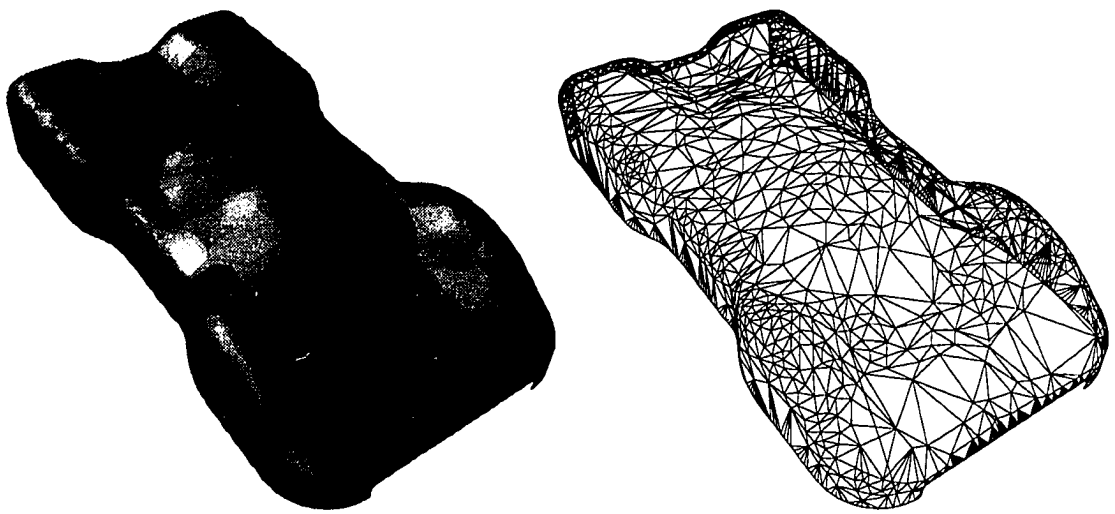


Figure 5.13: Sports car surface, 90% decimated: shaded (left), wireframe (right)

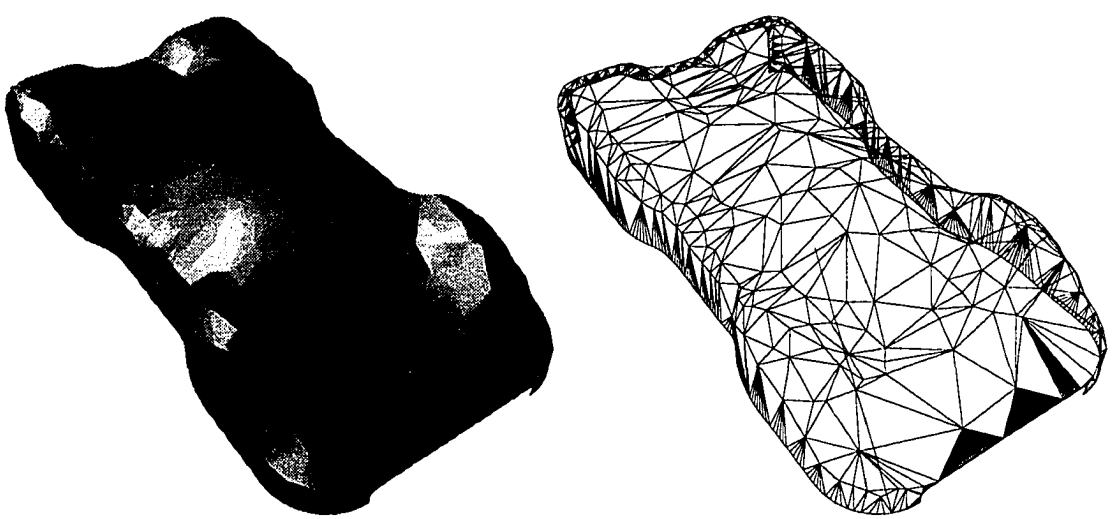


Figure 5.14: Sports car surface, 95% decimated: shaded (left), wireframe (right)

decimation algorithm progresses. The decimated bracket surface illustrates the ability of the local tessellation algorithm to preserve topological features such as "holes." This geometry would benefit from an option to specify local decimation constraints in the mounting hole regions, as opposed to the current global surface constraint. The driven cavity and heat sink geometries permit significant vertex deletion, 95% and 85%, respectively, without sacrificing sharp edge resolution. However, these examples emphasize the need for interior edge and corner decimation criteria, as noted by Schroeder et al. [30].

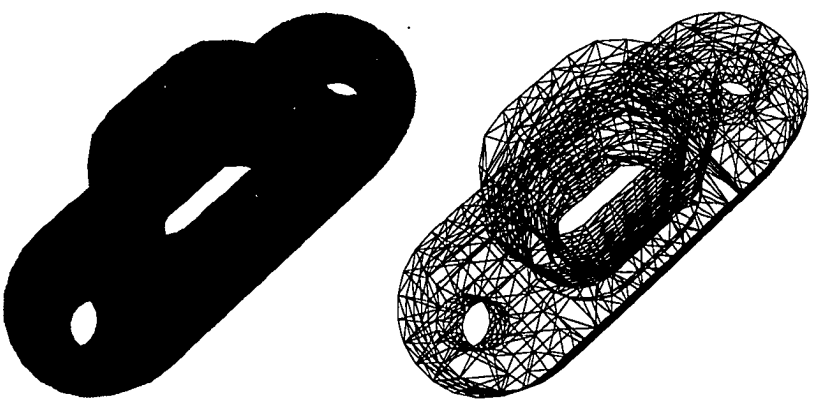


Figure 5.15: Initial bracket surface, 1,204 vertices: shaded (left), wireframe (right)

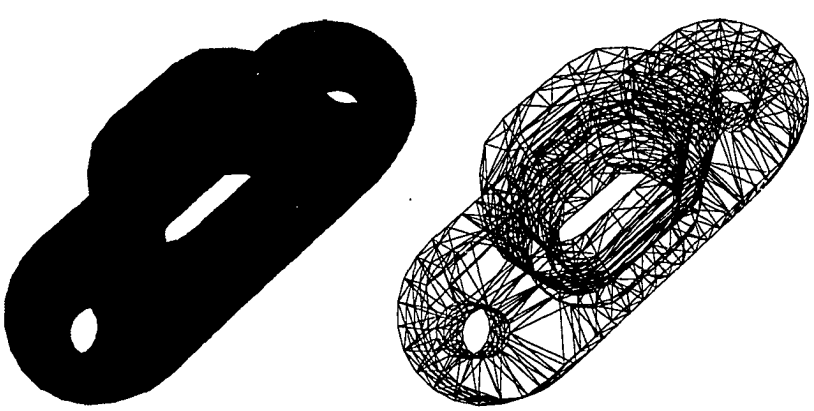


Figure 5.16: Bracket surface, 32% decimated: shaded (left), wireframe (right)

Finally a detailed history of pathological surface decimation appears in Figure 5.23 for the sports car surface. The initial surface is defined by 1,089 vertices. Four decimation snapshots are depicted. All interior vertices are removed to demonstrate the robustness of the surface projection and decimation algorithms.

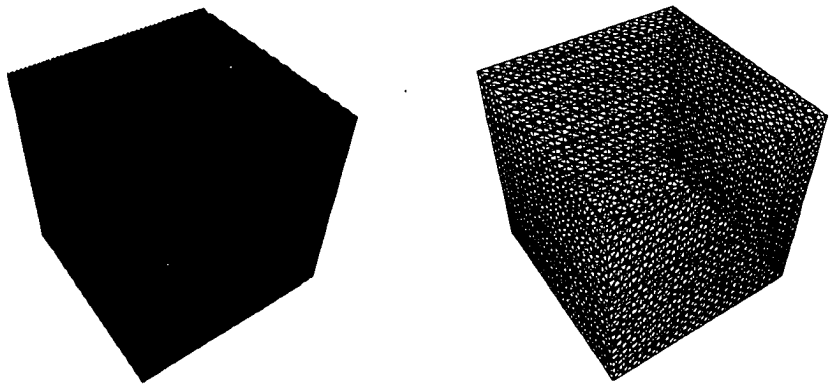


Figure 5.17: Initial cavity surface, 2,402 vertices: shaded (left), wireframe (right)

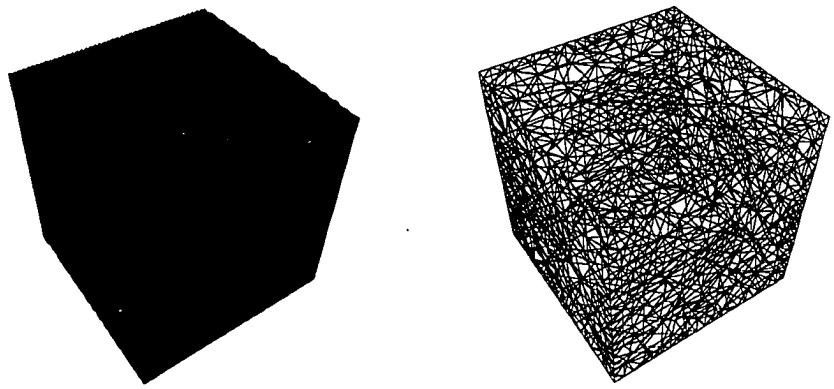


Figure 5.18: Cavity surface, 53% decimated: shaded (left), wireframe (right)

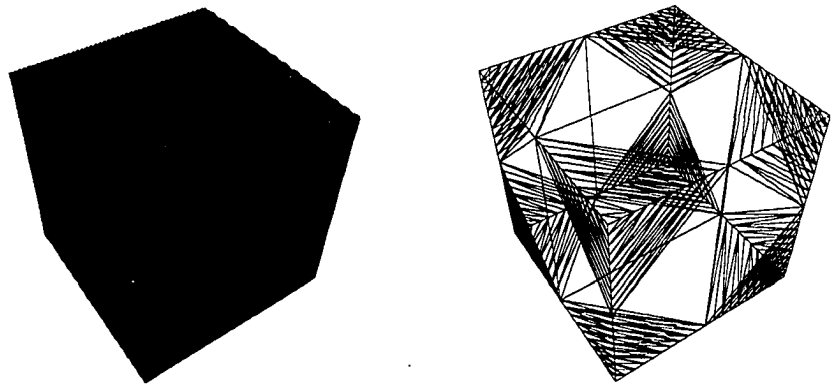


Figure 5.19: Cavity surface, 90% decimated: shaded (left), wireframe (right)

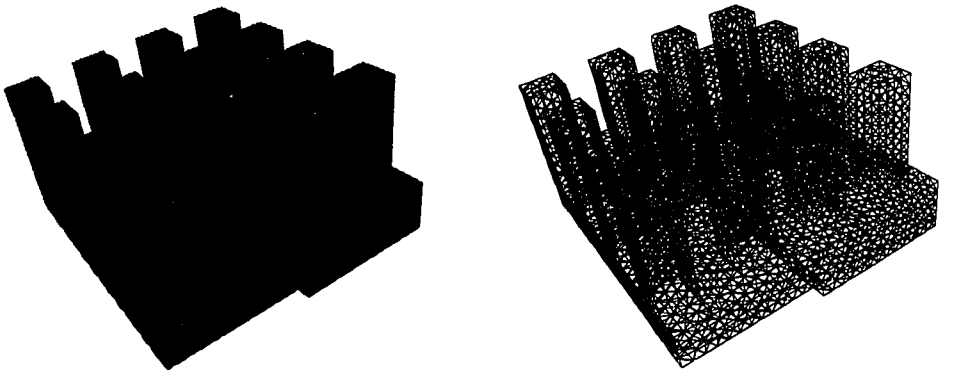


Figure 5.20: Initial heat sink surface, 7,938 vertices: shaded (left), wireframe (right)

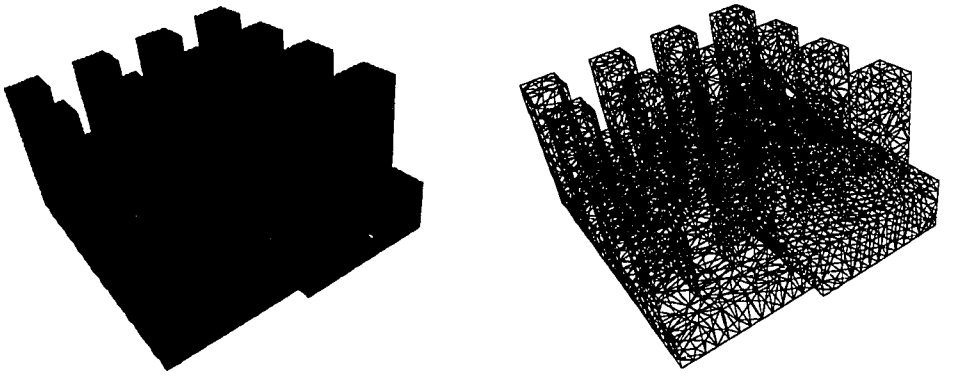


Figure 5.21: Heat sink surface, 63% decimated: shaded (left), wireframe (right)

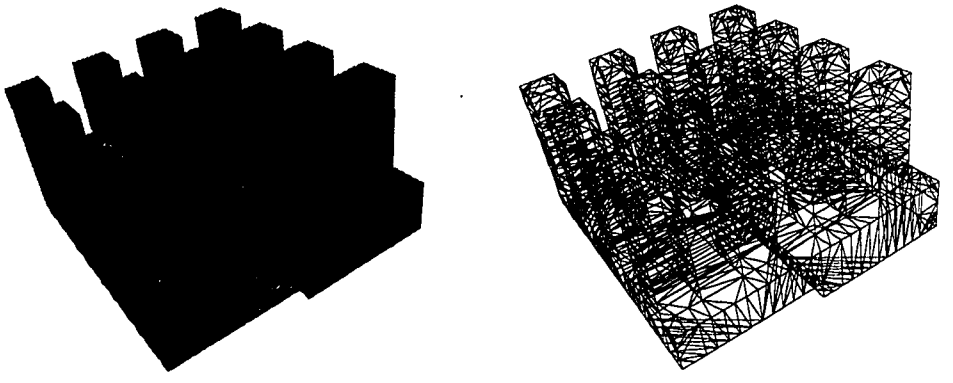
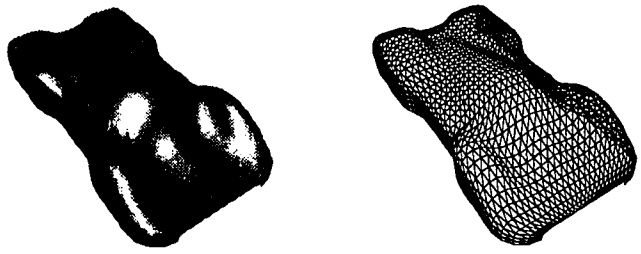
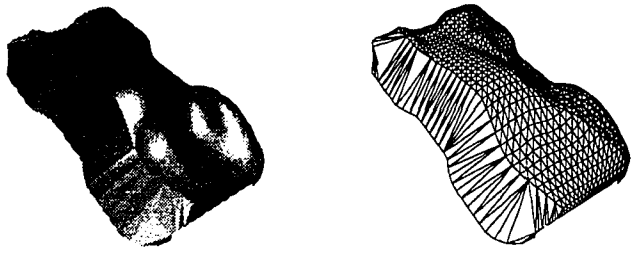


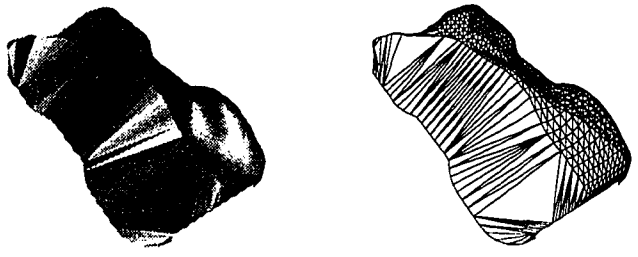
Figure 5.22: Heat sink surface, 85% decimated: shaded (left), wireframe (right)



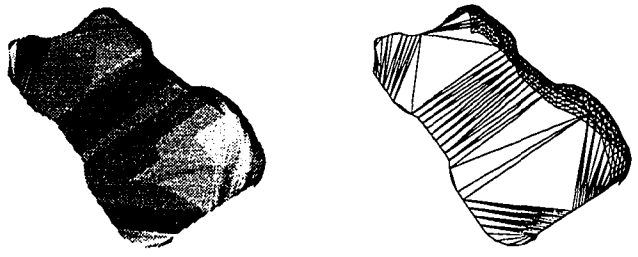
(Initial surface, 1,089 vertices)



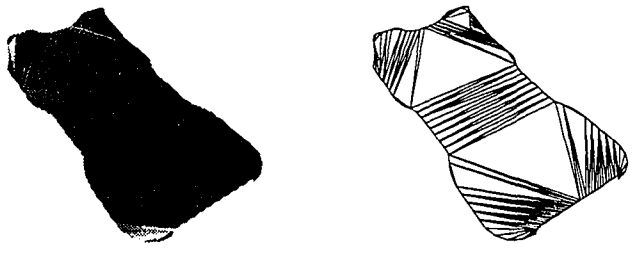
(31% decimated)



(48% decimated)



(65% decimated)



(88% decimated)

Figure 5.23: Pathological decimation of sports car surface, five snapshots: shaded (left), wireframe (right)

CHAPTER 6. VOLUME DECIMATION

The triangle-based reduction algorithm for general surfaces developed by Hamann [19] is projected to have a natural extension to higher-dimensional manifolds. However, Hamann's reduction scheme has not been generalized to higher-dimensional tessellations. In contrast, the method developed in this research may be generalized to higher dimensions in a straightforward manner. Examples of two simple volume tessellations are presented in Figure 6.1 for a tetrahedron and a cube.

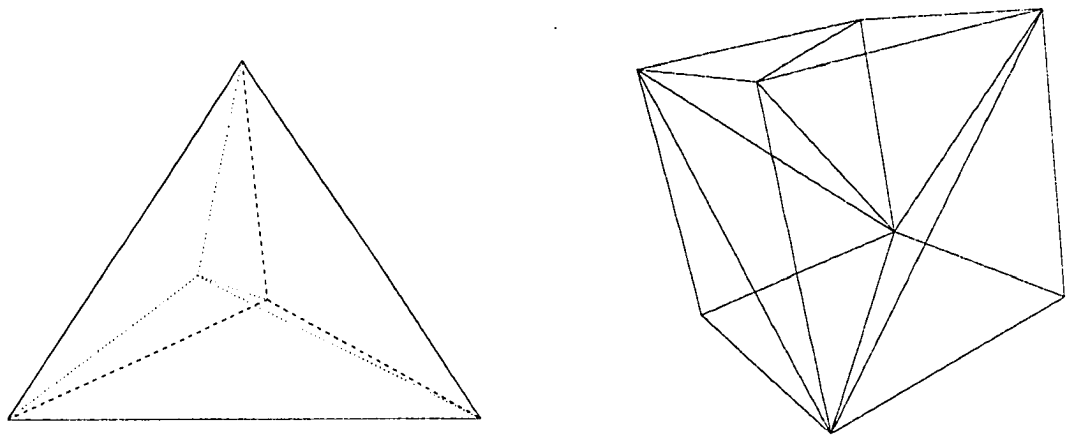


Figure 6.1: Simple tetrahedron and cube volume tessellations

3D Tetrahedrization

A set of points in three dimensions is always tetrahedralizable, with the exception of a coincident, colinear, or coplanar vertex set. The dual property of the three-dimensional Voronoi diagram can be used to construct a volume tetrahedrization. However, a nonconvex three-dimensional polyhedron is not always tetrahedralizable. Formally, the problem can be stated as follows: *Determine if a three-dimensional polyhedron can be decomposed into a set of non-overlapping tetrahedra whose vertices are vertices of the polyhedron.*

This problem is shown to be NP-complete by Ruppert and Seidel [29]. The difficulty arises from the constraints imposed by the existing faces of the polyhedron. An algorithm for tetrahedralizing a nonconvex polyhedron by adding Steiner points (points that are not vertices) is presented by Chazelle and Palios [10]. However, use of this complex algorithm would undermine the present goal of performing vertex-based volume decimation. After all, why risk adding points to tetrahedralize a constrained volume which was initially created by removal of a vertex? Therefore the Chazelle and Palios algorithm is intentionally not implemented.

Inability to tetrahedralize a hole is indicated by topology consistency conditions. If these conditions are not satisfied, the candidate decimation vertex cannot be removed at the current decimation step. The vertex is simply queued at the end of the candidate decimation list in this case.

Volume Decimation Algorithm

The general decimation algorithm for volume tessellations is based on the following constraints:

1. Boundary vertices and vertices of degenerate elements are retained. Interior vertices are candidates for removal
2. Pathological condition to demonstrate method and numerical robustness

Algorithm:

Given a volume tessellation, for each vertex, v, in the candidate decimation list:

1. Remove v
2. Apply an unconstrained tessellation algorithm to the vertices defining the nonconvex local boundary loop
3. Classify the new tetrahedra as either interior or exterior with respect to the original local boundary loop
4. Remove the original tetrahedra incident to vertex v
5. Insert the valid tetrahedra identified in Step 3

The planar local tessellation algorithm developed in **CHAPTER 3** is easily generalized to higher dimensions. The same pitfalls exist, with a few potential additions. The general local tessellation algorithm that follows is analogous to the former, except that triangles are replaced by tetrahedra and edges become triangular faces. In general, triangles are replaced by n-simplices and edges become $(n-1)$-simplices.

General Local Tessellation Algorithm

Given that the decimation criteria have been satisfied for the candidate vertex, the following algorithm is applied to tessellate the hole created by its removal.

1. Identify and store the local boundary loop $(n-1)$-simplices defined by the vertices adjacent to the candidate vertex.
2. Input the list of adjacent vertices into an unconstrained tessellation algorithm and return the new local connectivity.
3. For each new n-simplex
   - Assign a unique identifier to each of its $(n-1)$-simplices
   - Record the number of times a boundary loop $(n-1)$-simplex is shared by the n-simplex
4. Test for the failure case—at least one original boundary $(n-1)$-simplex does not exist in the new tessellation. If true, the candidate vertex cannot be removed without violating the tessellation topology. In this case exit and proceed to the next candidate vertex.
5. Initialize the set of valid $(n-1)$-simplices, which contains the original boundary $(n-1)$-simplices and/or $(n-1)$-simplices belonging to valid interior n-simplices.
6. **Classification—Phase 1:** Sort the new n-simplices onto one of three stacks (**Valid**, **Interior**, or **Exterior**) based on their unique $(n-1)$-simplex identifiers and the original boundary $(n-1)$-simplex set.

   If the n-simplex exclusively shares an original boundary $(n-1)$-simplex, the n-simplex must exist in the interior of the local boundary loop. Therefore remove all common $(n-1)$-simplices from the set of valid $(n-1)$-simplices. Subsequently add remaining $(n-1)$-simplices to the set of valid $(n-1)$-simplices. Then push the n-simplex index onto the **Valid Stack**. Otherwise push the n-simplex onto one of two stacks:

   - **Interior Stack:** no n-simplex $(n-1)$-simplices match an original boundary $(n-1)$-simplex.
   - **Exterior Stack:** one or more n-simplex $(n-1)$-simplices match an original boundary $(n-1)$-simplex.
7. Construct the set of valid interior $(n-1)$-simplices by extracting the set of boundary $(n-1)$-simplices from the set of valid $(n-1)$-simplices.

8. **Classification—Phase 2:** Process the **Interior** and **Exterior** stacks in order while either of these stack dimensions remain dynamic.

   Pop an n-simplex from the current stack. If the intersection of its $(n-1)$-simplex set and the valid interior $(n-1)$-simplex set exists, remove all common $(n-1)$-simplices from the set of valid interior $(n-1)$-simplices. Subsequently add remaining $(n-1)$-simplices to the set of valid interior $(n-1)$-simplices. Then push the n-simplex onto the **Valid Stack**. Otherwise queue (insert) the n-simplex on the bottom of the stack.

9. Upon convergence, exit. Topological implications of the contents of the **Valid Stack** are discussed below in the **Topology Consistency** section.

10. Pop each n-simplex off the **Valid Stack** and insert it into the hole to preserve the original topology and boundary geometry.

Topology Consistency

Recall from **CHAPTER 2** that the number of triangles remaining after the removal of an interior vertex and insertion of the new local tessellation may be computed by means of Euler's formula [28]. For a planar subdivision,

$$V - E + F = 2 \tag{6.1}$$

where V is the number of vertices, E is the number of edges, and F is the number of faces. In general, when an interior vertex is removed, the number of edges must decrease by at least three. Therefore exactly two triangles must be removed. This result holds true for the removal of an interior vertex from a planar or general surface tessellation.

Euler's formula can be generalized in R^m. An equivalent formula that governs 3D tessellations with internal faces, edges, and vertices is documented by Okabe et al. [27]. Let $\bar{T}$ be a tessellation of a bounded set S in R^m, and n_i be the number of i-faces in $\bar{T}$. Then the Euler-Schlaefli formula

$$\sum_{i=0}^{m}(-1)^i n_i = 1 + (-1)^m \tag{6.2}$$

holds. For a general three-dimensional tessellation, $m = 3$, so that Equation 6.2 reduces to

$$n_0 - n_1 + n_2 - n_3 = 0 \tag{6.3}$$

where n_0, n_1, n_2, and n_3 denote the number of 0-, 1-, 2-, and 3-dimensional faces of the polyhedra, usually called the vertices (V), edges (E), faces (F), and tetrahedra (T), respectively.

For the sake of consistency, Equation 6.3 is rewritten as

$$V - E + F = T \tag{6.4}$$

Geometric Necessary Condition

The necessary conditions for area and volume tessellations provide a minimal geometric constraint. The original area or volume contained within the local boundary loop must be conserved if a vertex is deleted from a planar or volume tessellation, respectively. However, this is not a sufficient condition. Nor does it provide information about the number of n-simplices to be inserted into the hole region. The cross-product expressions for triangle area and tetrahedron volume follow.

For a triangle with vertices (a,b,c)

$$Area(a,b,c) = \frac{1}{2}\left|\vec{ab} \times \vec{ac}\right| \tag{6.5}$$

For a general tetrahedron with vertices (a,b,c,d)

$$\begin{aligned} Volume(a,b,c,d) &= \frac{1}{3} \cdot base \cdot height & (6.6)\\ &= \frac{1}{3}\left|\frac{1}{2}\left(\vec{ab} \times \vec{ac}\right) \cdot \vec{ad}\right| & (6.7)\\ &= \frac{1}{6}\left|\left(\vec{ab} \times \vec{ac}\right) \cdot \vec{ad}\right| & (6.8) \end{aligned}$$

Topology Sufficient Condition

Based on the discussion above, deletion of an interior vertex from a general surface topology requires the size of the **Valid Stack** to be exactly two less than the original number of incident triangles. This topology sufficient condition can not be easily extended to volume tetrahedrizations in spite of the existence of Equation 6.4. This is true because the number of edges, faces, and tetrahedra that exist in the final valid local volume tessellation are unknown. Furthermore, Equation 6.4 merely provides a necessary condition.

Therefore the converging method identified as the general local tessellation algorithm was devised. If the Phase 1 classification yields an empty interior $(n-1)$-simplex set, then the hole region is convex or the new local tessellation boundary is interpreted to be topologically inconsistent. Convexity can be tested by comparing the number of new local n-simplices to the number of n-simplices placed on the **Valid Stack**. The Phase 1 classification typically yields a non-empty interior $(n-1)$-simplex set for $n \geq 3$. The Phase 2 classification must result in

an empty interior $(n-1)$-simplex set upon exit in order to guarantee the topology sufficient condition. This sufficient condition renders the tedious check of the local geometric necessary condition obsolete.

Surface Algorithm Departure

Deletion of a vertex, v, from a volume tessellation may yield an increase in the number of tetrahedra tessellating the original local boundary loop, compared to the number of tetrahedra formerly incident at v. Therefore a net decrease in the number of vertices may yield a net increase in the number of tetrahedral elements. This feature may be useful or detrimental, depending on the particular application and optimization goals. While the general algorithm accounts for this detail, the current decimation implementation does not permit the number of tetrahedral elements to increase. A small modification of the memory allocation tracking scheme is required to provide this enhancement.

Failure Case

The volume local tessellation algorithm may fail even in the event that the constrained geometry is tetrahedralizable. Consider the case of a unit cube with an interior vertex, v. Deletion of v creates a degenerate condition for the unconstrained Delaunay tessellation algorithm because more than 4 vertices lie on the same circumsphere. Choice of the diagonal which determines the two faces on each side of the convex hull is not unique. Thus existence of a non-empty interior $(n-1)$-simplex (face) set could be equated to inconsistent topology in the event that some of the original boundary loop faces exist in the candidate volume tessellation while others do not.

Unique Hashing Function

The algorithm requires a unique key corresponding to each boundary face in the local loop to enable face comparison with the faces for the new set of tetrahedra.

> **Problem 2** *For any combination of three unique integer indices, return a unique integer key.*

Recall from Equation 3.6 that a unique prime number was associated with each vertex in the local boundary loop as

$$P(v_i) = Prime\ Number[i] \tag{6.9}$$

Given a unique set of vertex indices and a one-to-one mapping to a list of prime numbers, a unique face index corresponding to any three vertices results from the product of the three prime numbers. In equation form,

$$H_2(v_1, v_2, v_3) = P(v_1) * P(v_2) * P(v_3) \tag{6.10}$$

Volume Boundary Extraction

An automatic method to classify a vertex in a volume tessellation as interior or boundary is useful. In lieu of another equation to solve for one of the two unknowns in Equation 6.4, a search for boundary faces—and thereby boundary vertices—was developed. The search is implemented via a binary tree structure where each node in the tree is a one-dimensional array sorted in ascending order. The tree is dynamic in that only faces which exist on the boundary (i.e., are shared only once) are retained. This method facilitates identification of holes and other interesting topological features.

For strictly convex objects, the convex hull of the vertex set could be used to define the boundary vertices and faces. The exact solution to the boundary classification problem is directly obtained. This method offers a significant savings over a general constrained or unconstrained tetrahedrization followed by a post-processing step to identify boundary faces and vertices.

In general, the normals of the resulting boundary faces are not oriented consistently. The direction of the outward facing normal may be established by intersecting the normal of an arbitrary boundary face with every other boundary face. An odd number of intersections defines an outward normal while an even number of intersections defines an inward normal. However, degenerate cases (e.g., intersections with points or edges) may exist. The orientation procedure begins with the ordered vertices for the current face and recursively cycles through respective incident faces, orienting each in turn. Examples of complex volume boundary extraction and consistent normal vector orientation are illustrated in Figures 6.2 and 6.3 for micro-processor heat sink and F-117 aircraft surface models, respectively.

Applications

Incremental stages of a unit cube volume decimation are illustrated in Figure 6.4. Shaded regions depict tetrahedra rejected during Phases 1 and 2 of the local tessellation classification algorithm for the current candidate decimation vertex. The decimation history is documented

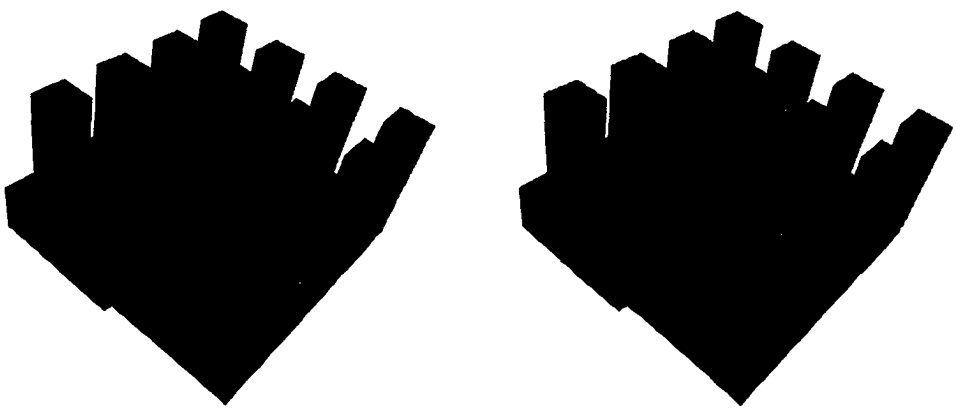


Figure 6.2: Initial shaded heat sink surface, 7,938 vertices: disoriented normals (left), oriented normals (right)

Figure 6.3: Initial shaded F-117 aircraft surface and sting, 30,925 vertices: disoriented normals (left), oriented normals (right)

through snapshots starting from the initial volume tessellation (upper left) to the final tessellation result (lower right) in order from left-to-right, top-to-bottom. The initial volume tessellation consisted of 41 vertices and 189 tetrahedra. Approximately 70% of the original vertices were removed.

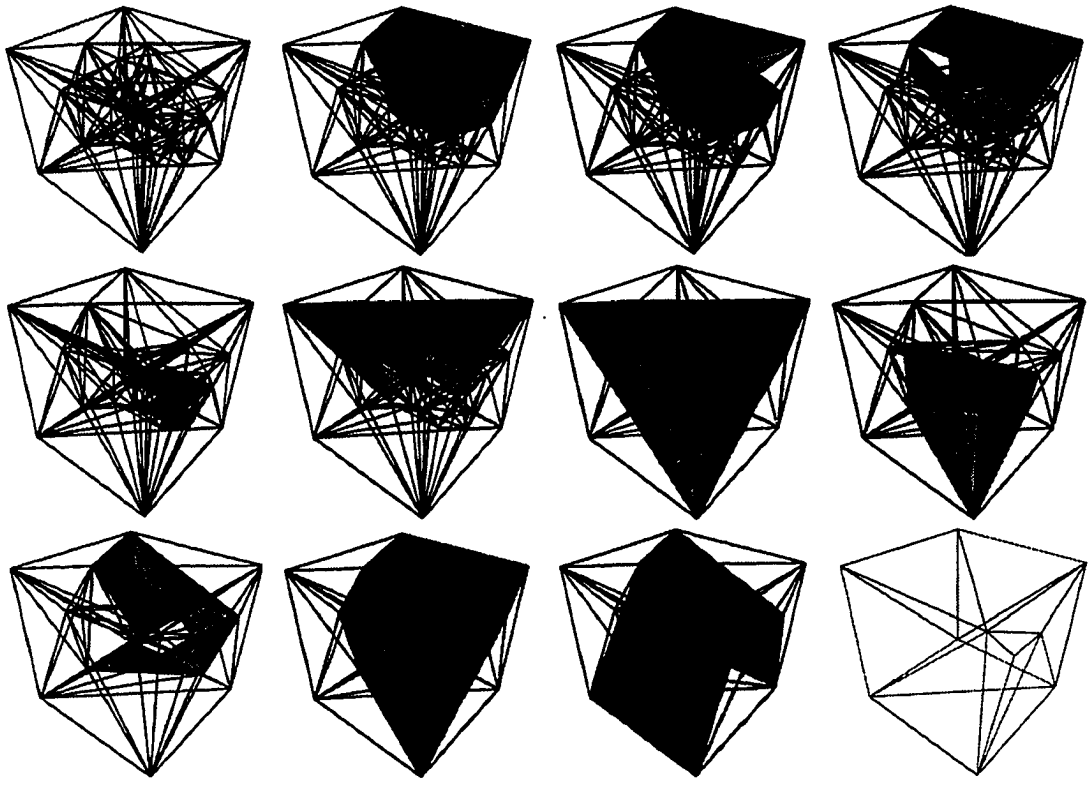


Figure 6.4: Series of decimation steps (and local tetrahedra) for a simple cube

Decimation results are summarized in Tables 6.1–6.3 for nine test geometries. The initial tessellation was generated using either a Delaunay or a Steiner tessellation technique. (Additional points may be added to the initial vertex set during a Steiner tessellation.) The geometry source is either a random cloud of points or a specific CFD volume mesh. Illustrations of the driven cavity and heat sink surface geometry appear in Figures 5.17 and 5.20. A curved channel volume tessellation is depicted in Figure 6.5. The initial shaded surface and wireframe mesh is shown. The geometry is composed of 23,255 interior vertices, 117,126 tetrahedra, and 7,048 boundary vertices.

Pathological vertex decimation results presented in Table 6.1 are impressive. In every case nearly 90% of the original interior vertices were successively removed. The 99 percentile interior

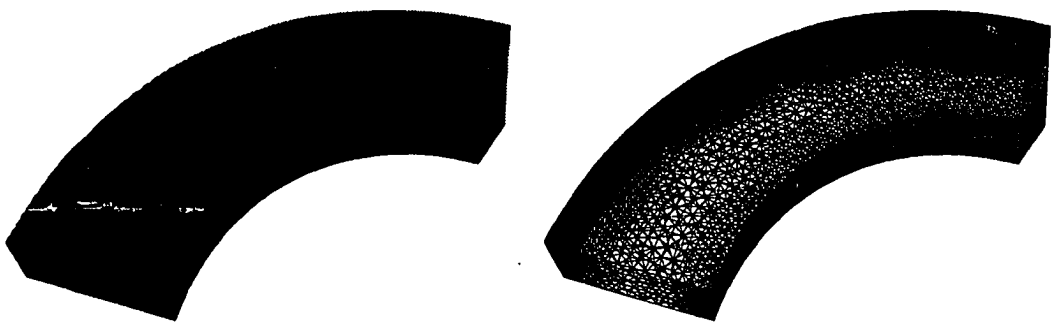


Figure 6.5: Curved channel volume mesh, 23,255 vertices: shaded (left), wireframe surface (right)

vertex decimation results for initial Delaunay meshes supercedes the low-to-mid 90 percentile ranking of the four initial Steiner tessellations. The data suggests that the Delaunay-based local tessellation algorithm is more efficient for decimating initial Delaunay tetrahedrizations than for initial Steiner volume tessellations. In general, the decimation algorithm assumes that the local volume tessellation will be a subset of the candidate local Delaunay tessellation.

Table 6.1: Volume decimation vertex percentages for Delaunay and Steiner tessellations

Initial Vertices	Geometry Source	Initial Tessellation	Vertices			
			Boundary	Removed	% Initial	% Interior
1,000	Random points	Delaunay	70	929	92.90	99.89
3,000	Random points	Delaunay	92	2,899	96.63	99.69
4,000	Random points	Delaunay	103	3,886	97.15	99.72
5,000	Random points	Delaunay	109	4,875	97.50	99.67
6,000	Random points	Delaunay	121	5,863	97.72	99.73
7,000	Constrained	Steiner	129	6,842	97.74	99.58
7,938	Heat sink	Steiner	4,304	3,236	40.77	89.05
12,837	Driven cavity	Steiner	3,980	8,448	65.81	95.38
23,255	Curved channel	Steiner	7,048	15,343	65.98	94.67

The percentage of initial tetrahedra deleted in each case is itemized in Table 6.2. The limited results also support a decimation efficiency differentiation between Delaunay and Steiner initial tessellation methods.

The initial maximum number of incident tetrahedra among all vertices in the set is reported in Table 6.3. Perhaps more interesting, the maximum number of candidate decimation

vertices rejected per complete iteration (one pass through the remaining candidate list) is detailed. These vertices were rejected solely on the basis that the post-decimation total number of tetrahedra in the local boundary loop would increase (recall the **Surface Algorithm Departure** discussion). Results support implementation of the aforementioned memory tracking enhancement to improve computational efficiency, particularly for Steiner tessellations.

In conclusion, testing of the pathological decimation constraints reveals robust performance manifest in the preservation of initial topology and boundary geometry for nonconvex geometries which may include internal voids.

Table 6.2: Volume decimation tetrahedra percentages for Delaunay and Steiner tessellations

Initial Vertices	Geometry Source	Initial Tessellation	Tetrahedra		
			Initial	Removed	% Initial
1,000	Random points	Delaunay	6,302	6,056	96.10
3,000	Random points	Delaunay	19,687	19,317	98.12
4,000	Random points	Delaunay	26,374	25,958	98.42
5,000	Random points	Delaunay	32,974	32,395	98.24
6,000	Random points	Delaunay	39,674	39,186	98.77
7,000	Constrained	Steiner	46,479	45,535	97.97
7,938	Heat sink	Steiner	32,939	22,454	68.17
12,837	Driven cavity	Steiner	64,490	52,368	81.20
23,255	Curved channel	Steiner	117,126	98,523	84.12

Table 6.3: Volume decimation incident tetrahedra and rejection rate for Delaunay and Steiner tessellations

Initial Vertices	Geometry Source	Initial Tessellation	Initial Maximum Incident Tetrahedra	Maximum Occurrence Increased Tetrahedra
1,000	Random points	Delaunay	52	2
3,000	Random points	Delaunay	62	12
4,000	Random points	Delaunay	60	22
5,000	Random points	Delaunay	62	44
6,000	Random points	Delaunay	75	53
7,000	Constrained	Steiner	82	87
7,938	Heat sink	Steiner	46	20
12,837	Driven cavity	Steiner	40	58
23,255	Curved channel	Steiner	40	144

CHAPTER 7. ALGORITHM PERFORMANCE

With the exception of Hoppe et al. [20], many of the published surface decimation algorithms lack performance statistics with respect to order analysis or actual run times. The following timing performance experiments were conducted on a Silicon Graphics 150 Mhz R4400 CPU with 128 MB RAM.

Planar Tessellation

A logarithmic plot of CPU seconds versus planar triangulation time for Watson's Delaunay tessellation scheme is shown in Figure 7.1. Random data sets ranging in magnitude from 25 to $200k$ vertices were generated within a circular convex hull by using separate random number invocations for vertex radius and angular position. The equation of the linear portion of the plot based on linear regression analysis is

$$log_{10}\ Time = m\ log_{10}\ Vertices - log_{10}\ b \tag{7.1}$$

which reduces to

$$Time = \frac{Vertices^m}{b} \tag{7.2}$$

where $m = 2.0823$ and $b = 2.6856 \times 10^6$. Equation 7.2 suggests that Watson's planar algorithm is of $O(N^2)$. However, in the original publication, Watson claimed that the execution time would grow more slowly than $N^{(n+1)/n}$, where $n = 2$ for a planar tessellation.

Planar Decimation

The decimation velocity is defined as

$$Velocity = \frac{Vertices}{CPU\ Time} \tag{7.3}$$

Each data point in Figures 7.2–7.4 represents the average time required to remove 500 interior vertices and update the connectivity for the current tessellation state. Removal of one interior

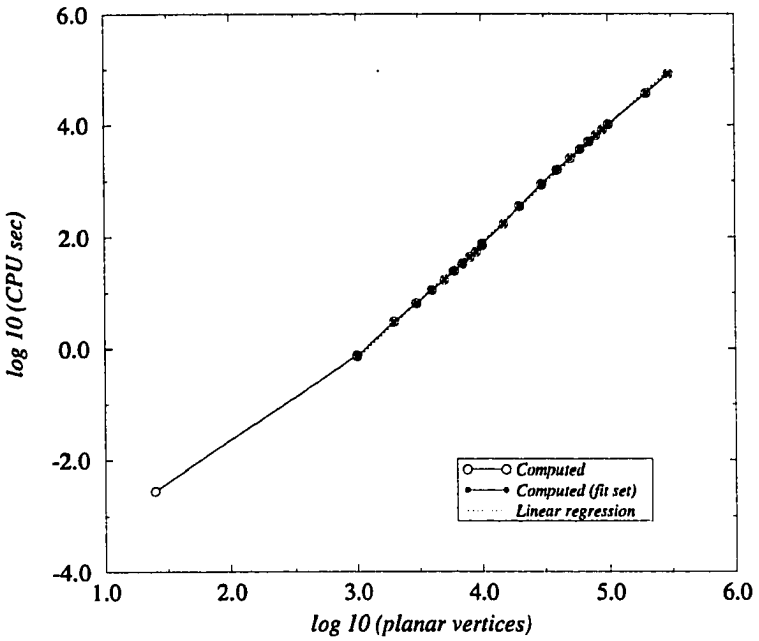


Figure 7.1: Watson's planar tessellation scheme performance

vertex corresponds to the removal of two triangles. Thus the limiting case in Figure 7.2 represents the deletion of $190k$ vertices ($380k$ triangles). The largest mesh size tested was $200k$ vertices ($190k$ interior, $10k$ boundary).

Two important characteristics of the decimation algorithm are illustrated in Figure 7.2. First, the average decimation velocity remains constant until the boundary limit is approached. The boundary limit appears in the form of a sharply decreasing velocity as the maximum number of vertices is approached. As interior vertices are removed, the decimation time approaches the time required to tessellate the set of retained boundary points. Second, because the data structures are not stored in contiguous blocks of memory, a performance bound exists for large problem sizes.

The original decimation algorithm was implemented with continuous dynamic memory allocation. That is, memory was allocated and deleted for each local candidate decimation vertex. The current algorithm is implemented based on peak need. The minimum required memory space is reallocated if the current allocation is insufficient. Performance ramifications of the peak memory allocation scheme are presented in Figure 7.3. As expected, minimizing the overhead of memory reallocation/deallocation produces a measurable performance improvement. A direct comparison of memory allocation effects on timing performance for several planar cases

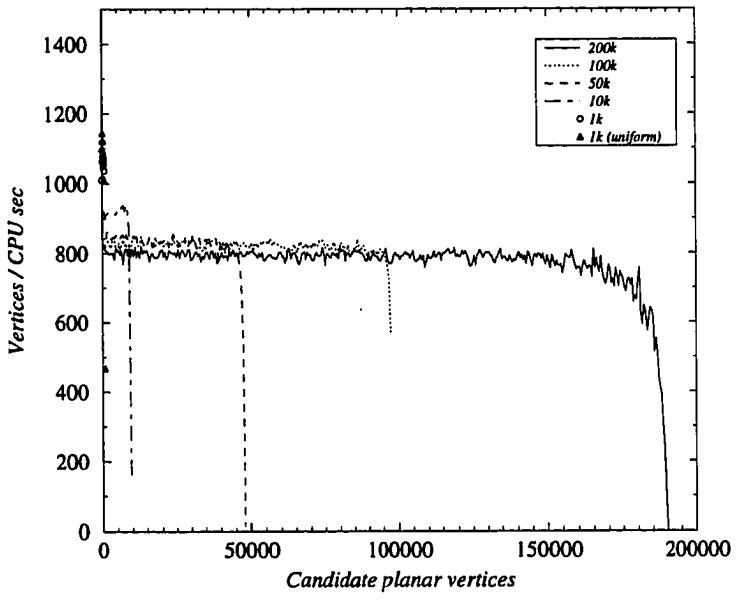


Figure 7.2: Planar decimation performance (dynamic memory allocation)

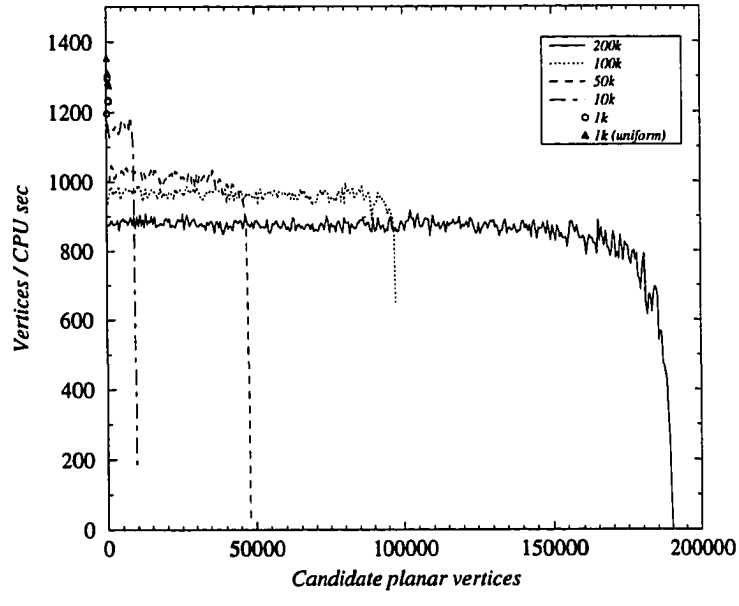


Figure 7.3: Planar decimation performance (peak memory allocation)

is shown in Figure 7.4. Peak memory allocation yields a 10% performance improvement over the dynamic memory allocation scheme for the $200k$ vertex mesh.

The cost of the tessellation classification scheme is indirectly documented in Figure 7.5. The amount of time required to compute the candidate local tessellation is 4-5 times less than the overall average time required to compute the local tessellation, perform the classification, and update the connectivity for the $200k$ vertex unstructured Delaunay mesh. The average local tessellation data was measured without actually performing vertex removal or updating the local connectivity. Therefore a direct performance comparison is hampered by the fact that the connectivity structure at each decimation step is not identical. Finally, since no vertices are actually removed, the performance penalty due to the encroaching boundary vertex limit does not appear in the local tessellation simulation data.

Surface Decimation

Surface decimation performance statistics are plotted in Figure 7.6 for the peak memory allocation scheme. Planar decimation results for $100k$, $200k$, and $300k$ vertex meshes are shown for comparison. Identical planar unstructured tessellations were used as a constant base, with projection plane computations incorporated for the surface performance results. Each data point represents the average time required to remove 500 interior vertices and update the local connectivity. The additional work required to project and transform the local boundary loop vertices for general surface decimation is less than 10% of the overall effort for the cases tested. The maximum vertex size, $300k$, corresponds to approximately $600k$ triangular faces.

Volume Decimation

Volume decimation performance results for the array of Delaunay and Steiner tessellations introduced in **CHAPTER 6** are shown in Figure 7.7 for the peak memory allocation scheme. An order of magnitude separates the absolute performance of the volume scheme from the planar and surface tessellation results. The increased complexity of creating a tetrahedrization versus a triangulation is obvious.

The cost of the volume tessellation classification scheme is also indirectly documented. The amount of time required to compute the candidate local volume tessellation is 4-5 times less than the nominal time required to compute the local tessellation, perform the classification, and update the connectivity. This factor is virtually identical to the planar tessellation simulation

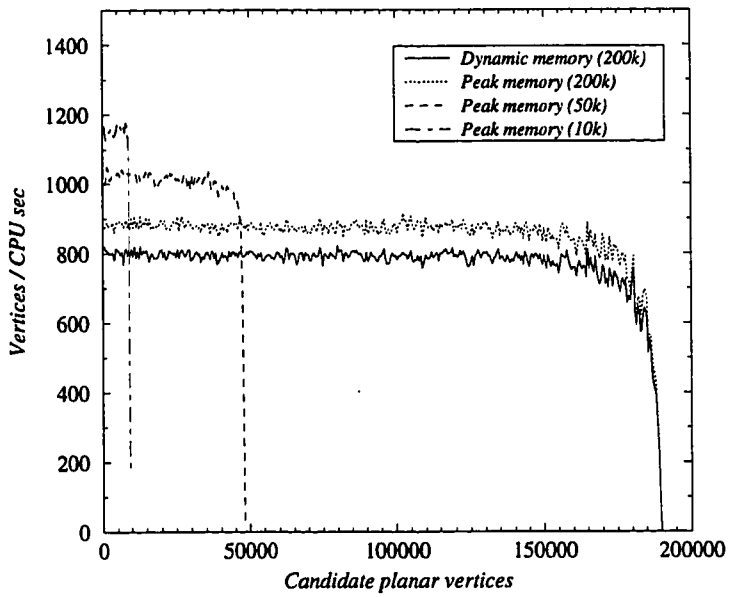


Figure 7.4: Memory allocation effects on planar decimation performance

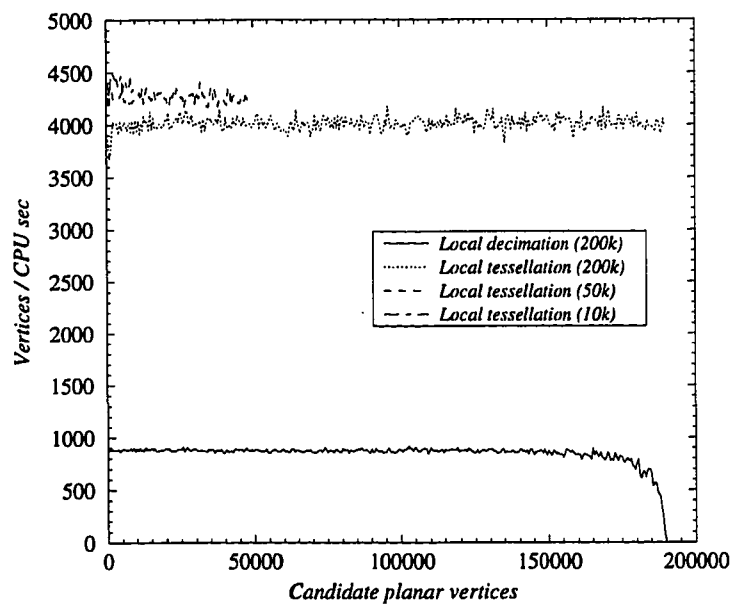


Figure 7.5: Local tessellation simulation and decimation performance (planar, peak memory allocation)

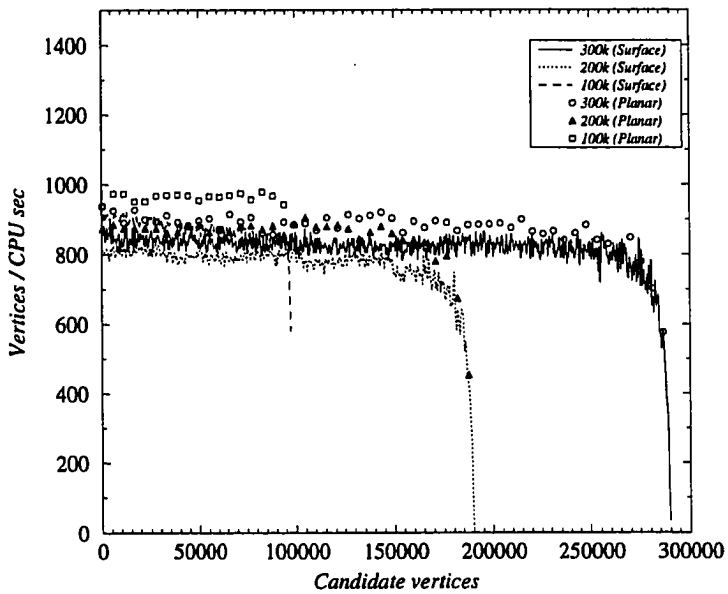


Figure 7.6: Surface decimation performance (peak memory allocation)

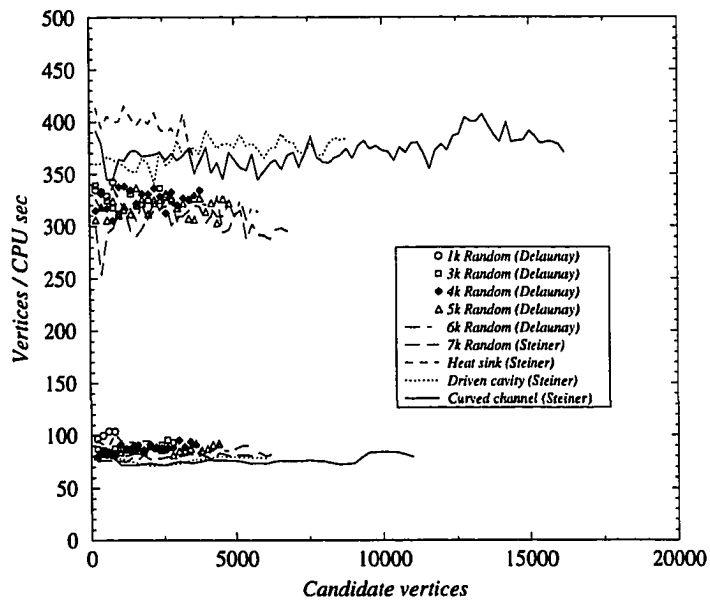


Figure 7.7: Local tessellation simulation and decimation performance (volume, peak memory allocation)

results reported above. The average local tessellation data was measured without actually performing vertex removal or updating the local connectivity. A direct performance comparison is thus belabored by the fact that the connectivity structure differs at each decimation step.

Nominal Tessellation Size

A sample of the number of incident triangles per candidate decimation vertex for an initial unstructured mesh size of $100k$ vertices is plotted in Figures 7.8 and 7.9. The former graph shows a closeup of the average trend while the latter details the overall process. Two conclusions are clearly evident. First, the average tessellation problem size input to the unconstrained Delaunay algorithm is approximately 10 vertices. Second, the dynamic nature of the decimation process is most turbulent as the boundary mesh resolution is approached (i.e., the number of incident triangles per vertex sharply increases toward the maximum number of vertices since boundary vertices are preserved). While the average problem size remains small through most of the process, at times there are clearly hundreds of triangles incident at the candidate decimation vertex. This pattern behavior is repeated for the $10k$ vertex data presented in Figure 7.10, but not for the 2,000 vertex case. The anomalous nature of the initial 2,000 vertex unstructured mesh contributes a sporadic local tessellation problem size throughout the decimation process.

A similar sample of incident tetrahedra density per candidate decimation vertex is plotted in Figure 7.11 for initial unstructured volume mesh sizes of 7,938 and 23,255 vertice. This data corresponds to the heat sink and curved channel CFD geometry introduced earlier. The average tessellation problem size input to the unconstrained Delaunay algorithm is approximately 25 vertices.

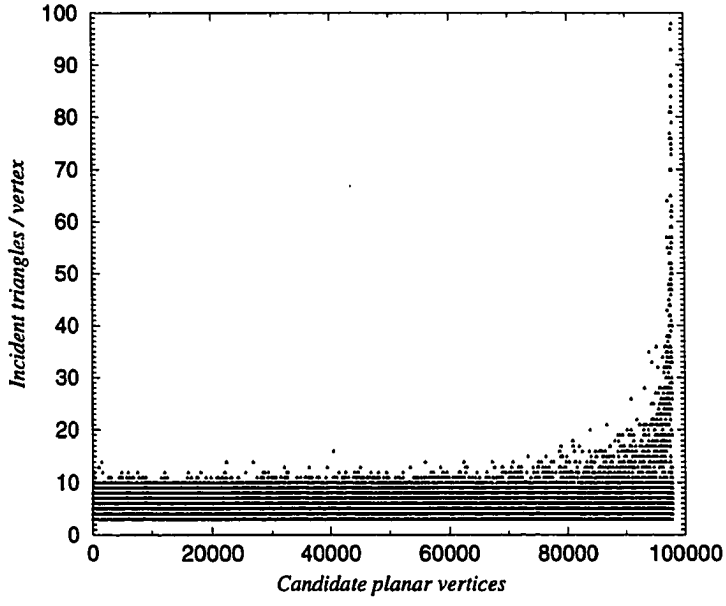


Figure 7.8: Incident triangles per candidate decimation vertex (nominal range)

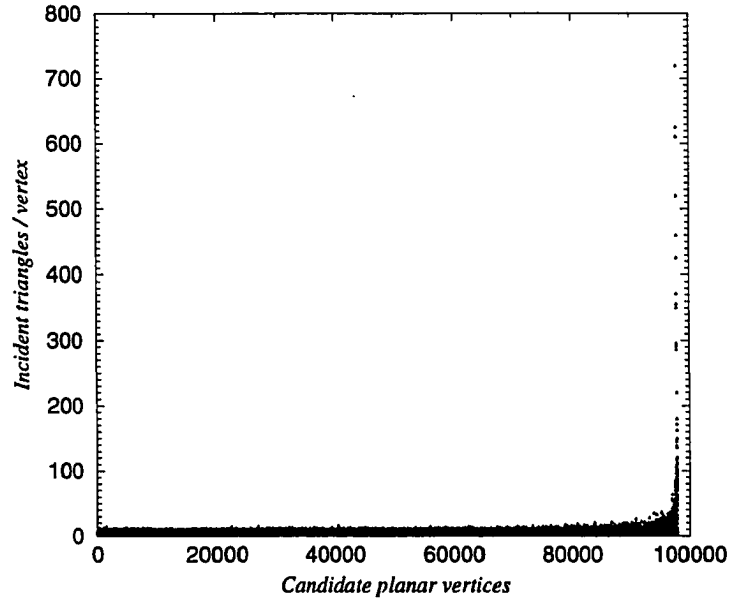


Figure 7.9: Incident triangles per candidate decimation vertex (sample extremes)

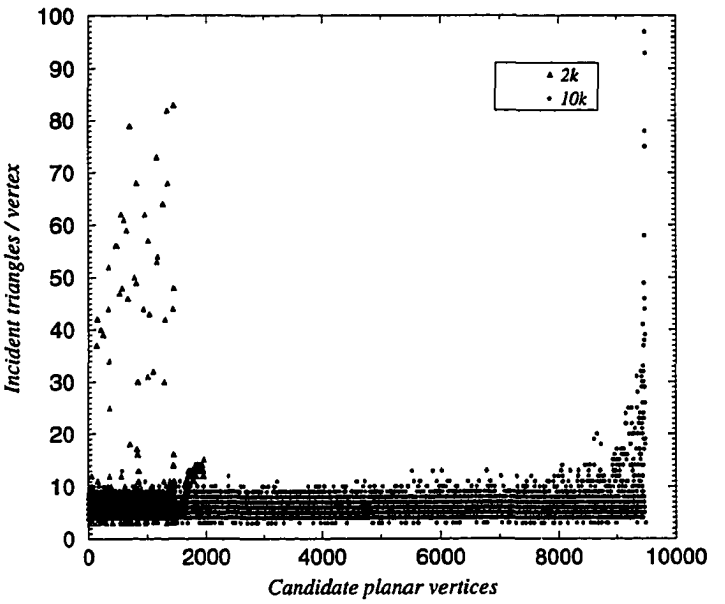


Figure 7.10: Incident triangles per candidate decimation vertex (2,000 and 10,000 initial planar vertices)

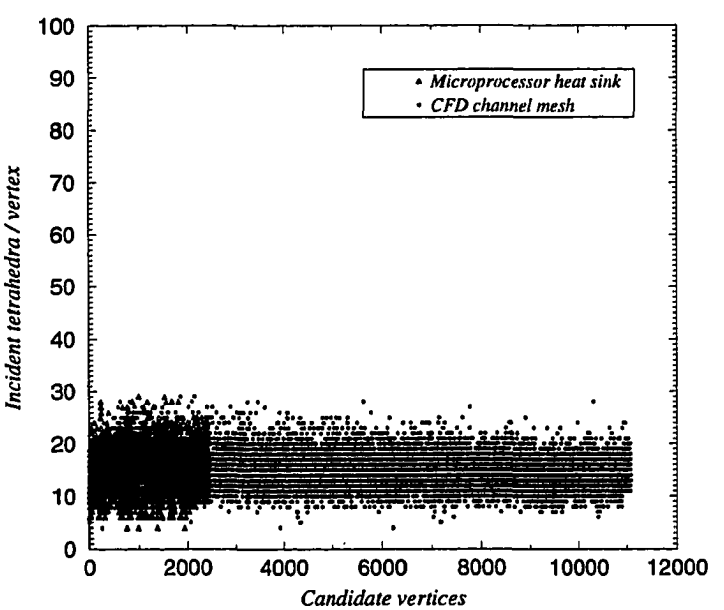


Figure 7.11: Incident tetrahedra per candidate decimation vertex (7,938 and 23,255 initial vertices)

CHAPTER 8. CONCLUSIONS

The formulation, implemention, and testing of a general algorithm to reduce the complexity of 2D and 3D triangulated meshes and 3D tetrahedral volumes is presented. The product is a generalized framework for two- and three-dimensional surface and volume decimation of unstructured discretized data sets. Local dynamic vertex removal is performed without history information while preserving the initial topology and boundary geometry. The focus of this research is how to remove a vertex from an existing unstructured n-dimensional tessellation, not on the formulation of decimation criteria. The criteria for removing a candidate vertex may be based on geometric properties or any scalar governing function specific to the application.

In general, the decimation algorithm assumes that the new local tessellation will be a subset of the candidate local Delaunay tessellation. While this works well in practice, regardless of how the original data set was generated, fairly trivial failure cases can be constructed. Data sets generated from NURBS surface sampling at arbitrary sampling density, as well as connectivity produced by applying a Delaunay algorithm to random point sets yielded excellent decimation results. Problems arise in enforcing global decimation constraints on certain geometries where local constraint application would be more prudent.

The following conclusions are made:

1. The decimation algorithm can be generalized to n-dimensional Delaunay tessellations. The current implementation demonstrates the functionality and robustness for unstructured planar, general surface, and volume tessellations for both Delaunay and non-Delaunay meshes.
2. The general decimation algorithm is conceptually simple.
3. The decimation algorithm is independent of a sorted data structure, which is critical for three-dimensional volume decimation.
4. Unlike the angle summation or ray-intersection methods, the local tessellation algorithm used to solve the n-simplex classification problem is integer based. This feature eliminates decision errors introduced by floating point comparisons.

5. The topology-based classification algorithm provides a sufficient condition which avoids degenerate cases inherent in the ray-intersection method.
6. The method presented applies to both convex and nonconvex polygons.
7. The decimation algorithm guarantees a Delaunay mesh after each decimation step, assuming the initial mesh is Delaunay and degenerate cases (e.g., four co-circular vertices in 2D or five co-spherical vertices in 3D) do not exist.
8. It is possible to construct an algorithm for bi-directional surface or volume mesh adaption (i.e., more coarse or refined) using the current decimation algorithm in conjunction with a point insertion method.
9. The enforcement of globally desirable properties (e.g., symmetry) is not guaranteed. Classical methods to handle this limitation include reflecting the solution for the partial domain across the symmetry plane.
10. The global solution is currently dependent on the starting location. However, development of reasonable decimation criteria will limit or negate potential negative repercussions.

Improvements

Developing an option to perform localized decimation in user-defined regions represents a feature of significant practical value. An interactive constraint interface that would enable the user to specify multiple regions of interest, each with independent local constraints, is envisioned. Similarly, this concept extends naturally for strict boundary decimation applications.

Enhancements to the current decimation algorithm implementation are logical. The most efficient existing n-dimensional unconstrained Delaunay tessellation algorithm should be incorporated. Current data structures were designed to foster intuitive access to tessellation information, with the explicit mission of demonstrating the feasibility of the proposed decimation algorithm—not necessarily to provide minimal storage or most efficient access. Therefore opportunity exists for improved efficiency through better data structure design. Finally, the bit concatenation-based hashing function should replace the current prime number-based hashing function.

The research community is actively addressing the question of how to intelligently remove or compress huge volumes of data. Aggressive solutions continue to evolve. An avenue of research with respect to the current decimation algorithm is, for a target model resolution,

what criteria can be used to identify "unnecessary" points. The answer is at least partially application dependent.

Future Research

The general nature, overall robustness, and computational efficiency of this new decimation algorithm suggest several avenues for further research. Volume visualization offers a particularly broad and rich area for future investigation. Given scalar or vector information distributed through a volume, the traditional approach is to generate section cuts or simply view iso-surfaces. Volume decimation may prove to be an effective tool for manipulating volume data into more meaningful and manageable representations. Accelerated interest in this field is represented by the work of Cignoni et al. [14]. An adaptive incremental technique is used to construct a sequence of tetrahedrizations for multi-resolution modeling and visualization of unstructured volumetric data.

Although the algorithm is quite efficient in its current form, research toward parallel implementations of the current planar, surface, and volume algorithms may facilitate novel uses, e.g., dynamic model resolution maintenance for visual simulation or virtual environment applications. Finally, in addition to obvious computer graphics and engineering analysis applications, the current algorithm could be employed in fields such as medical modeling using CAT and MR scans, meteorological mapping, and geophysical modeling based on seismic experiments or physical core samples.

Computational Geometry Impact

The classification scheme developed for the new decimation algorithm may offer a new alternative to solve 2D and 3D point-location problems (i.e., detect whether or not a point is inside a nonconvex boundary). The identification of a point as interior or exterior to a faceted boundary is clearly within the application scope. In fact, it may be possible to sort multiple points simultaneously. A candidate point(s) could simply be added to the set of points defining the local boundary loop. Subsequent application of the unconstrained Delaunay tessellation algorithm to this coordinate data would return a triangle/tetrahedra set. If all the triangles/tetrahedra incident to the point in question are members of the interior or exterior stack, the point classification is obvious. Otherwise the point lies on the local boundary loop or the original local boundary loop connectivity was not preserved. This scheme is limited

to a half-space implementation, where all candidate points must lie on the interior side of an edge/face of the local boundary loop convex hull. The existence of trivial failure cases may negate any potential utility of this technique.

The classification algorithm may also impact research efforts to determine whether or not a constrained set of 3D points is tetrahedralizable without adding additional vertices. The current convergence criteria will reject local tessellations that are not topologically consistent. However, extensions of the current research are once again constrained by the half-space limit and may be adversely impacted by the issue of whether or not the original local boundary loop is preserved in the candidate local tessellation.

REFERENCES

[1] T.J. Baker, "Automatic mesh generation for complex three-dimensional regions using a constrained (Delaunay) triangulation," *Journal of Engineering with Computers*, 5 (3-4): 161-175, 1989.

[2] T.J. Baker, "Three-dimensional mesh generation by triangulation of arbitrary point sets," *AIAA 87-1123-CP*, 1987.

[3] A. Gandhi and T.J. Barth, "3D unstructured grid refinement and optimization using 'edge-swapping'," NASA Ames Research Center, Moffett Field, CA, 1993.

[4] T.J. Barth, N.L. Wiltberger, and A.S. Gandhi, "Three-dimensional unstructured grid generation via incremental insertion and local optimization," NASA Ames Research Center, Moffett Field, CA, 1992.

[5] T.J. Barth, "On unstructured grids and solvers," NASA Ames Research Center, Moffett Field, CA, 1990.

[6] T.J. Barth, "Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations," *Lecture Notes for the von Karman Institute for Fluid Dynamics*, NASA Ames Research Center, Moffett Field, CA, 1994.

[7] M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," *Xerox PARC Technical Report, CSL-92-1*, Xerox Palo Alto Research Center, 1992.

[8] W.L. Briggs, *A multigrid tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[9] Y. Bowyer, "Computing Dirichlet tessellations," *Computer Journal*, 24: 162-166, 1981.

[10] B. Chazelle and L. Palios, "Triangulating a non-convex polytope," *Proceedings of the 5th Annual ACM Symposium on Computational Geometry*, 393-400, 1989.

[11] L.P. Chew, "Guaranteed-quality mesh generation for curved surfaces," *Cornell Report*, Department of Computer Science, Cornell University, Ithaca, New York, Year 1992.

[12] L.P. Chew, Private communication, January 1995.

[13] B.K. Choi, H.Y. Shin, Y.I. Yoon, and J.W. Lee, "Triangulation of scattered data in 3D space," *Computer Aided Design*, 20 (5): 239-248, 1988.

[14] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno, "Multiresolution modeling and visualization of volume data based on simplicial complexes," Proceedings of ACM SIGGRAPH, *1994 Symposium on Volume Visualization*, Washington, D.C., 1994.

[15] P. Cignoni, C. Montani, and R. Scopigno, "A merge-first divide and conquer algorithm for E^d Delaunay triangulations," Submitted November 1992 to *Computational Geometry: Theory and Application*, pending review.

[16] Y. Correc and E. Chapuis, "Fast computation of Delaunay triangulations," *Advanced Engineering Software*, 9 (2): 77-83, 1987.

[17] M.J. DeHaemer, Jr. and M.J. Zyda, "Simplification of objects rendered by polygonal approximations," *Computers & Graphics*, 15 (2): 175-184, 1991.

[18] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer graphics: Principles and practice*, 2nd ed., Addison-Wesley, Reading, Massachusetts, 1990.

[19] B. Hamann, "A data reduction scheme for triangulated surfaces," *Computer Aided Geometric Design*, 11 (2): 197-214, 1994.

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," *Computer Graphics*, Proceedings of ACM SIGGRAPH 1993, Annual Conference Series, 19-26, 1993.

[21] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics*, Proceedings of ACM SIGGRAPH 1992, 26 (2): 71-78, 1992.

[22] M.A. Khan, "A mesh reduction approach to parametric surface polygonization," *M.S. Thesis*, Iowa State University, 1994.

[23] W. Lorensen and H. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, Proceedings of ACM SIGGRAPH 1987, 21 (4): 163-169, 1987.

[24] C.M. Maksymiuk and M.L. Merriam, "Surface reconstruction from scattered data through pruning of unstructured grids," *AIAA-91-1584*, 1991.

[25] D.J. Mavriplis and V. Venkatakrishnan, "Agglomeration multigrid for viscous turbulent flows," *AIAA-94-2332*, 1994.

[26] M.L. Merriam, "3D CFD in a day: The laser digitizer project," *AIAA-91-1654*, 1991.

[27] A. Okabe, B. Boots, and K. Sugihara, *Spatial tessellations: Concepts and applications of Voronoi diagrams*, John Wiley & Sons, Chichester, 1992.

[28] F.P. Preparata and M.I. Shamos, *Computational geometry: An introduction*, Springer-Verlag, New York, 1985.

[29] J. Ruppert and R. Seidel, "On the difficulty of tetrahedralizing three-dimensional non-convex polyhedra," *Proceedings of the 5th Annual ACM Symposium on Computational Geometry*, 380-392, 1989.

[30] W.J. Schroeder, J.A. Zarge and W.E. Lorensen, "Decimation of triangle meshes," *Computer Graphics*, Proceedings of ACM SIGGRAPH 1992, 26 (2): 65-70, 1992.

[31] M. Tanemura, T. Ogawa, and N. Ogita, "A new algorithm for three-dimensional Voronoi tessellation," *Journal of Computational Physics*, 51: 191-207, 1983.

[32] G. Turk, "Re-tiling polygonal surfaces," *Computer Graphics*, Proceedings of ACM SIGGRAPH 1992, 26 (2): 55-64, 1992.

[33] D.F. Watson, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes," *The Computer Journal*, 24 (2): 167-172, 1981.

[34] I. Zeid, *CAD/CAM theory and practice*, McGraw-Hill, Inc., New York, 1991.

APPENDIX 1. TWO-DIMENSIONAL CLASSIFICATION

The figures that follow illustrate post-Phase 1 and Phase 2 triangle classification results for both convex and nonconvex local boundary loops. The number in each figure indicates the original position of the removed vertex. The associated former unstructured mesh can be reconstructed by connecting this point to every other local boundary loop vertex. These examples were selected to demonstrate the robustness of the local tessellation algorithm.

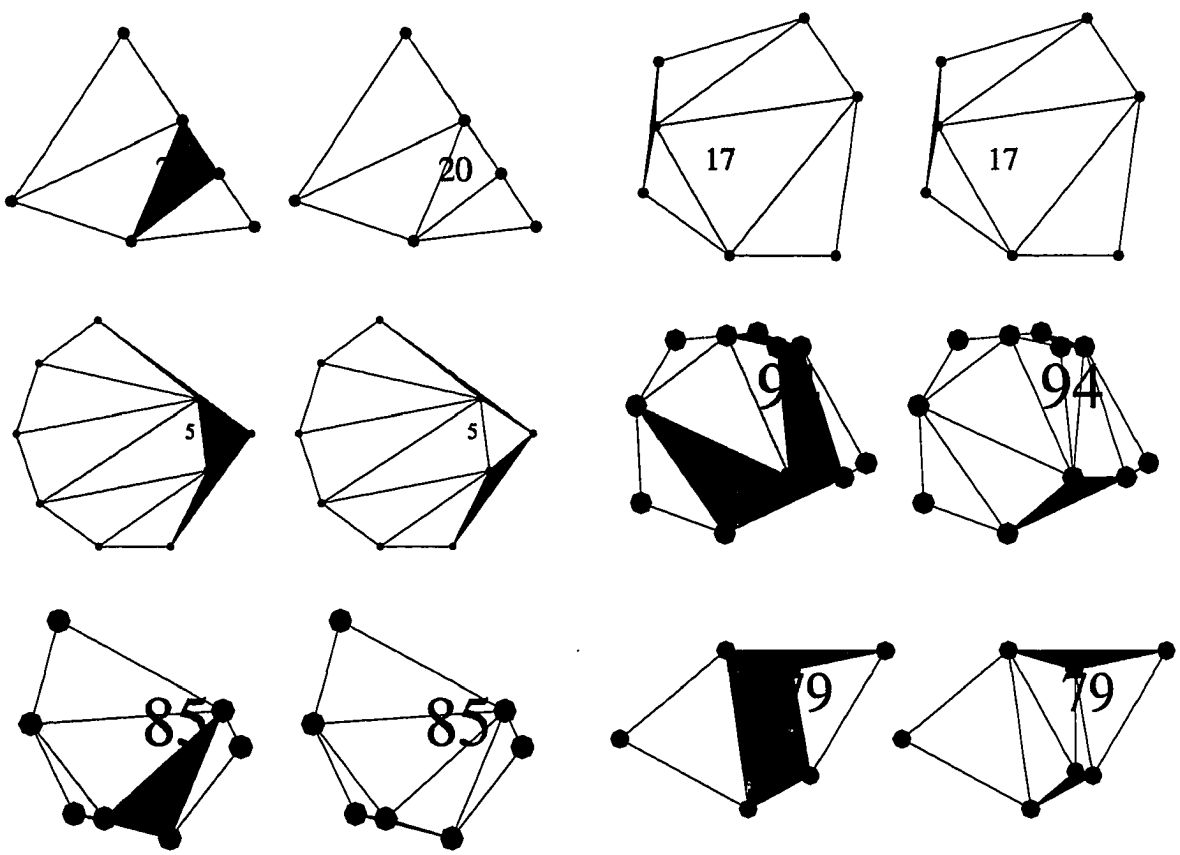


Figure A.1: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

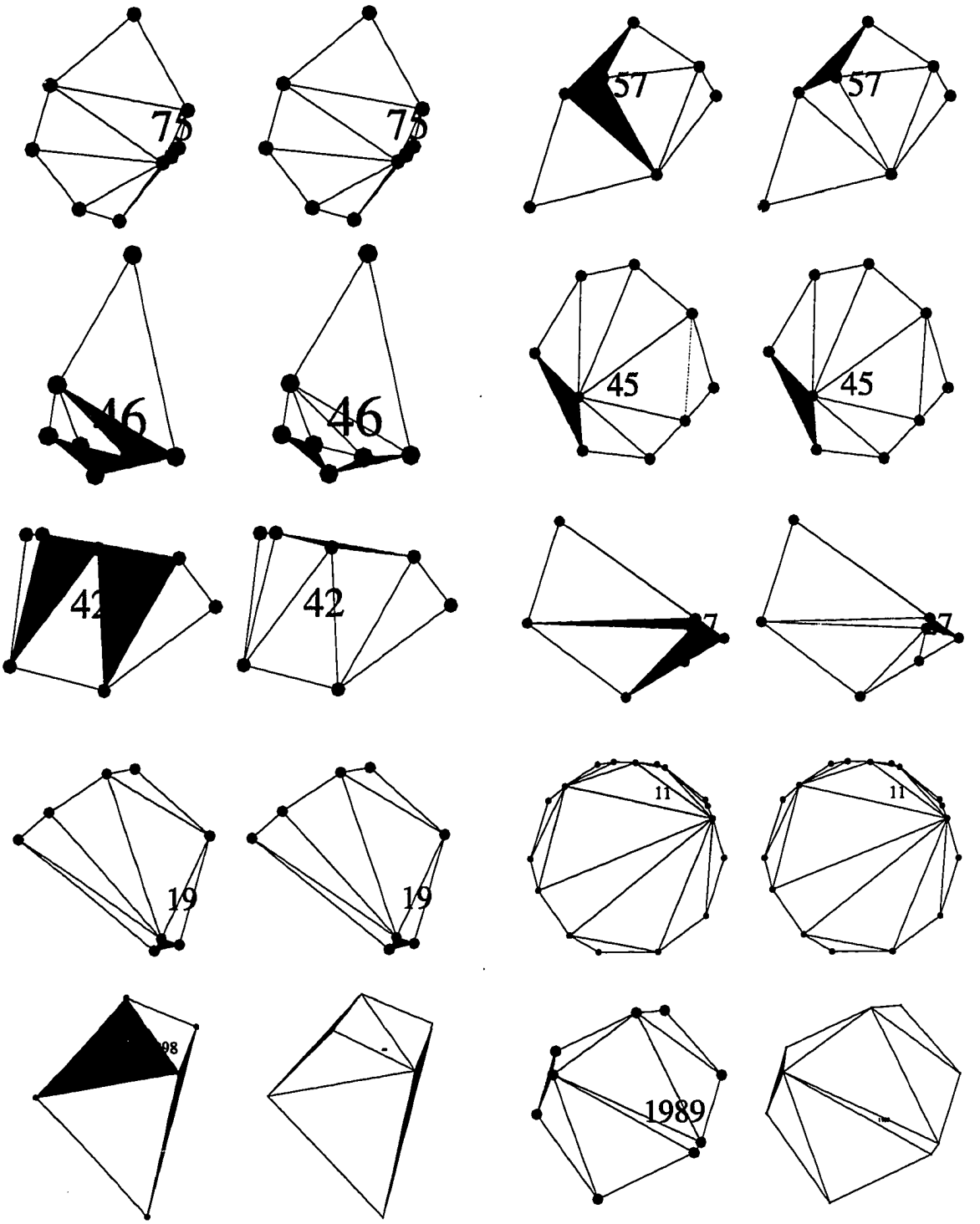


Figure A.2: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

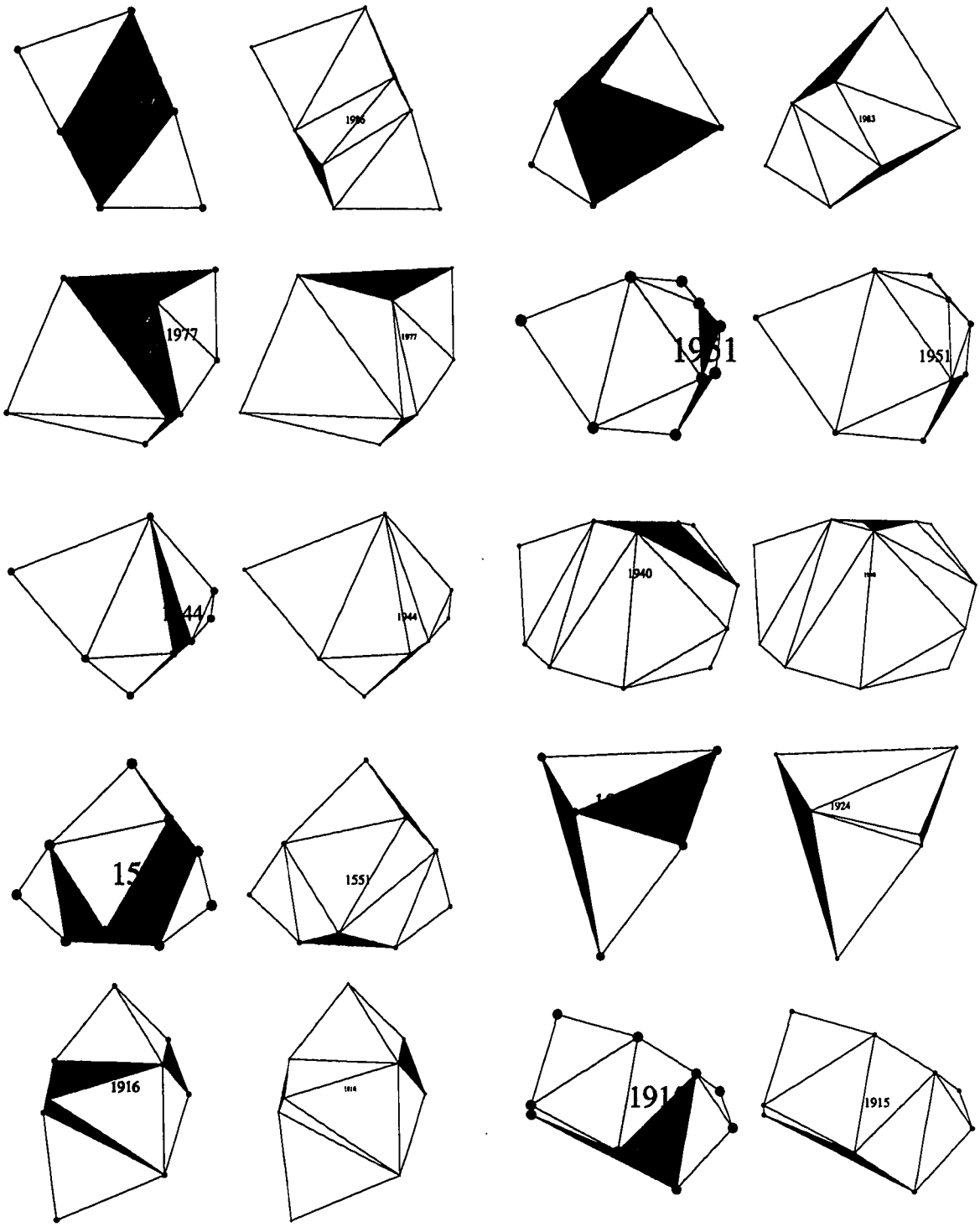


Figure A.3: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

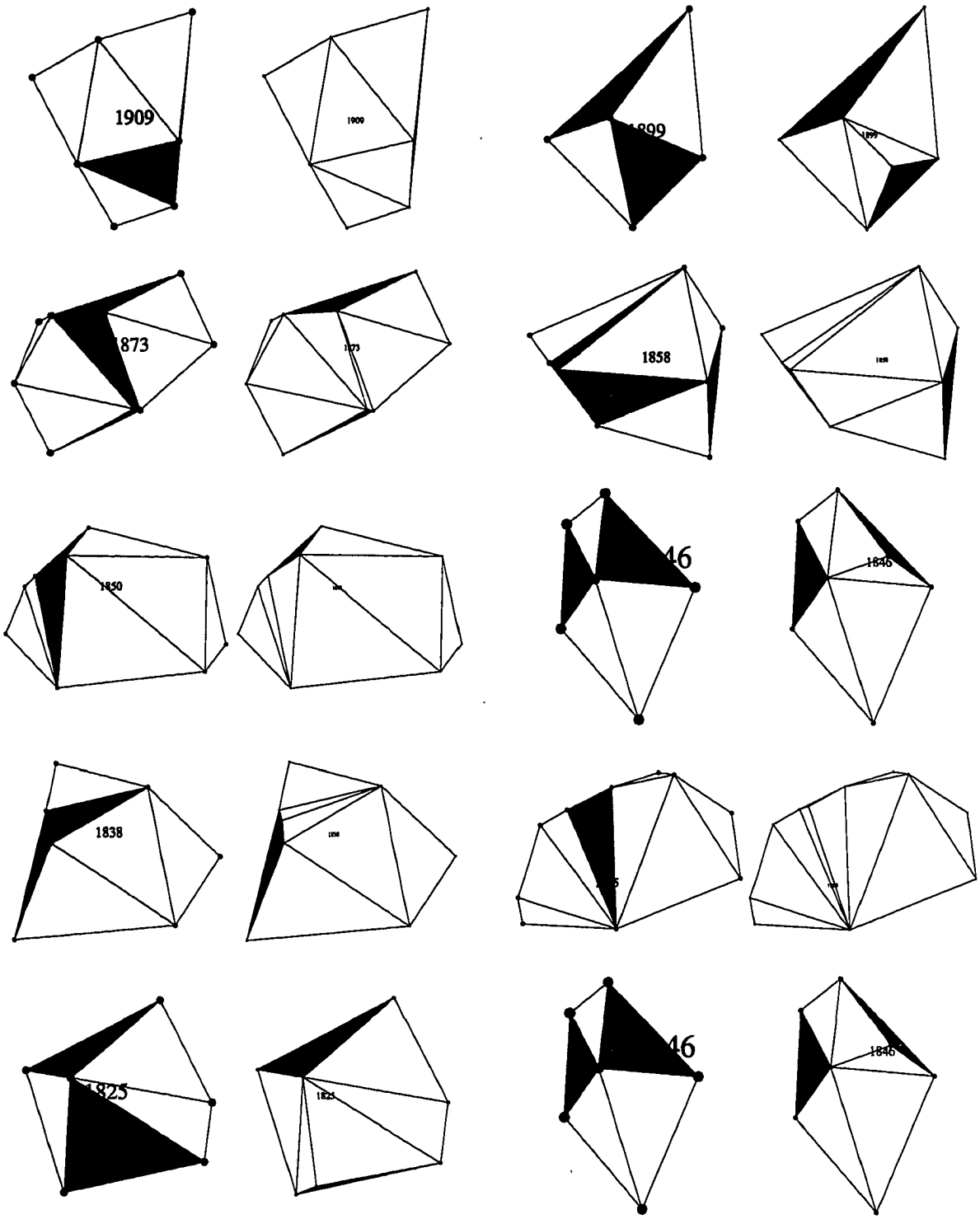


Figure A.4: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

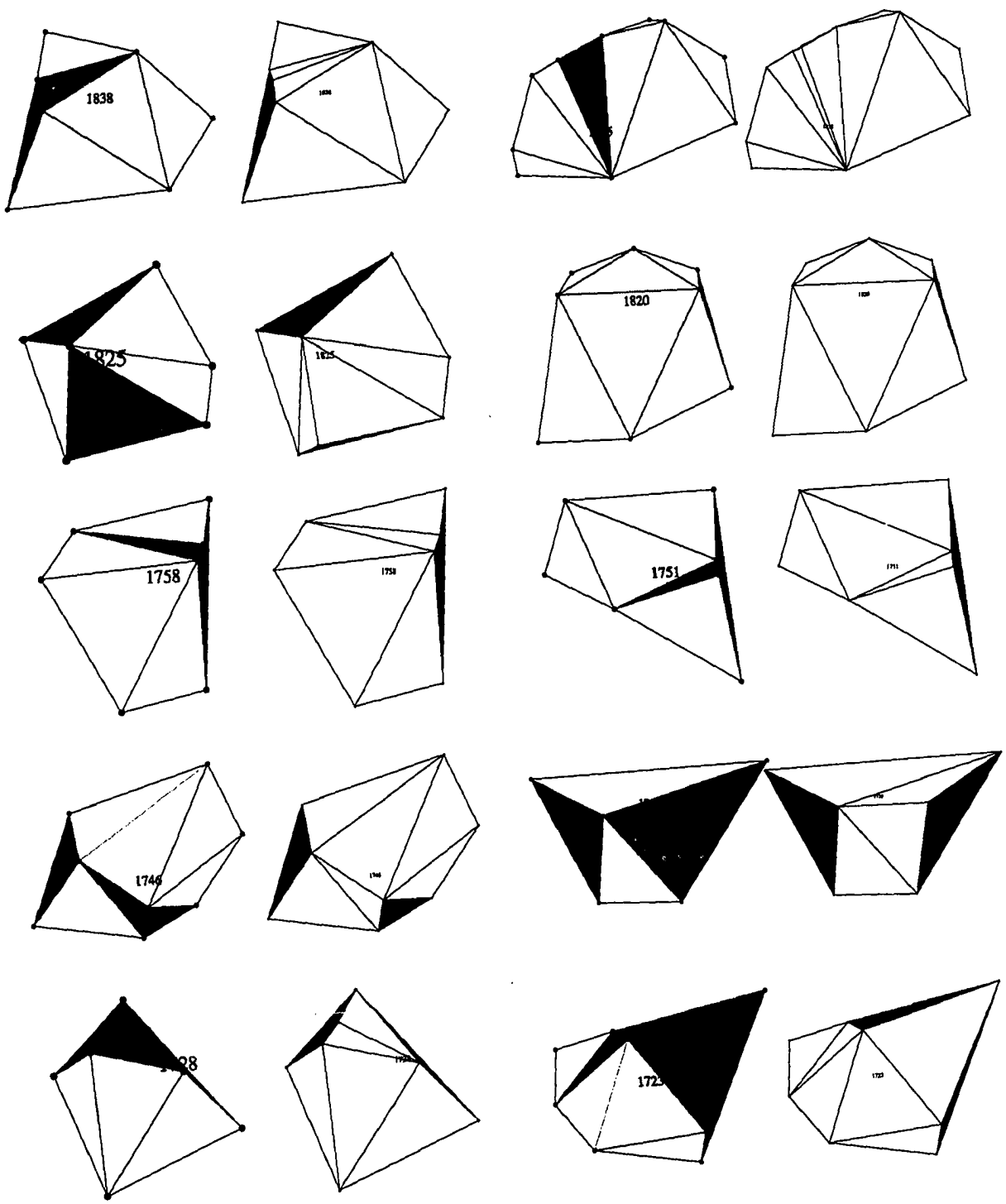


Figure A.5: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

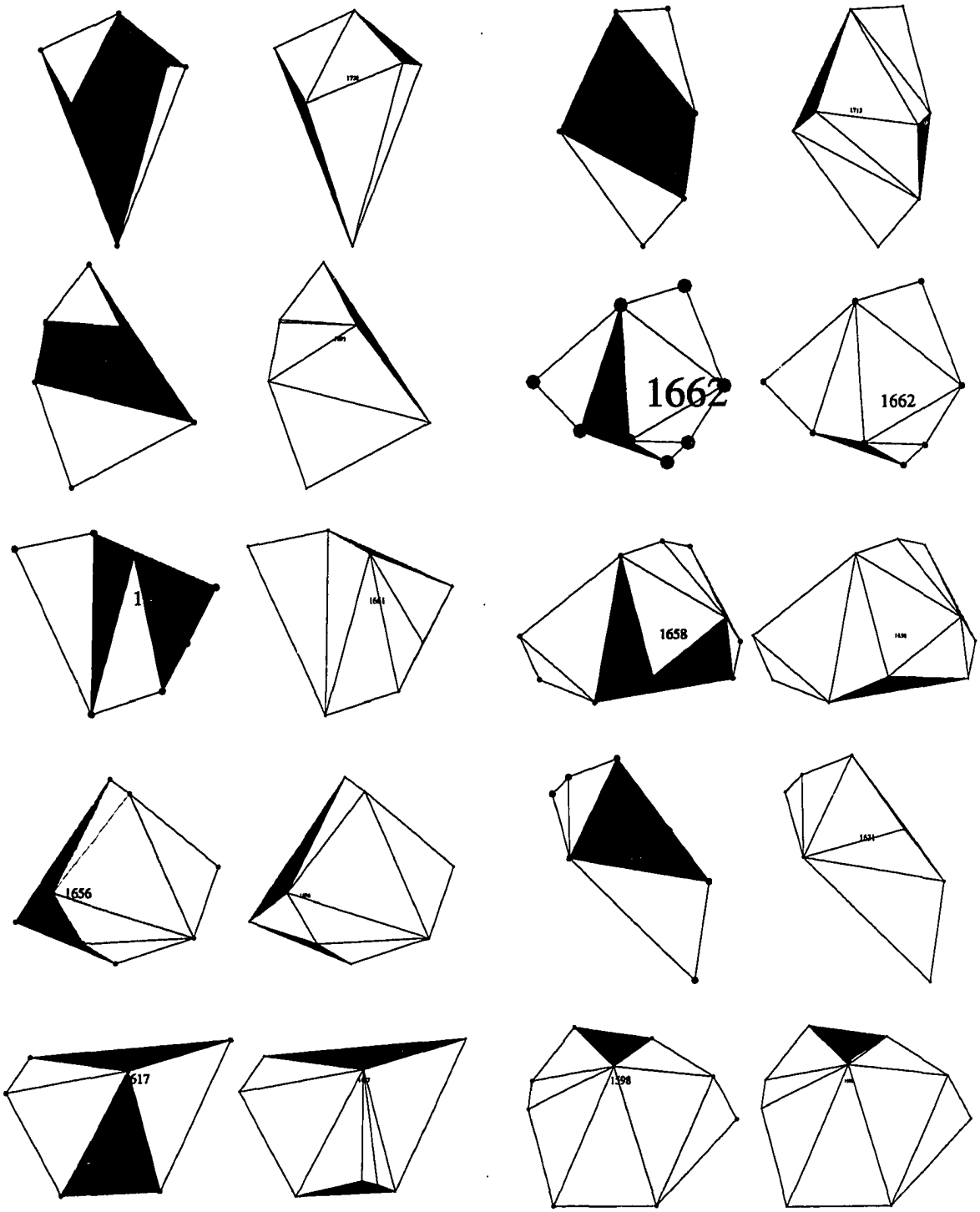


Figure A.6: 2D classification algorithm examples: Phase 1 (left), Phase 2 (right)

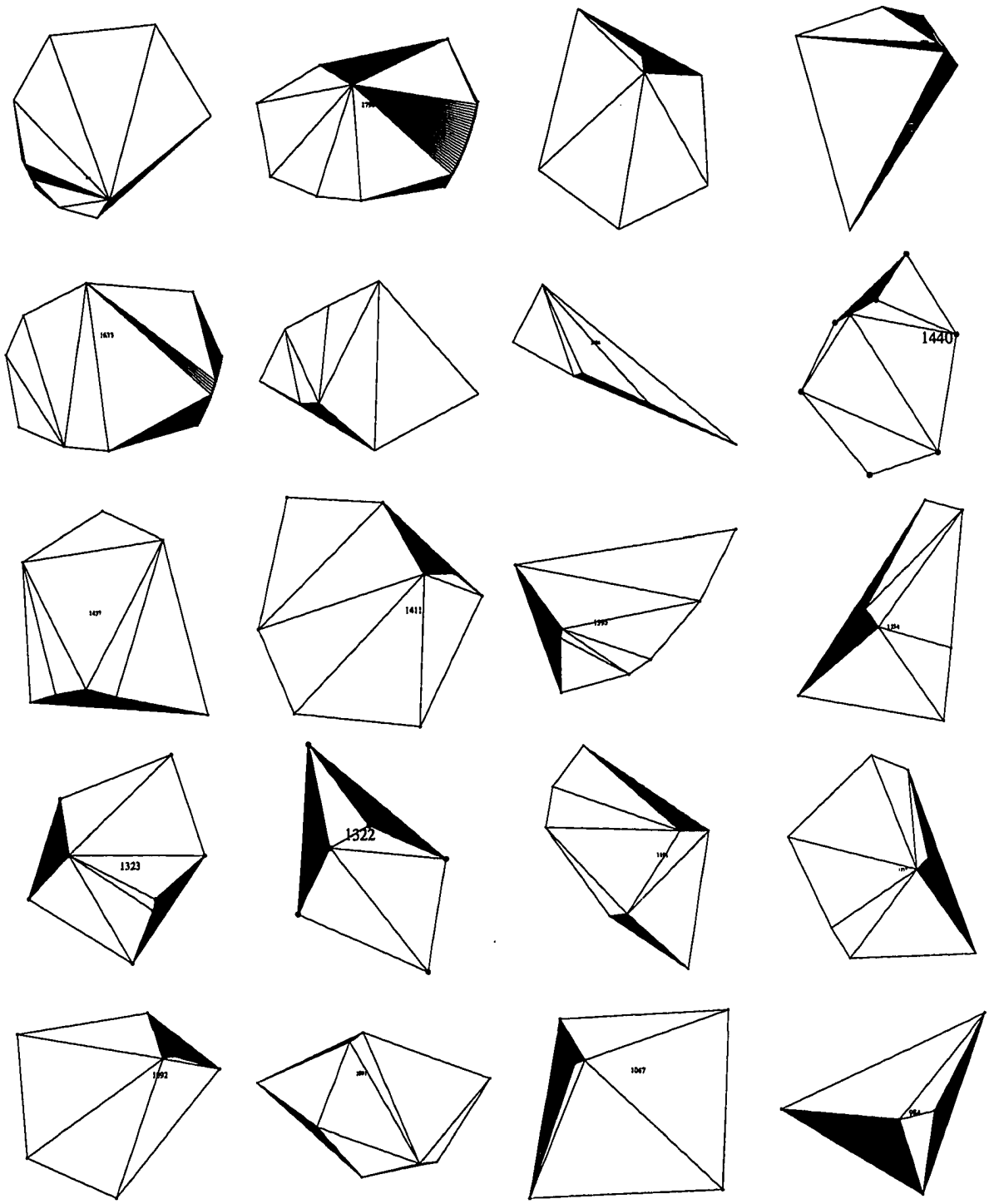


Figure A.7: 2D classification algorithm examples: Phase 2 final results

APPENDIX 2. COMMERCIAL INTEREST

Recent interest in decimation algorithms for general surface modeling has produced a number of commercial products. Cyberware of Monterey, California markets a 3D color digitizer with surface editing tools, NURBS surface software, and the Cymage mesh decimation algorithm. An advertisement in *IRIS Universe*, Summer 1994, boasts of reducing a $530k$ unstructured triangular mesh of a complex facial mask to a mere $25k$ triangles.

Engineering Animation Inc. of Ames, Iowa has expressed interest in the current research with regard to dynamic rendering applications for complex scenes. Vital Images Inc. of Fairfield, Iowa is exploring possible applications of the decimation algorithm for seismic mapping and volume rendering for petroleum and mining applications. A research group from the Massachusetts Institute of Technology is considering applying the algorithm to reduce data produced from a closed-loop design cycle: CAD/CAM model $\Rightarrow$ NC milling $\Rightarrow$ freeform sculpting $\Rightarrow$ NC slicing $\Rightarrow$ surface or volume decimation $\Rightarrow$ CAD/CAM model.

The relevance of the current generic research is evidenced by the March 1995 issue of *Computer Graphics World*, which dedicates a special supplement to "3D Digitizing for Engineering."

APPENDIX 3. SOFTWARE NOTES

Graphical visualization of the implemented decimation algorithm was accomplished by means of Silicon Graphic's Inventor, an object-oriented 3D graphics toolkit. The current research code is compatible with Inventor versions 1.1.2 and 2.0. The Graphical User Interface (GUI) is enhanced with the OSF/Motif library. Therefore the program can be ported to any platform running openGL and the X Window System. The 2D mesh and 3D surface plots contained herein were generated by dumping PostScript images from Inventor 1.1.2. Most of the 2D plots comparing analytical or experimental data to computational results were produced in XMGR. Other 2D illustrations were created using XFIG or Silicon Graphic's SHOWCASE drawing and presentation tool.