3D SURFACE MODELING UTILITIES FOR USE IN TCAD

by

John F. Sefler

Memorandum No. UCB/ERL M95/92

28 October 1995

COVER PAGE

3D SURFACE MODELING UTILITIES FOR USE IN TCAD

by

John F. Sefler

Memorandum No. UCB/ERL M95/92

28 October 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

3D Surface Modeling Utilities for use in TCAD

John F. Sefler

Department of Electrical Engineering & Computer Sciences
University of California at Berkeley
Berkeley, CA 94720

October, 1995

Abstract

As 3D lithography, etching, and deposition simulation tools such as SAMPLE-3D (from Berkeley), SPEEDIE (from Stanford), and EVOLVE (from Arizona State) are used to simulate more complicated device topologies, more specialized algorithms are needed to maintain well-behaved evolving surface triangulations. In this thesis a number of 3D geometric surface modeling utilities are developed. Some of the services include Delaunay triangulation, triangle decimation, surface smoothing, solid surface extraction, and thin triangle removal. Although intended for use in the TCAD community, these utilities can provide 3D geometric services to most any 3D geometric modeling package that reads and writes surfaces based on a winged-edge data structure.

Underlying the surface modeling utilities is a flexible query system that permits the utility procedures to inquire about local surface topography and whether or not a specified re-triangulation method will inadvertently alter the characteristic 3D shape of the surface. This allows surface advancement and surface modeling procedures to selectively use mesh modification methods to accomplish a desired goal. A simple application of the surface modeling utilities is the ability to extract solid geometries from a single triangulated surface for interconnect analysis using FASTCAP (from M.I.T.).

Among Berkeley's suite of integrated circuit process simulation programs known as SAMPLE-3D, the program NETCH houses the 3D surface advancement models for etching and deposition. To avoid lag time caused by saving and loading surface geometry, the supporting surface modeling utilities have been incorporated in the same framework as NETCH, and are available at the parser level of an instruction file.

Table of Contents

List of Figures

Acknowledgments

Above all, I wish to acknowledge my late father Dr. George F. Sefler. By profession, Dad was a philosopher, a magician, and a motivator. By nature, Dad was an educator who taught many how to achieve greatness! Near the top of his list of tools for greatness was an education. I recall the only bumper sticker he ever put on his car stating, “If you think education is expensive, try ignorance.” Maintaining the same beliefs as my father, I regard the education I have received from the students and faculty at Berkeley as priceless.

Among the faculty members, I first thank Prof. Neureuther who welcomed me into his TCAD research group. He introduced me to the problems addressed in this thesis. I am especially thankful to him for his kindness and willingness to put his ever-growing stack of paperwork aside whenever I walked through his office door. I also want to thank Prof. Carlo Séquin whose experience and interest in solving “really cool” geometry problems has rubbed off on me. Among all of my class assignments at Berkeley, those from Prof. Séquin were the most interesting; the kind that you can’t stop working on because you are self-driven to do more.

Among my classmates, I am thankful for the friendships of Derek Lee, with whom I took many classes; Robert Wang, who always answered my questions about TCAD tools; and John Helmsen, who helped me work out many of the intricate details of thin triangle removal. Other friends who I have had the pleasure to work with include Dr. Paul Hagouel, Alex Quezada, Ravi Gunturi, the Fall 1994 CS285 class on Procedural Generation of Objects, and the Spring 1995 CS294-5 class on Architectural CAD.

I also want to thank my family: Cecilia, George, and my wife Andrea. Andrea’s desire to return to Berkeley to complete her graduate work in chemistry gave me the opportunity to complement my engineering background by formally studying computer science. Now I can truly bridge the gap between engineers who write software and software engineers.

Finally, I am grateful to the Sematech Center of Excellence in Lithography and the Semiconductor Research Corporation whose financial support made this work possible.

Chapter 1:

Introduction

1.1 A Brief Historical Perspective

In the late 1980's and early 1990's, Kenny Toh [21] and Edward Scheckler [18] began the development of SAMPLE-3D, a suite of integrated circuit process modeling tools for predicting topological profiles in three dimensions. Their approach was to extend the 2D "string-based" [10] algorithm used in SAMPLE [14, 15] to 3D. In 2D, the "string" refers to a set of nodes and connecting segments representing the front of an advancing surface during etching and deposition simulation. In 3D, the "string" paradigm translated into a surface mesh of nodes and connecting segments to form triangles. For the most part, they successfully achieved their objective of translating the manufacturing process models into 3D solvers. During their development, unexpected topography challenges related to the triangulated surface mesh were unveiled. One such problem was handling the formation of "loops" during lithography simulation. "Looping" is a non-physical phenomenon whereby nodes in the surface start tracing paths through the resist that have already been etched; thereby creating topological loops. In 1992, an efficient "delooping" solution using an octtree data structure for spatial sorting was developed by John Helmsen [7].

Recognizing that additional geometric utilities are required to maintain a well-behaved surface mesh and prevent the onset of irregular triangulations during surface advancement, the work contained in this thesis began in late 1993.

1.2 The Development Framework for the Utilities

To aid in the prediction of more intricate topographies by integrated circuit process modeling tools, a reliable system capable of providing geometry services that condition the surface in a desirable way is necessary. This system must be able to correct surface triangulations when irregularities arise as well as prevent the onset of irregularities. Moreover, these utilities will operate on surfaces evolving in time, thus geometric modifications must not destroy data encapsulated in the elements of the mesh that relate to the surface's advancement.

With these requirements in mind, a geometrical query based system including local re-triangulation operators was built. This system provides the foundation for all the surface modeling utilities presented in this thesis and can be used to build additional utilities as needed.

Because the algorithms have been implemented as utility services, they can be used independently or in combinations to help correct problematic irregularities in a mesh's triangulation. In addition to TCAD simulators, other scientific applications can utilize these algorithms in one of two ways. Either the corresponding software modules can be extracted from NETCH, the program in which these algorithms have been bundled, or NETCH can be used as a mini-server through the exchange of geometry files that are loaded, operated on, and saved.

1.3 Manuscript Overview

Some of the surface modeling utilities presented in this thesis include:

- Delaunay Triangulation (2 $^1/_2$ D) - a method of re-triangulating the surface so as to make the triangle facets as equilateral as possible without moving nodes.
- Decimation of Triangles - a method of reducing geometric nodal information without altering the distinct features in the surface topology.
- Thin Triangle Removal - a method of removing acute triangles without altering the surface topology.
- Triangle Smoothing - a way of reducing the surface roughness by flattening the

dihedral angle between adjacent triangles through re-triangulation of fixed nodal points.

- Node Smoothing - a way of reducing the non-planarity of the surface by repositioning nodes to geometric averages of their surrounding neighbors.
- Solid Extraction - a process whereby bounded enclosing surfaces are extracted from a single triangulated surface using "cutting planes".

Common to all the utilities, particular attention has been paid to preserving important geometrical features such as edges, ridges, and corners. This is achieved by building decision trees consisting of geometrical queries to decide correctly which local mesh modification routines to use.

1.4 Thesis Organization

This thesis is organized in a "bottom up" manner. Beginning in Chapter 2, an introductory background is given to the localized surface operations and geometrical queries used to build the surface modeling utilities. Because of the reference-like nature of the material covered in Chapter 2, the surface utility chapters beginning with Chapter 3 could be read first.

The algorithms governing each utility constitute the subject of Chapters 3 through 7. When relevant, references are made to the supporting query routines in Chapter 2. At that time, the query material may be more relevant to the reader.

Following the "bottom up" organization, Chapter 8, presents results from a small application employing the surface modeling utilities; the computation of capacitance matrices from a rigorously simulated 3D device topology.

At the end of a few chapters, a brief discussion about improving the efficiency of an algorithm has been included. Should the surface modeling utilities be enhanced in the future, these sections provide guidelines for a better re-implementation or modification of the software.

Finally, the Appendix contains a listing of additional simulator commands recognized by NETCH, the SAMPLE-3D program housing the 3D surface advancement routines and the supporting surface modeling utilities. Using an instructional file containing the appropriate simulator commands, NETCH can be used as a mini-server for the surface modeling utilities.

Chapter 2:

Local Surface Queries and Triangulation

2.1 Introduction

Following the "bottom up" organization of this thesis, this chapter begins with a presentation of the fundamental data structures used to store a triangular surface mesh and then leads into the assembly of many geometry based queries used to characterize the regularity of a surface's triangulation. After identifying the different surface, triangle, segment, and node classifications, local re-triangulation methods with accompanying queries about their affect on characteristic topography destruction are presented. This system provides the bottom layer upon which all of the utilities discussed in subsequent chapters are built.

Each of the queries presented corresponds to a public function in the software. As such, this chapter has a reference-like organization and is intended for use by subsequent software developers. For this reason, casual readers may skip this chapter.

2.2 Linked Lists and Winged-Edge Data Structures

All of the surface modeling utilities developed in this thesis are based on doubly linked lists of winged-edge data structures in the C programming language. In the context of SAMPLE-3D, these data structures were first presented by Toh [22] and Scheckler [18]. For completeness, the essential contents of the data structures are also presented here.

To represent a single triangulated surface, three linked lists consisting of triangle structures, segment structures, and node structures are used. The triangle structure contains a minimum of three pointers each to a segment structure. The segment structure contains a minimum of two pointers each to a node structure. And, the node structure contains a minimum of three pointers to data structures containing the x, y, and z coordinate positions of the surface nodes. Although this is more than the minimum information needed to define the connectivity of a surface, redundant information is useful for faster access to elemental data.. There are additional pointers within the segment structures to the neighbor triangles, and lists of adjacent segments and adjacent triangles within the node structures. Figure 1 shows a schematic of the doubly linked winged-edge data structures, and Fig. 2 contains a small excerpt of the C data structures. The triangles,

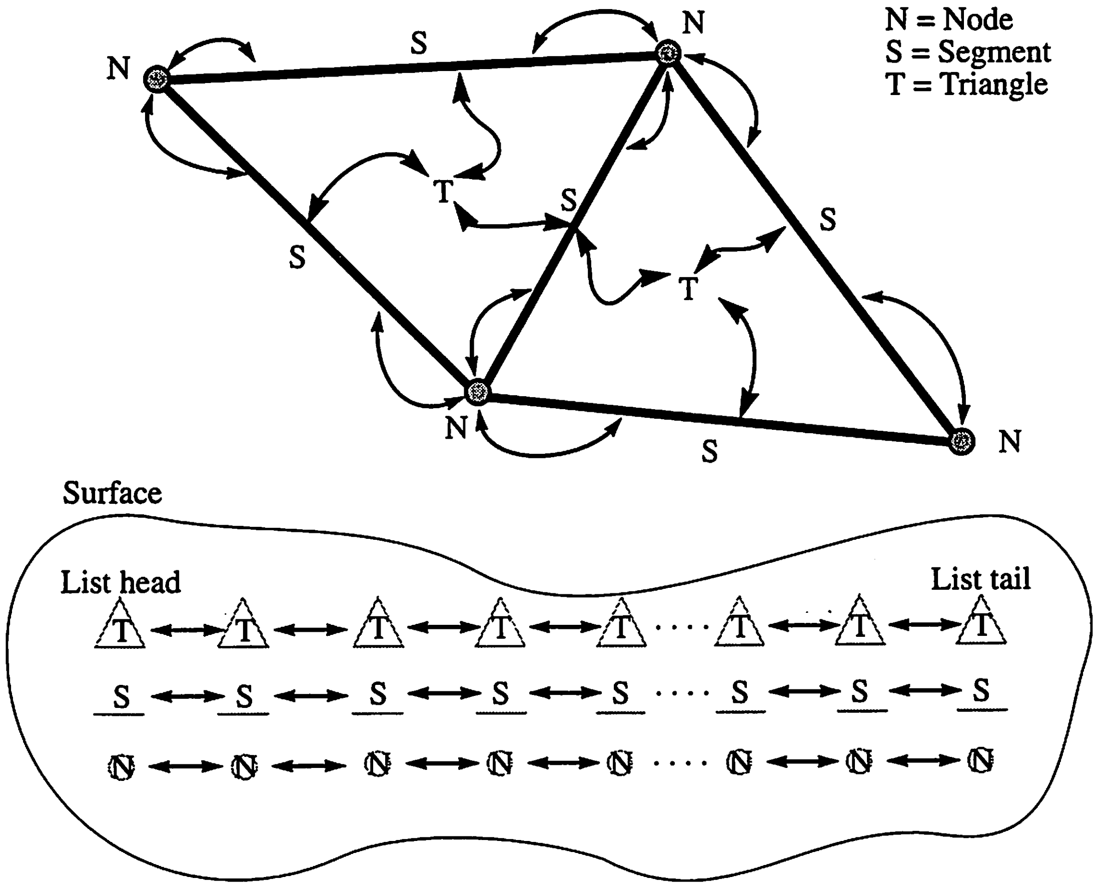


Figure 1 Graphical schematic of the winged-edge data structure and doubly linked lists of triangles, segments, and nodes that comprise a surface.

segments, and nodes individually comprise the elemental components of the surface.

```
/* 3D data */
typedef struct data_strct {
   double x; double y; double z;
} data;


/* used for linked-lists of adjacent
   segments to nodes */
typedef struct adjsegm_strct {
   struct segm_strct *adjacent_segm;
   struct adjsegm    *next_adjsegm;
} adjsegm, *adjsegmptr;


/* used for linked-lists of adjacent
   triangles to nodes */
typedef struct adjtria_strct {
   struct tria_strct *adjacent_tria;
   struct adjtria *next_adjtria;
} adjtria, *adjtriaptr;


/* Node Data Structure */
typedef struct node_strct {
   int  ID;
   struct node_strct *next;
   struct node_strct *prev;

   data        coord;
   adjsegmptr  adjsegm_head;
   adjtriaptr  adjtria_head;
} node, *nodeptr;
```

```
/* Segment Data Structure */
typedef struct segm_strct {
   int    ID;
   struct segm_strct *next;
   struct segm_strct *prev;

   nodeptr n1;
   nodeptr n2;
   struct tria_strct *nbrtri[2];
} segm, *segmptr;


/* Triangle Data Structure */
typedef struct tria_strct {
   int    ID;
   struct tria_strct *next;
   struct tria_strct *prev;

   segmptr s1;
   segmptr s2;
   segmptr s3;
} tria, *triaptr;
```

Figure 2 Representation of the winged-edge data structure in C.

2.3 Geometrical Query Approach

In this section, numerous geometric query procedures and heuristics are presented. Fundamental to the success of the surface modeling utilities presented in the succeeding chapters is the ability to make inquiries about the local topology of the surface triangulation. Whether a triangle is small, thin, short-sided, or equilateral may affect the decision taken by an algorithm whether or not to remove, merge, or retain the elemental components of the surface.

2.3.1 Surface Queries

The following sections contain descriptions of queries used to detect various surface topology conditions and calculations.

2.3.1.1 Is Surface Continuous?

A continuous surface contains a single group of connected triangles. It is possible however, to have a single surface represented by a linked-list of triangles containing groupings that topologically are not connected to each other. This is called a disconnected surface and is depicted in Fig. 3. To detect a discontinuous surface, every triangle in the

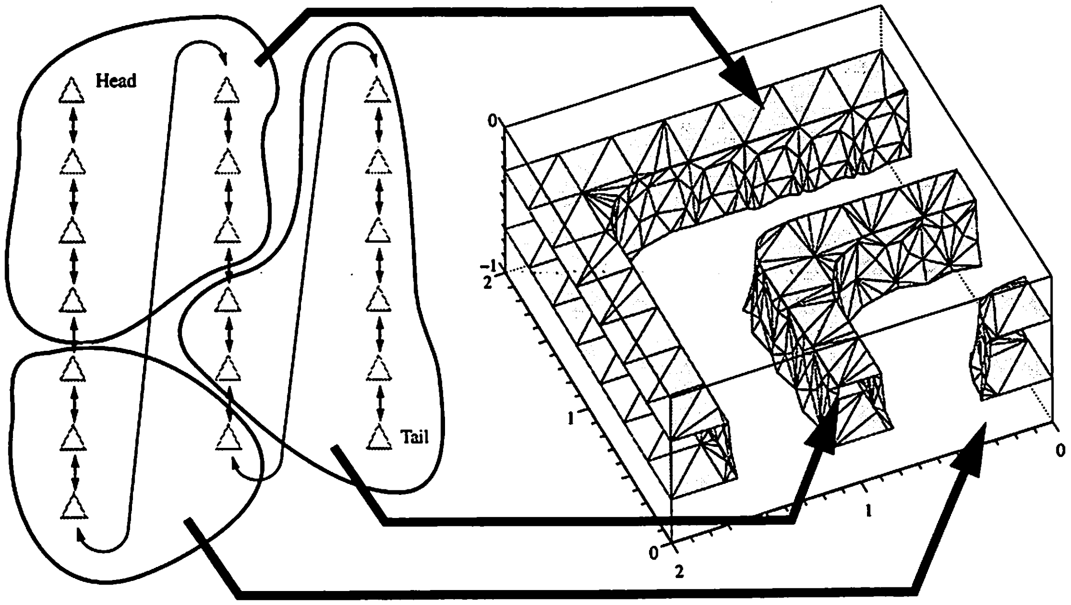


Figure 3 A linked-list and topological representation of a single discontinuous surface containing 3 groups.

linked-list is marked with a 0 flag. Then the head of the triangle list is marked with a 1.

Using a breadth-first search method, all of the list head's neighbors are also marked with a 1. At the end, if there are any triangles left with a 0 flag, then the surface is topologically disconnected. It is not continuous.

To identify the number of continuous groups within a discontinuous surface, one of the remaining triangles marked with a 0 is now marked with a 2 and the breath-first search is repeated by marking all adjacent triangles to it with a 2. As long as an unmarked triangle remains in the linked-list, the breadth-first search is repeated incrementing the marking flag by one each time. Once all triangles have been positively marked, the number of continuous groupings in the discontinuous surface is equal to the largest marking flag.

To treat each of the continuous groups as independent surfaces, a simple bin sorting of the triangle, segment, and nodes is performed. First the nodes and segments are marked with the same flag as their adjacent triangles. Then the elements whose markings are greater than 1, are deleted from the original linked-list and added to its own linked-list corresponding to the marking flags.

2.3.1.2 Is Surface Enclosing?

An enclosing surface is analogous to a balloon. It has an inside and outside. When all of the segments in the surface have exactly two legitimate neighbor triangles (not NULL), then the surface is enclosing.

2.3.1.3 Is Surface Self-Intersecting?

A self-intersecting surface is one that contains a subset of triangles that overlap in three space. An example resulting from the advancement of a surface during resist development using a ray-trace algorithm is depicted in Fig. 4. Self-intersection of a surface was investigated by John Helmsen [7] who is responsible for the software development of loop detection and removal. For completeness, the self-intersection query is included here.

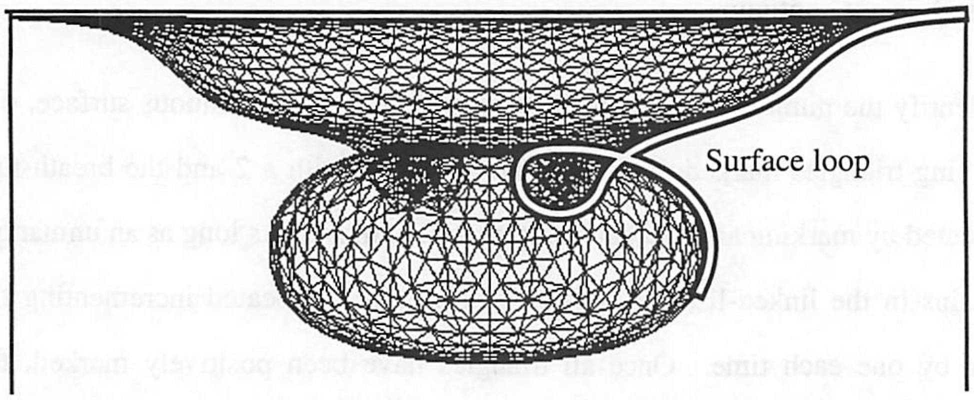


Figure 4 A self-intersecting surface (one of the interior loop is outlined).

To detect if a surface is self-intersecting, each of the triangles in the surface are inserted into an octtree where they are spatially sorted and checked for intersections with nearby triangles that are not already directly connected by common segments. If any intersections are detected, then the surface is self-intersecting.

2.3.1.4 Surface Area?

To calculate the total surface area, the areas of all triangles in the surface are calculated and summed.

2.3.2 Triangle Queries

The following sections contain descriptions of queries used to detect various properties and geometric conditions related to the triangles in a surface. Fig. 5 contains a schematic representation of many triangle classifications discussed in this section.

Some of the classifications are based on the notion of an ideal segment length introduced by Toh [22] and Scheckler [18]. An ideal segment length is specified by the

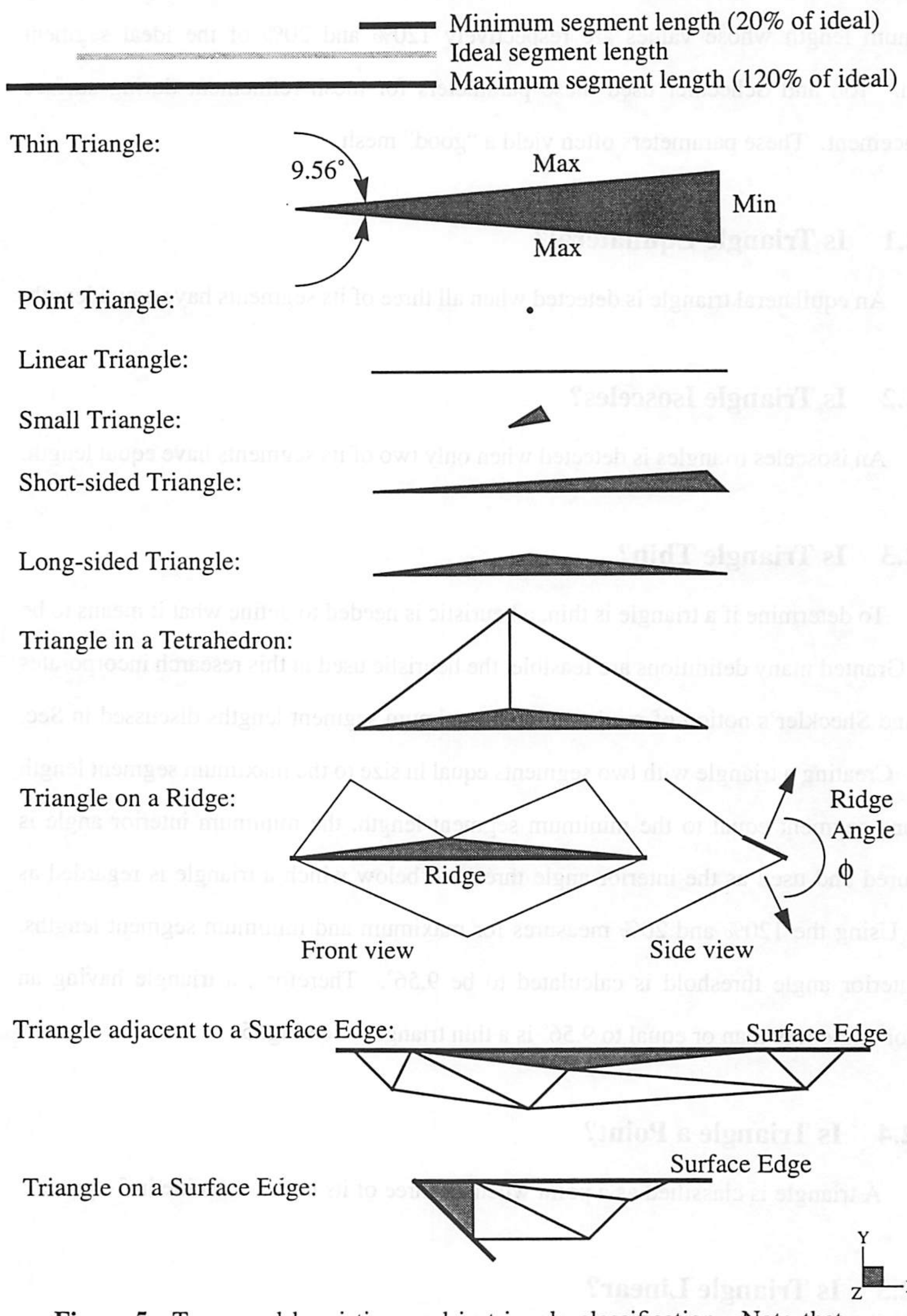


Figure 5 Types and heuristics used in triangle classification. Note that any given triangle may satisfy more than one classification.

mesh density requested by the user. Corresponding to this length are a maximum and minimum length whose values are respectively 120% and 20% of the ideal segment length. Toh and Scheckler used these parameters for mesh refinement during surface advancement. These parameters often yield a "good" mesh.

2.3.2.1 Is Triangle Equilateral?

An equilateral triangle is detected when all three of its segments have equal length.

2.3.2.2 Is Triangle Isosceles?

An isosceles triangles is detected when only two of its segments have equal length.

2.3.2.3 Is Triangle Thin?

To determine if a triangle is thin, a heuristic is needed to define what it means to be thin. Granted many definitions are feasible, the heuristic used in this research incorporates Toh and Sheckler's notion of minimum and maximum segment lengths discussed in Sec. 2.3.2. Creating a triangle with two segments equal in size to the maximum segment length and one segment equal to the minimum segment length, the minimum interior angle is measured and used as the interior angle threshold below which a triangle is regarded as thin. Using the 120% and 20% measures for maximum and minimum segment lengths, the interior angle threshold is calculated to be 9.56°. Therefore, a triangle having an interior angle less than or equal to 9.56° is a thin triangle. See Fig. 5.

2.3.2.4 Is Triangle a Point?

A triangle is classified as a point when all three of its nodes coincide in 3-space.

2.3.2.5 Is Triangle Linear?

A linear triangle has three nodes positioned along a line in 3-space. If it has zero area and is not a point triangle, then it is linear.

2.3.2.6 Is Triangle Small?

A small triangle is composed of three segments whose lengths are all less than the minimum segment length.

2.3.2.7 Is Triangle Short-sided?

A triangle is classified as short-sided when the ratio between its longest segment and shortest segment exceeds the ratio of the maximum segment length to minimum segment length (i.e. 6:1).

2.3.2.8 Is Triangle Long-sided?

A long-sided triangle is one that does not satisfy the short-sided criteria. For example, an equilateral triangle with a maximum to minimum segment length ratio of 1:1 is considered to be long-sided.

2.3.2.9 Is Triangle on a Ridge?

A triangle on a ridge has at least one segment classified as a ridge segment. Refer to Sec. 2.3.3.6 (Is Segment on a Ridge?).

2.3.2.10 Is Triangle on 3 Ridges?

When all three segments of a triangle are classified as ridge segments, then the triangle is on three ridges.

2.3.2.11 Is Triangle in a Tetrahedron?

A triangle is classified as being in a tetrahedron when one of its nodes, called the tetrahedron node, has three neighboring segments and three neighboring triangles. The tetrahedron description comes from the hypothetical volume mesh equivalent to the surface mesh. When three triangles and three segment exclusively share a common node, then the three triangles would be three of the four faces of the solid tetrahedron. The fourth face of the tetrahedron would be created by the three segments opposite the tetrahedron node in the three triangles. As will be pointed out in Sec. 2.4.1.6 (Will Tetrahedron Removal Alter Topography?), the three triangles sharing the "tetrahedron node" can be replaced by the hypothetical fourth face of the tetrahedron to reduce the granularity of the surface triangulation or to eliminate thin triangles provided the surface topology is not significantly altered during the removal.

2.3.2.12 Is Triangle on a Surface Edge?

A triangle is on a surface edge when two of its segments are on a surface edge. Refer to Sec. 2.3.3.4 (Is Segment on a Surface Edge?).

2.3.2.13 Is Triangle Adjacent to a Surface Edge?

A triangle is adjacent to a surface edge when only one of its segments is on a surface edge.

2.3.2.14 Is Triangle in a Surface Corner?

A triangle in the corner of a surface has two segments positioned on a surface edge, and these two segments are positioned in orthogonal planes coinciding with the principle axes.

2.3.3 Segment Queries

The following sections contain descriptions of queries used to detect various properties and geometric conditions related to the segments in a surface.

2.3.3.1 Is Segment in an X-Plane?

When the difference between the x coordinate values of the two nodes of a segment is zero, then the segment lies in an x-plane.

2.3.3.2 Is Segment in an Y-Plane?

When the difference between the y coordinate values of the two nodes of a segment is zero, then the segment lies in a y-plane.

2.3.3.3 Is Segment in an Z-Plane?

When the difference between the z coordinate values of the two nodes of a segment is zero, then the segment lies in a z-plane.

2.3.3.4 Is Segment on a Surface Edge?

A segment on a surface edge has only one valid neighbor triangle. The other is `NULL`. This assumes that a segment can only have two neighboring triangles.

2.3.3.5 Is Segment on a Surface Fold?

A surface fold is analogous to a perfectly creased sheet of paper folded back onto itself. When the dihedral angle between the adjacent triangles of a segment approaches 180°, the segment is considered to be on a surface fold.

2.3.3.6 Is Segment on a Ridge?

A segment is classified as a ridge segment when the dihedral angle between the normal vectors of the triangle faces adjacent to the segment exceeds some threshold. The threshold is called the ridge angle and can be changed to any default value (such as 45°). The assumption made when a ridge segment is detected, is that the surface topology across the segment is significantly non-planar and therefore the segments provides a significant role in defining the characteristic shape of the surface at that point. As such, triangulation algorithms should be wary of the topography across this segment. See Fig. 5 for a schematic.

2.3.3.7 Is Segment at the End of a Ridge?

When a ridge segment has one or both of its nodes at the end of a ridge (Sec. 2.3.4.6), then the segment is at the end of a ridge.

2.3.4 Node Queries

The following sections contain descriptions of queries used to detect various properties and geometric conditions related to the nodes in a surface.

2.3.4.1 Is Node on a Surface Edge?

If any of the segments connected to the node are on a surface edge, then the node is also on a surface edge.

2.3.4.2 Is Node on a Surface Corner?

Within NETCH, the parameters *xmin*, *xmax*, *ymin*, and *ymax* define the boundaries of the rectangular simulation region. If a node coincides with one of the coordinate pairs of the simulation boundaries, then the node is "on a surface corner".

2.3.4.3 Is Node on a Ridge?

A node is "on a ridge" when it is connected to a segment that is "on a ridge." Refer to Sec. 2.3.3.6 (Is Segment on a Ridge?).

2.3.4.4 Is Node on a Ridge Corner?

A node formed by at least three merging ridges is "on a ridge corner." For example, a node on the corner of a box would be considered to be "on a ridge corner."

Note that this query attempts to discriminate between a node positioned on the corner of macroscopic ridge and microscopic ridge. A microscopic ridge is defined in Sec. 2.3.4.5 (Is Node on a Micro Ridge Corner?). This query views a ridge corner from a macroscopic perspective; the ridge segments emanating from the node in question must continue to follow a well defined sequence of additional ridge segments beyond one segment length away from the node.

2.3.4.5 Is Node on a Micro Ridge Corner?

A node formed by at least three merging ridges is "on a micro ridge corner." Unlike the query in Sec. 2.3.4.4 (Is Node on a Ridge Corner?), this query does not attempt to identify how well behaved the ridges emanating from the node are.

2.3.4.6 Is Node at the End of a Ridge?

A node is at the end of a ridge when only one ridge segment is attached to the node. Refer to Sec. 2.3.3.6 (Is Segment on a Ridge?).

2.3.4.7 Is Node on a Fold Back?

When a node is attached to exactly two different segments and two different triangles, then this identifies a situation where the node is the peak of adjacent fold back triangles. Refer to Sec. 2.4.2.2 (Is Triangle in a Fold-Back?).

2.4 Local Re-connectivity Procedures

Having established numerous geometrical queries, systematic removal methods have been implemented for removing triangles. The decision as to which method to employ in the local re-triangulation of a surface mesh is dependent on the objective of the governing algorithm. This is the subject of the subsequent chapters. In the remainder of this chapter, the removal methods are presented as well as a few degenerate surface triangulations that can and do occur.

2.4.1 Triangle Removal Queries and Methods

In this section, the most common methods for removing a targeted triangle are presented. Accompanying each removal method are the necessary queries concerning whether or not the chosen removal method will alter topological features in a surface. Fig. 6 contains a collection of the removal methods.

2.4.1.1 Will Merging 1 Node Alter Topography?

As shown in Fig. 6, merging 1 node is the process of pushing the elemental neighbors of one node into a second node of a common triangle. The removal method actually removes two triangles - the triangle of interest and its adjacent triangle. Usually, merging one node is used to remove a short-sided triangle. Before using this method, a number of geometric queries need to be answered. Referring to Fig. 7, responding "yes" to any of the following pseudo-C-code questions means that merging a single node will alter the surface topology and therefore should *not* be used.

```
•  if ( is_node_on_a_surface_edge(N1)      &&
         is_node_on_a_surface_edge(N3)      &&
         is_node_on_a_ridge(N1)             &&
         is_node_on_a_ridge(N3))            return(YES);

•  if ( is_node_on_a_ridge(N1)             &&
         is_node_on_a_ridge(N3)             &&
        !is_segment_on_a_ridge(S2))         return(YES);

•  if ( is_node_on_a_surface_edge(N1)      &&
         is_node_on_a_surface_edge(N3)      &&
        !is_segment_on_a_surface_edge(S2))  return(YES);

•  if ( is_node_on_a_surface_edge(N1)      &&
         is_node_on_a_ridge(N3)             &&
        !is_segment_on_a_surface_edge(S2)   &&
        !is_segment_on_a_ridge(S2))         return(YES);
```

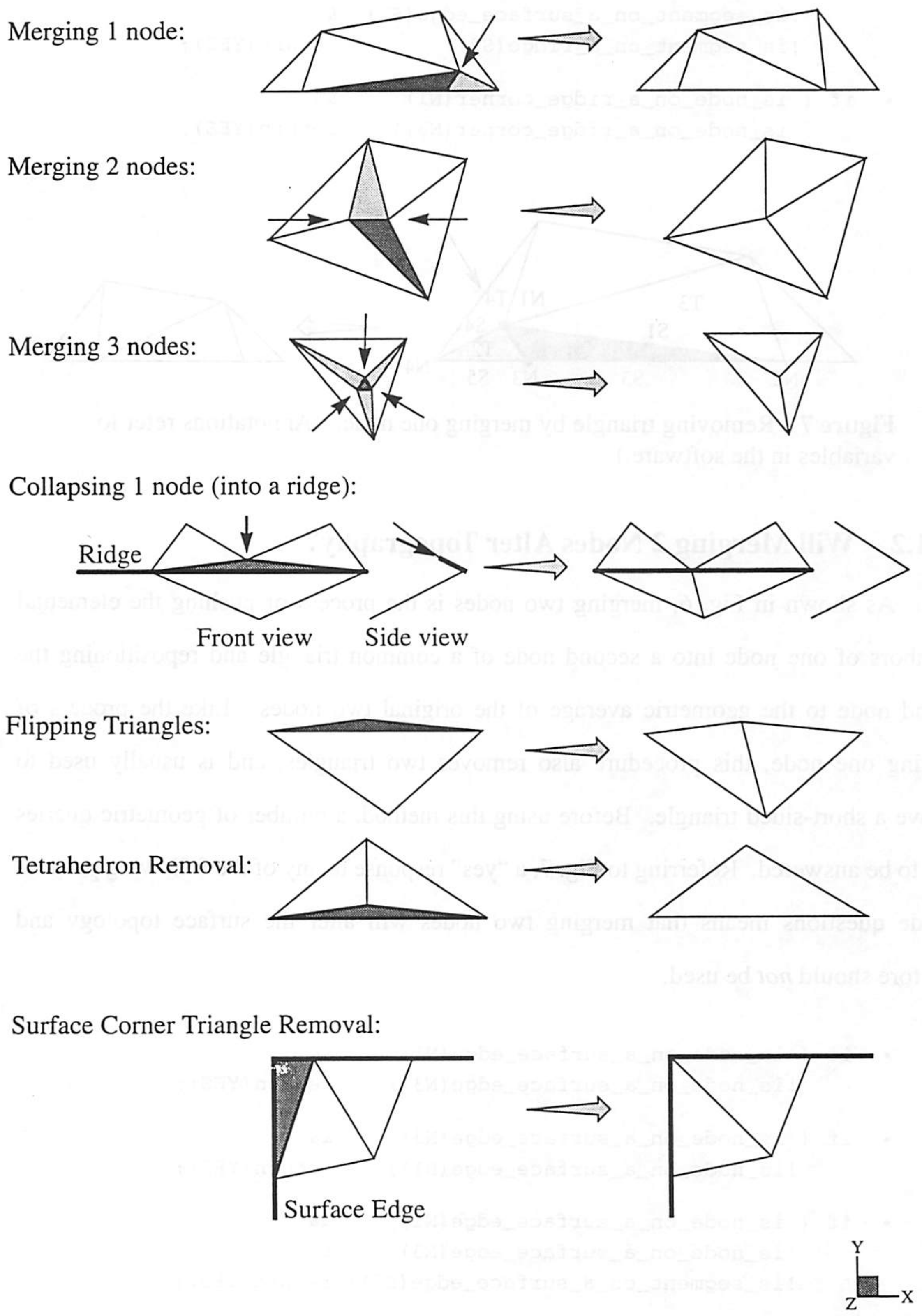


Figure 6 Triangle removal methods.

```
•  if ( is_node_on_a_surface_edge(N3)        &&
        is_node_on_a_ridge(N1)               &&
       !is_segment_on_a_surface_edge(S2)     &&
       !is_segment_on_a_ridge(S2))           return(YES);

•  if ( is_node_on_a_ridge_corner(N1)        &&
        is_node_on_a_ridge_corner(N3))       return(YES);
```

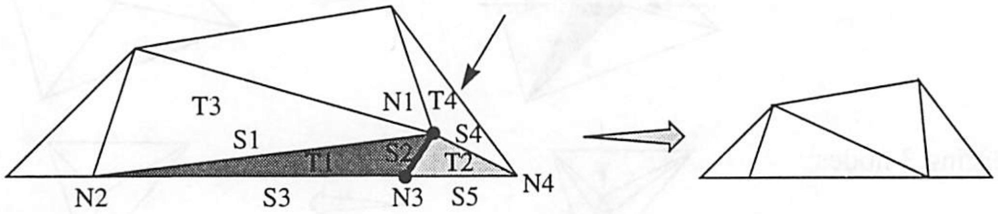


Figure 7 Removing triangle by merging one node. (Annotations refer to variables in the software.)

2.4.1.2 Will Merging 2 Nodes Alter Topography?

As shown in Fig. 6, merging two nodes is the process of pushing the elemental neighbors of one node into a second node of a common triangle and repositioning the second node to the geometric average of the original two nodes. Like the process of merging one node, this procedure also removes two triangles, and is usually used to remove a short-sided triangle. Before using this method, a number of geometric queries need to be answered. Referring to Fig. 7, a "yes" response to any of the following pseudo-C-code questions means that merging two nodes will alter the surface topology and therefore should *not* be used.

```
•  if ( is_node_on_a_surface_edge(N1)        &&
       !is_node_on_a_surface_edge(N3))       return(YES);

•  if ( is_node_on_a_surface_edge(N3)        &&
       !is_node_on_a_surface_edge(N1))       return(YES);

•  if ( is_node_on_a_surface_edge(N1)        &&
        is_node_on_a_surface_edge(N3)        &&
       !is_segment_on_a_surface_edge(S2))    return(YES);

•  if ( is_node_on_a_surface_edge(N1)        &&
        is_node_on_a_ridge(N1))             return(YES);

•  if ( is_node_on_a_surface_edge(N3)       &&
        is_node_on_a_ridge(N3))             return(YES);

•  if ( is_node_on_a_ridge(N1)              &&
       !is_node_on_a_ridge(N3))             return(YES);

•  if ( is_node_on_a_ridge(N3)              &&
         !is_node_on_a_ridge(N1))           return(YES);

•  if ( is_node_on_a_ridge(N1)              &&
        is_node_on_a_ridge(N3)              &&
       !is_segment_on_a_ridge(S2))          return(YES);

•  if ( is_node_on_a_ridge_corner(N1))      return(YES);

•  if ( is_node_on_a_ridge_corner(N3))      return(YES);

•  if ( is_node_on_a_surface_corner(N1))    return(YES);

•  if ( is_node_on_a_surface_corner(N3))    return(YES);

•  if ( is_segment_on_a_ridge(S2)           &&
      ( num_ridge_segm_connected_to_node(N1) !=
        num_ridge_segm_connected_to_node(N3))) return(YES);
```

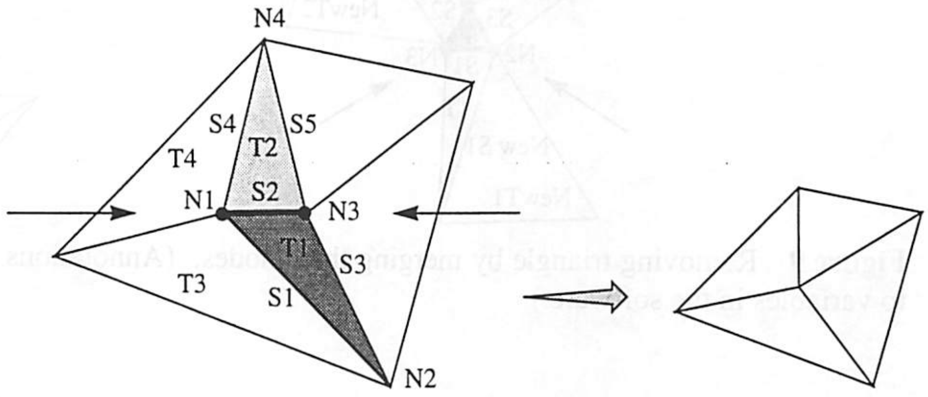


Figure 8 Removing triangle by merging two nodes. (Annotations refer to variables in the software.)

2.4.1.3 Will Merging 3 Nodes Alter Topography?

As shown in Fig. 6, merging three nodes is the process of replacing the unwanted triangle by a single node at its geometric center. This is the most greedy procedure as it removes a total of four triangles; itself and its three neighbors. This removal method is used only in situations where the triangle does not border any significant deviations in topology. Referring to Fig. 9, a "yes" response to any of the following pseudo-C-code questions means that merging three nodes will alter the surface topology and therefore should *not* be used.

```
• if (is_node_on_a_surface_edge(N1)) return(YES);
• if (is_node_on_a_surface_edge(N2)) return(YES);
• if (is_node_on_a_surface_edge(N3)) return(YES);
• if (is_node_on_a_ridge(N1))        return(YES);
• if (is_node_on_a_ridge(N2))        return(YES);
• if (is_node_on_a_ridge(N3))        return(YES);
```

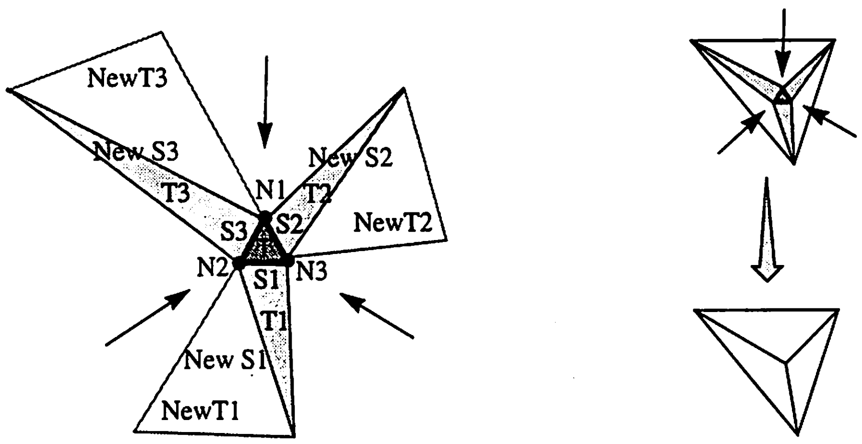


Figure 9 Removing triangle by merging three nodes. (Annotations refer to variables in the software.)

2.4.1.4 Will Collapsing 1 Node Alter Topography?

As shown in Fig. 6, collapsing one node is the process of pushing the node opposite the longest segment of the triangle along its minimum altitude to a new position along the longest segment. Then the triangle connectivity is altered so as to split the longest segment into two smaller segments as well as splitting the neighbor triangle into two coplanar triangles equal in area to the original. This procedure does not remove triangles; rather, provides a local re-triangulation. It is primarily used on unwanted triangles that are long-sided. Referring to Fig. 9, a "yes" response to any of the following pseudo-C-code questions means that collapsing one node will alter the surface topology and therefore should *not* be used.

```
• if (is_node_on_a_surface_edge(N2)) return(YES);
• if (is_node_on_a_ridge(N2))         return(YES);
```

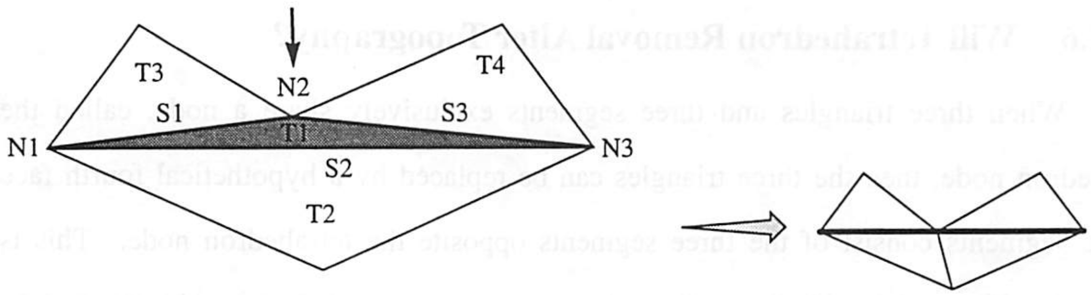


Figure 10 Removing triangle by collapsing one node. (Annotations refer to variables in the software.)

2.4.1.5 Will Flipping Triangles of Segment Alter Topography?

As shown in Fig. 6, flipping triangles about their common segment is simply the process of reconnecting the common segment to the nodes originally not in common. This procedure is often called an "edge flip" and is extensively used in Delaunay triangulation. See Chapter 3 for more information. Notice that no triangles are removed, rather a local re-triangulation occurs. Referring to Fig. 11, a "yes" response to any of the following pseudo-C-code questions means that flipping triangles will alter the surface topology and therefore should *not* be used.

```
• if (is_segment_on_a_surface_edge(longest_segm)) return(YES);
• if (is_segment_on_a_ridge(longest_segm))         return(YES);
```

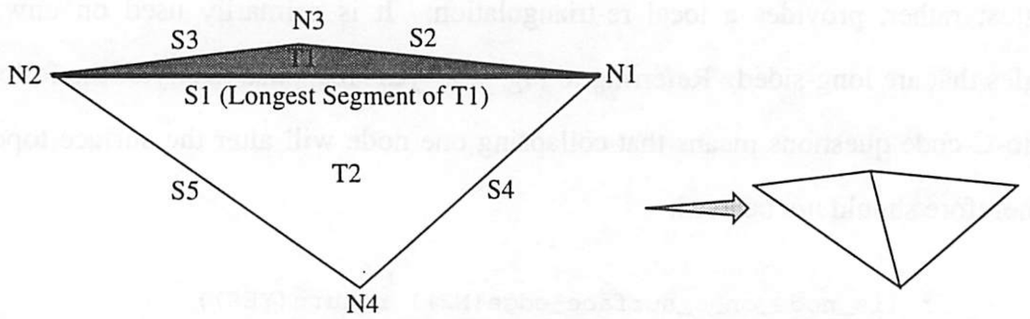


Figure 11 Local re-triangulation by flipping adjacent triangles. (Annotations refer to variables in the software.)

2.4.1.6 Will Tetrahedron Removal Alter Topography?

When three triangles and three segments exclusively share a node, called the tetrahedron node, then the three triangles can be replaced by a hypothetical fourth face whose segments consist of the three segments opposite the tetrahedron node. This is depicted in Fig. 6. Listed below is the only query necessary before using the tetrahedron removal procedure. If the query returns "yes", then the tetrahedron node forms the apex of a ridge corner (Sec. 2.3.4.4), and its removal will alter the surface topography.

```
• if (is_node_on_a_ridge_corner(tetra_node)) return(YES);
```

2.4.1.7 Will Surface Corner Triangle Removal Alter Topography?

When trying to remove a triangle that is in a surface corner (Sec. 2.3.2.14), the following question must be answered before using the surface corner removal procedure. Refer to Fig. 12 for the nodes.

```
• if (is_node_on_a_ridge(N2)  &&
         is_node_on_a_ridge(N3)) return(YES);
```

If the query response is "no", then merging one node is used to remove the corner triangle as shown in Fig. 12. However, since the topological corner must still be preserved, either N1 or N2 is merged into N3. If one of them is on a ridge, then the other node is chosen. If neither node is on a ridge, then the node connected to the shorter edge segment, S2 or S3, is merged into N3.

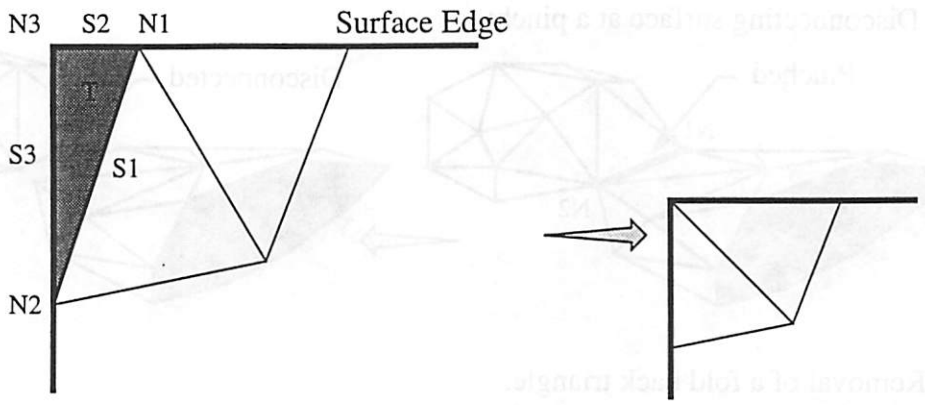


Figure 12 Removal of a triangle in a surface corner. (Annotations refer to variables in the software.)

2.4.2 Degenerate Surface Conditions and their Correction

Due to various reasons during the advancement of a simulation mesh or due to local mesh refinement algorithms, a strange surface connectivity often results. These situations must be located and rectified when they occur. In this section, a few of these irregular occurrences are presented. Figure 13 shows schematic of the degenerate surface conditions.

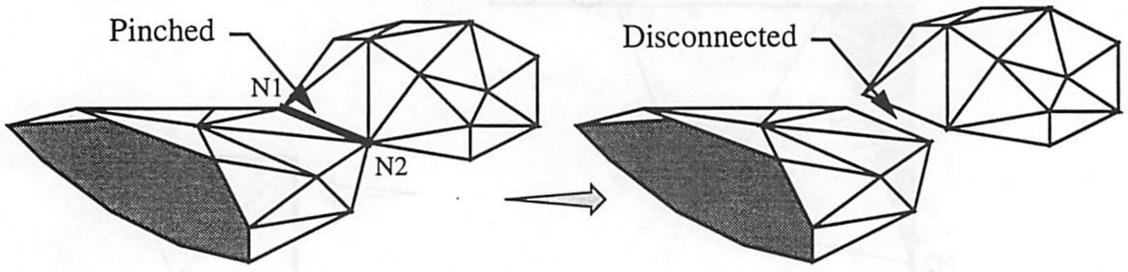
2.4.2.1 Is Surface Pinched?

As depicted in Fig. 13, when two adjacent segments are connected to one another at both of its nodes, a surface pinch has occurred. This is easily detected by checking the segments attached to each node. For each segment attached to a node (N1), if one of the other segments attached to the same node has its second node (N2) in common, then a surface pinch has been detected. This operation is $O(n \cdot s_{avg}^{2})$ where n represents the number of nodes in the surface and s_{avg} represents the average number of segments attached to a node.

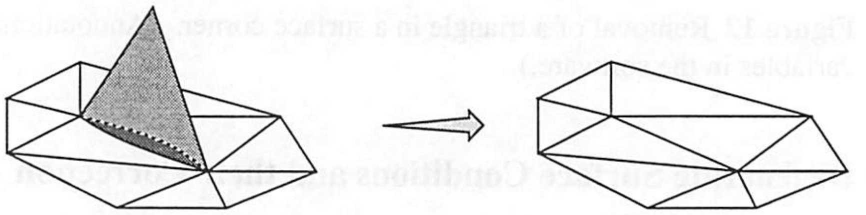
2.4.2.2 Is Triangle in a Fold-Back?

A fold back situation occurs when two of a triangles adjacent triangles happen to

(A) Disconnecting surface at a pinch:



(B) Removal of a fold back triangle:



(C) Removal of a disconnected triangle:

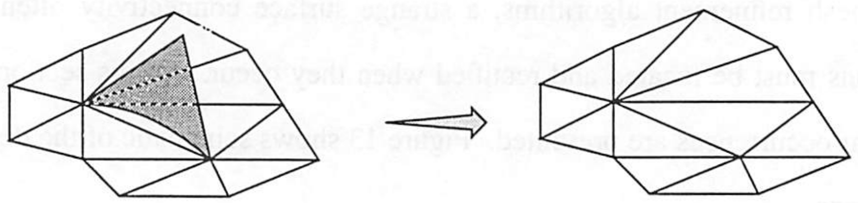


Figure 13 Degenerate surface conditions and results after their correction procedures.

be the same triangle. In other words, two triangles have been doubly connected to each other so that they coincide with each other. This phenomenon is depicted in Fig. 13.

2.4.2.3 Is Triangle Disconnected?

If for some unlikely reason, a single triangle or back-to-back triangles have lost their attachment to the surface through their segments, then the triangle(s) is(are) identified as being disconnected. Realize that their nodes may still be connected to a valid surface mesh. A disconnected triangle is identified by having all three of its neighboring triangles equal (or NULL). This phenomenon is depicted in Fig. 13.

Should a disconnected triangle be detected, its presence is likely due to a logical error in a surface utility's decision tree that improperly chooses a triangle removal method.

2.5 Improving Query Efficiency

Reviewing the sections under 2.4.1 (Triangle Removal Queries and Methods), notice that some queries are answered by considering the response from multiple groups of sub-queries. In many cases, queries are wastefully repeated.

One way to avoid the computational inefficiency of redundant procedure calls is to include boolean fields corresponding to the queries in the data structures of the surface elements. Then an update procedure can be used to reset all of the fields to a consistent value. This method has some advantages and disadvantages. The primary advantage is computational efficiency due to readily accessible data. Two disadvantages include an increase in memory consumption due to the size of the structures, and the need for a consistency policy that updates the query values whenever the surface is advanced or re-triangulated.

As the cost of memory decreases, its consumption becomes more tolerable thereby making this disadvantage minimal. However, the consistency policy is more difficult to police. To achieve faster simulation speeds, my software predecessors used this method to include many extra fields in the elemental data structures. Unfortunately, a software engineering policy of maintaining valid data in the data structures was not upheld. Consequently, a global update of the data structures to correct invalidated data fields is often required between operations written by different authors. Not wishing to contribute to the current problem, I chose not to add additional fields to the data structures. The following band-aid question then arrises, "Is it more expensive to make redundant function calls? Or, is it more expensive to make global updates to the data structures between operations?"

Chapter 3:

Delaunay Triangulation in 2 $^1/_2$ D

3.1 What is Delaunay triangulation?

For a given set of points in 2D, a Delaunay triangulation will produce triangles that are as equilateral as possible. For this reason it is a popular 2D surface triangulation scheme [2]. Formulation of a Delaunay triangulation can be thought of as the dual to a Vornoi diagram. A Vornoi cell contains the set of all points in 2D whose distance from a node is less than the distance from any other node. Mathematically this is expressed in Equ. 1. The Voronoi diagram then consists of line segments separating the Voronoi cells

$$V(n_i) = \bigcap \{p \in \Re^2 | d(p, n_i) < d(p, n_j)\} \qquad \textbf{(Equ. 1)}$$

while the Delaunay triangulation is composed of the line segments connecting the nodes of adjacent Voronoi cells.

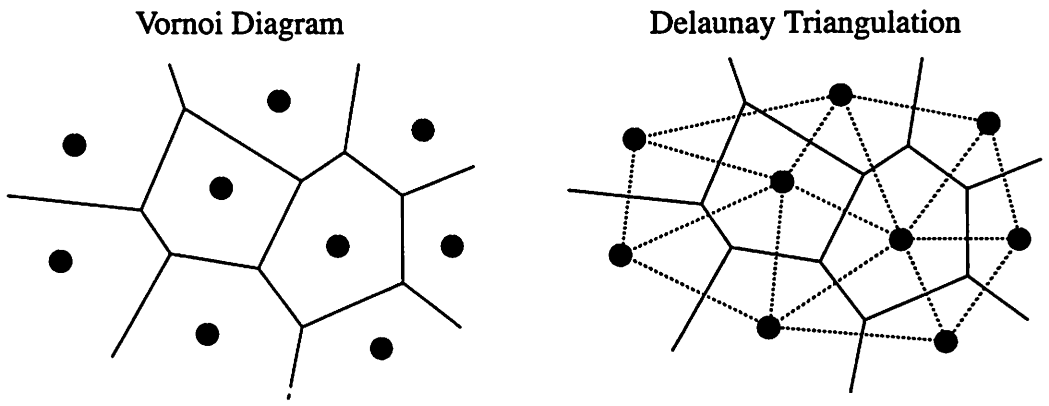


Figure 14 Schematic of the Vornoi diagram and its dual, the Delaunay triangulation for a given set of nodes in 2D.

3.1.1 Properties of Delaunay Triangulation in 2D

In 2D meshes, the Delaunay triangulation imposes the following properties:

- maximizes the minimum interior angle of all its triangles (this is equivalent to the "empty-circle condition[1]"),
- minimizes the radius of the largest circumcircle, and
- minimizes the radius of the largest enclosing circle.

In general, these properties are recognizable in a mesh when the triangles are reasonably equilateral. Figure 15 shows a depiction of these properties.

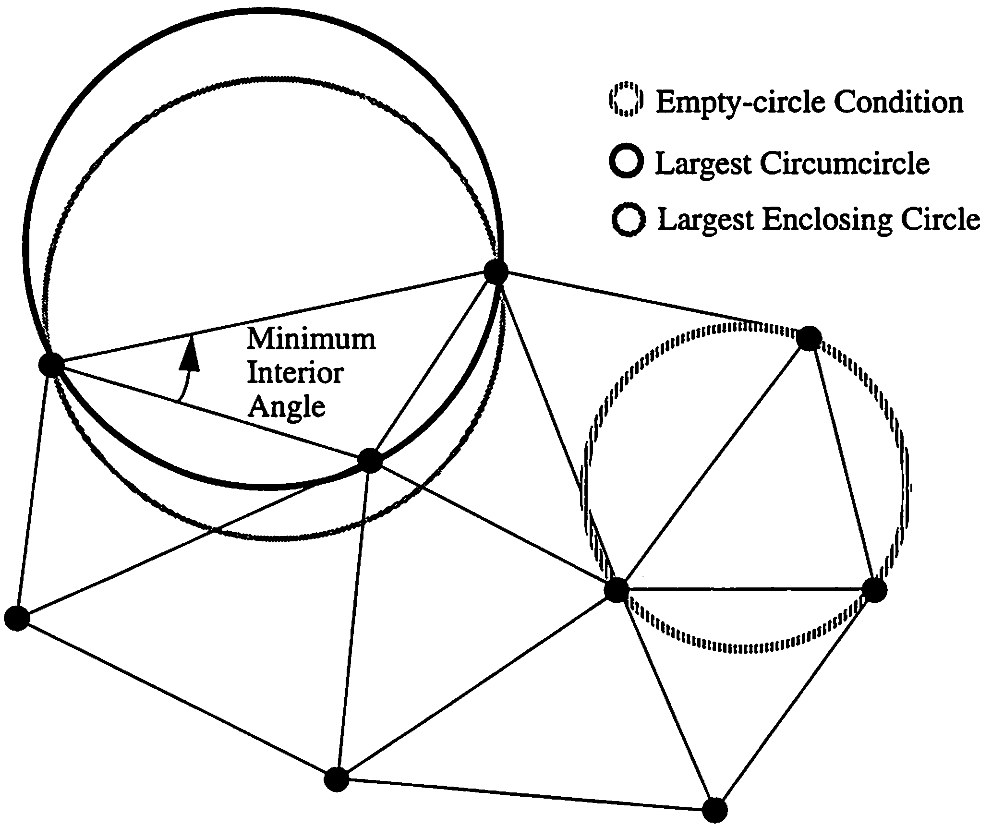


Figure 15 The three properties of a Delaunay triangulation.

1. The empty-circle condition refers to the 2D Delaunay triangulation property that a circle passing through the points of a triangle will not contain any other points in the mesh. A near exception occurs when four points of a perfect square are triangulated; the fourth point will lie *on* a circle, but *not in* a circle.

3.1.2 Edge Flipping and the Reversed Quadrilateral

The most common algorithm used to compute a Delaunay triangulation incrementally from a given triangulation is called "Edge Flipping". In this algorithm, segments common to reversed quadrilaterals are flipped until all reversed quadrilaterals have been removed. A reversed quadrilateral is identified by two neighboring triangles whose two angles opposite the diagonal sum to more than 180°. Or, a reversed quadrilateral can be identified when one of the vertices opposite the common edge lies inside the circle through the other three vertices (In-Circle test). This algorithm guarantees the three properties above and runs in $O(n^2)$. Figure 16 shows a demonstration of the edge flipping algorithm.

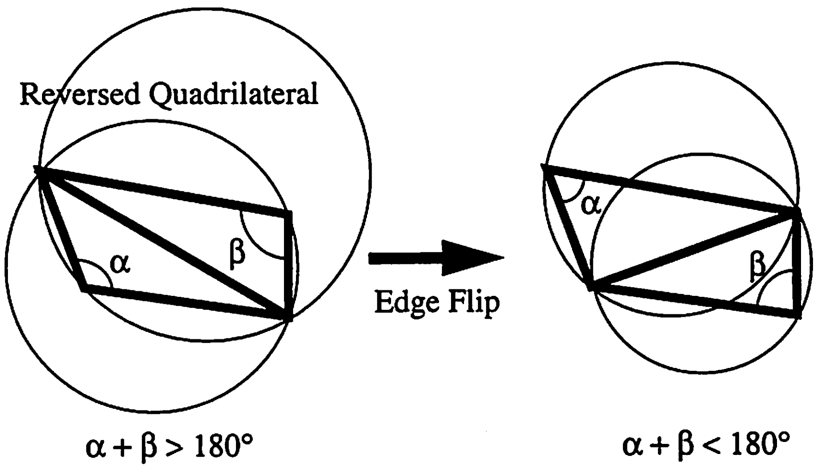


Figure 16 Recursive edge flipping of reversed quadrilaterals is a simple algorithm for computing a Delaunay triangulation.

3.2 Extending 2D Delaunay Triangulation to 2 $^1/_2$ D

Under the assumption that a three-dimensional mesh can be approximated *locally* by a two-dimensional surface, the 2D Delaunay triangulation properties can be extended to 2 $^1/_2$ D. This assumption is justifiable provided adjacent triangles in a reversed quadrilateral approach coplanarity. As such a threshold must be set to determine if adjacent triangles are approximately coplanar. Here is where the concept of a "ridge segment" enters. Recall from Sec. 2.3.3.6 that the notion of a ridge is used to denote when a segment defines a significant deviation in the surface topology, and that the "ridge angle" can be redefined as the granularity of the surface triangulation changes.

3.2.1 Algorithm for 2 $^1/_2$ D Delaunay Triangulation

Outlined below in pseudo-code is the popular edge flipping algorithm with a small enhancement. Before flipping an edge of a reversed quadrilateral, the segment is first check to see if it is on a ridge. If it is a ridge segment, then no flipping is performed. This distinguishes the algorithm as a 2 $^1/_2$ D triangulator.

```
for each interior surface Edge of a given triangulation
{
   add Edge to a Queue
}

while Queue is not empty
{
   remove the first Edge from Queue

   if (Quadrilateral $Q_e$ is reversed and Edge is not on a ridge)
   then
   {
      flip Edge of $Q_e$

      add to Queue each side of $Q_e$ not already in the Queue and
      not on the surface edge
   }
}
```

In Fig. 17, results of the 2 $^1/_2$ Delaunay triangulation algorithm are presented. The ridge angle was set to a high value of 60° so that all interior surface edges in this example would be viable candidates for flipping.

Note that Delaunay triangulation does not remove triangles, segments, or nodes. It simply redefines the connectivity so as to guarantee the properties described above (Sec. 3.1.1).

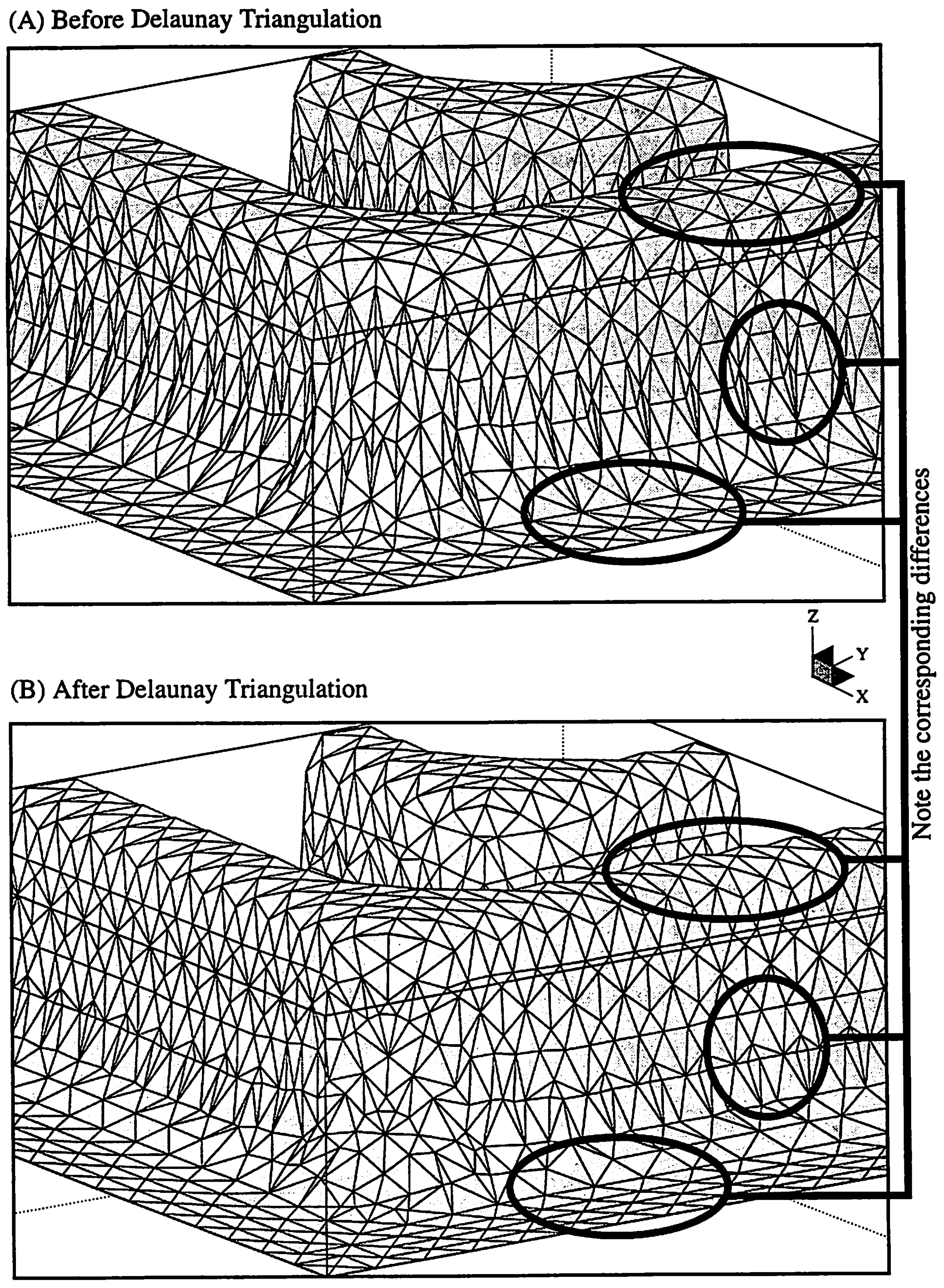

(A) Before Delaunay Triangulation
(B) After Delaunay Triangulation
Z
Y
X
Note the corresponding differences

Figure 17 A 3D surface mesh before (A) and after (B) Delaunay triangulation with a high ridge angle threshold set to 60° so all interior surface edges would be considered viable candidates for flipping.

3.2.2 Constrained Algorithm for 2 $^1/_2$ D Delaunay Triangulation

For reference in subsequent algorithms, the Delaunay triangulation algorithm described above (Sec. 3.2.1) can be slightly modified and applied to the interior re-triangulation of a general convex or concave polygon. Outlined below in pseudo-code is the modified algorithm. Figure 18 shows what edges are regarded as interior edges of an enclosing polygon.

```
for each interior Edge of an enclosing polygon
{
   add Edge to a Queue
}

while Queue is not empty
{
   remove the first Edge from Queue

   if (Quadrilateral $Q_e$ is reversed and Edge is not on a ridge)
   then
   {
      flip Edge of $Q_e$

      add to Queue each side of $Q_e$ not already in the Queue and
      not on the enclosed polygon
   }
}
```

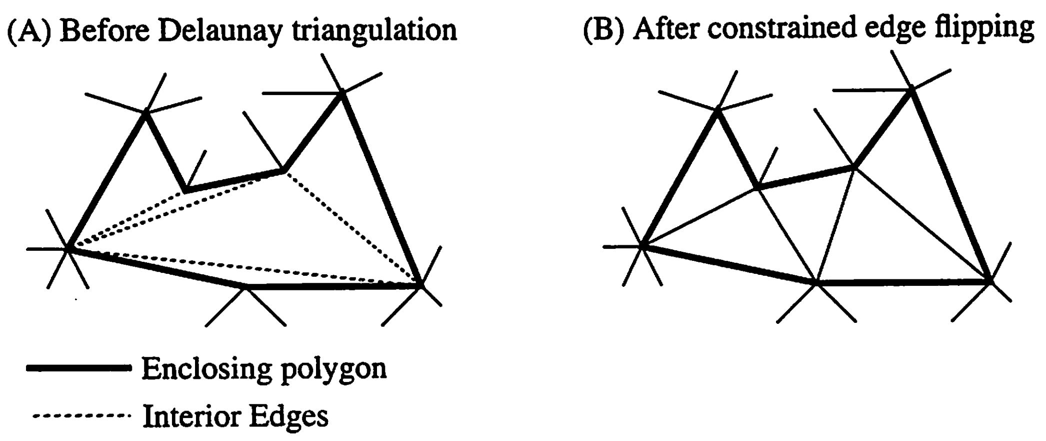

(A) Before Delaunay triangulation
(B) After constrained edge flipping
Enclosing polygon
Interior Edges

Figure 18 Delaunay triangulation constrained to flipping only edges within an enclosing polygon.

3.3 Suggestions for Future Development

In the current versions of the SAMPLE-3D programs (NETCH version 2.1 and DEVELOP version 1.0), an ad-hoc polygon subdivision procedure was developed for resolving the triangulation of a polygon created by the intersection of triangles in a "delooping" process. Refer to John Helmsen's Ph.D. thesis Sec. 5.4.4 (Triangle Subdivision) [8]. In his procedure, undesirable triangulations containing long narrow triangles can result. His subdivision procedure can benefit from the constrained Delaunay triangulation as described in Sec. 3.2.2.

Chapter 4:

Decimation of Triangles

4.1 What is Decimation of Triangles?

Decimation of triangles is a process whereby the number of triangles used to represent a topographic feature is reduced without significantly altering what is considered to be the essential 3D shape.

Topography simulators like SAMPLE-3D [21,17,18] often begin with a flat triangulated surface containing a fine tessellation so as to capture the intricate curvature that results during surface advancement. Once the device's essential shape emerges in the evolving surface after a few time steps, regions of low geometric curvature remain finely tessellated. As simulation continues for minutes or hours, a considerable amount of computer time and memory is wastefully consumed by advancing the finely tessellated low curvature regions. In this situation, the decimation of triangles can be used to reduce the degree of tessellation in areas of low curvature thereby reducing computer memory usage and increasing simulation speeds without affecting the surface regions containing interesting topography. For these reasons, decimation of triangles can be very useful in TCAD simulators as larger surface areas are simulated.

In this chapter, two schemes for reducing the granularity of an advanced surface mesh in regions of low geometric curvature are presented. The first scheme is a greedy algorithm that removes as many triangles as possible; the second scheme, derived from the first, is an incremental algorithm that prevents too many triangles from being removed at once. These variations on the same algorithm allow for flexibility depending on the intended usage.

It should also be noted that the notion of ideal segment lengths presented in Sec. 2.3.2 (Triangle Queries) is disregarded in the decimation algorithm.

4.2 Use of a Normalized Coplanarity Vector

As previously stated, the decimation algorithm identifies regions on the surface where the geometric curvature is low. In these regions, triangles are removed. To identify these regions the notion of a normalized coplanarity vector is introduced. Although similar, It is *not* the same as a Gaussian radius of curvature vector consisting of a vector originating at the center of a sphere that is tangent to the surface at a given point. The normalized coplanarity vector used in this algorithm originates at a node on the surface and points in the direction of the center of its Gaussian curvature sphere.

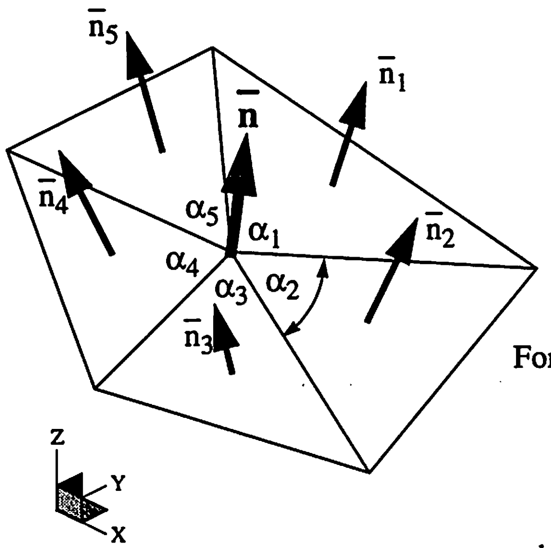
Because of the way the normalized coplanarity vector is computed at a node (Sec. 4.2.1 and Sec. 4.2.2), the magnitude of the vector approaches 1 when the surface is perfectly planar and it approaches 0 when a cusp is present. The magnitude of the vector is then compared to a user-definable threshold above which the surface is considered planar, and decimation is applied about that node. This is how the normalized coplanarity vector will be used.

What about a saddle point? Understand that the normalized vector is used to measure the *local* planarity of surface at a node without concern for units or signs and is therefore dependent on the tessellation granularity. If the surface is finely tessellated at a given saddle point, the triangles surrounding the saddle point approach coplanarity and the normalized surface vector would have a magnitude near 1. Conversely if the surface is coarsely tessellated, the triangles would loose their degree of coplanarity and the magnitude of the normalized coplanarity vector would approach 0. Therefore a true saddle point can be targeted as an area of low geometric curvature if the tessellation is very fine.

4.2.1 Computing Normalized Coplanarity Vectors

Given only the geometric position of the surface triangles, a normalized estimation of the surface coplanarity can be made. As depicted in Fig. 19 and Equ. 2, the normalized coplanarity vector is computed as a vector summation of the normal vectors of each triangle surrounding the node. However, each normal vector is weighted by the fraction of the angle the triangle makes with the total angle of all the triangles common to the node. In this manner, acute triangles bordering the node have less impact on the vector calculation. Notice that the magnitude of the normal will be 1 when the surrounding triangles are coplanar, and it is less than 1 when the triangles are not coplanar. Also notice

$$\bar{n} \cong \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3+\alpha_4+\alpha_5}\bar{n}_1 + \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3+\alpha_4+\alpha_5}\bar{n}_2 + \dots$$



Formally:

$$\bar{n} \cong \sum_{i=1}^{k} \frac{\alpha_i}{\sum_{i=1}^{k} \alpha_j} \bar{n}_i \qquad \textbf{(Equ. 2)}$$

where k = number of surrounding triangles
As $|\bar{n}| \Rightarrow 1$, surface is perfectly flat at node
As $|\bar{n}| \Rightarrow 0$, surface contains a cusp at node

Figure 19 Computing a normalized coplanarity vector.

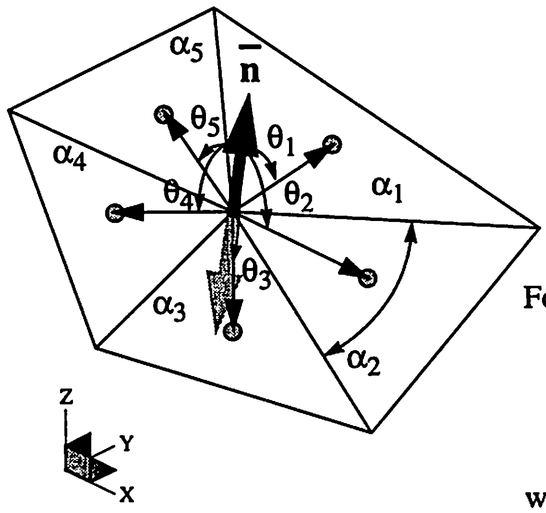
that the vector addition method requires consistency in the orientation of the triangles' normal direction. Refer to Sec. 7.2.5 (Orienting Triangles of an Enclosed Surface).

4.2.2 Computing the Direction of the Coplanarity Vector

Having computed the normalized coplanarity vectors, it may also be important to know the direction of curvature. Although not required to decimate triangles, it could be used in surface modeling utilities for crenulation reduction and is therefore included.

As depicted in Fig. 20 and Equ. 3, the direction of the normalized coplanarity vector is computed by taking a weighted average of the angles between the coplanarity vector and the corresponding vectors to the center of the surrounding triangles. If the average angle is greater than 90°, the coplanarity vector is reversed to point in the concave direction.

$$\theta_{avg} \cong \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3+\alpha_4+\alpha_5}\theta_1 + \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3+\alpha_4+\alpha_5}\theta_2 + \dots$$



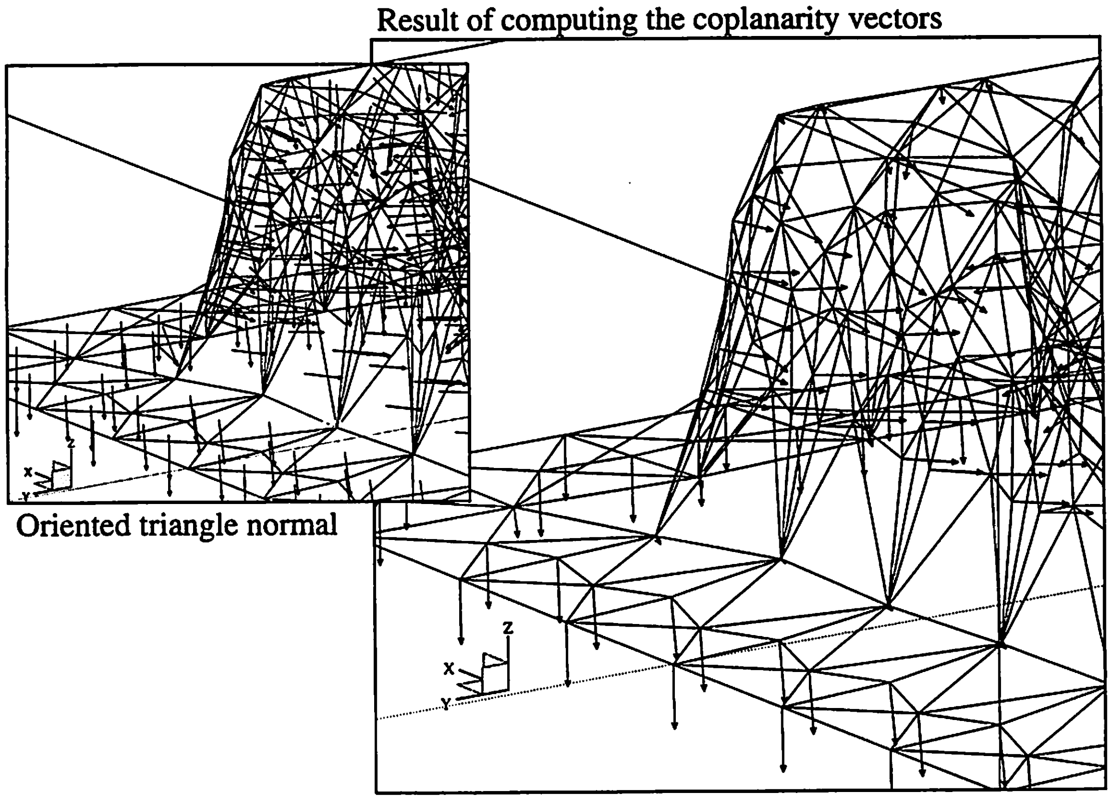
Formally:

$$\theta_{avg} \cong \sum_{i=1}^{k} \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j} \theta_i \qquad \textbf{(Equ. 3)}$$

where k = number of surrounding triangles
if $\theta_{avg} \leq 90°$ already in the correct direction
if $\theta_{avg} > 90°$ reverse coplanarity direction

Figure 20 Computing the direction of the coplanarity vector. Note that it parallels the direction of surface curvature.

For inspection purposes, Fig. 21 shows an overlay of the normalized coplanarity vectors on a real surface. Notice at the bottom portion of the final surface that the direction of the coplanarity vectors are sensitive to small crenulations in the mesh.



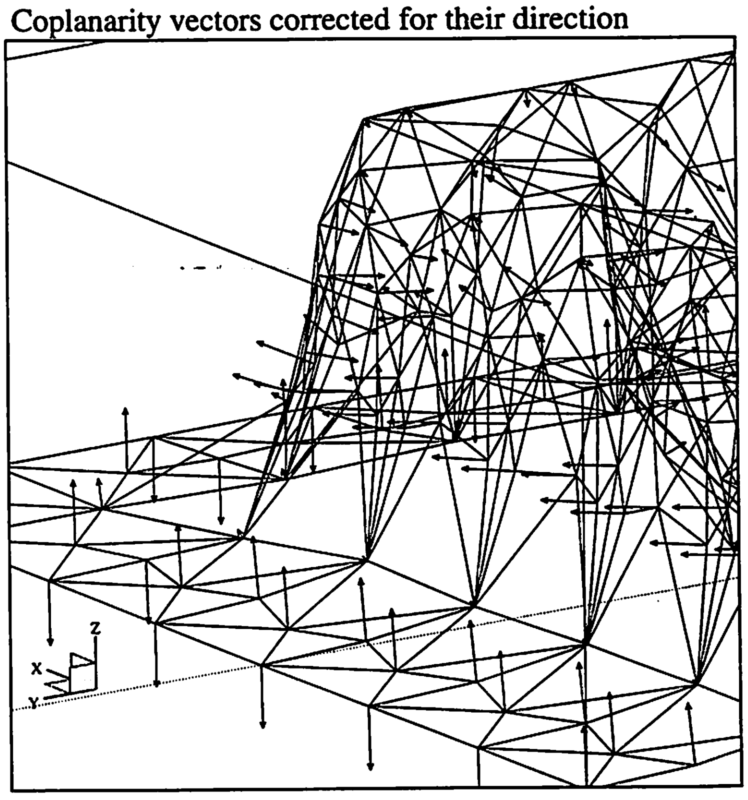


Figure 21 Results of the normalized coplanarity vector computations.

4.3 Algorithm for Decimating Triangles

Outlined below in pseudo-code is a description of the algorithm for decimating triangles across a surface.

```
for each Node of a surface not on a surface corner
{
   compute the normalized Coplanarity at Node

   if (Coplanarity > Threshold_Coplanarity) then
   {
      if (Node is on a surface edge) then
      {
         Add a segment whose nodes are the surface edge nodes
         adjacent to the Node

         Add a triangle whose segments are the new segment and
         the surface edge segments adjacent to the Node
      }

      find the surrounding polygon

      remove the Node and re-triangulate surrounding polygon
   }
}
```

Notice that two portions of the algorithm are emphasized with an underline; what to do when the targeted node is on a surface edge, and the node removal followed by re-triangulation. These instructions are detailed in the following subsections.

4.3.1 Preconditioning Edge Nodes for Re-triangulation

As pointed out in the decimation algorithm, all nodes are potential candidates for removal except those on a surface corner. The remaining nodes are either internal to the surface mesh or on the edge of the surface. Realize that nodes on the edge of the surface mesh are not enclosed by a surrounding polygon. When this type of node has been targeted for removal, a preconditioning step is required. As depicted in Fig. 22 the preconditioning step first adds a segment whose nodes are the surface edge nodes followed by the addition of a triangle whose segments include the newly added segment and the surface edge segments adjacent to the targeted node. Then the surrounding polygon is found. This step reduces the problem down to the re-triangulation of an approximately planar convex or concave polygon.

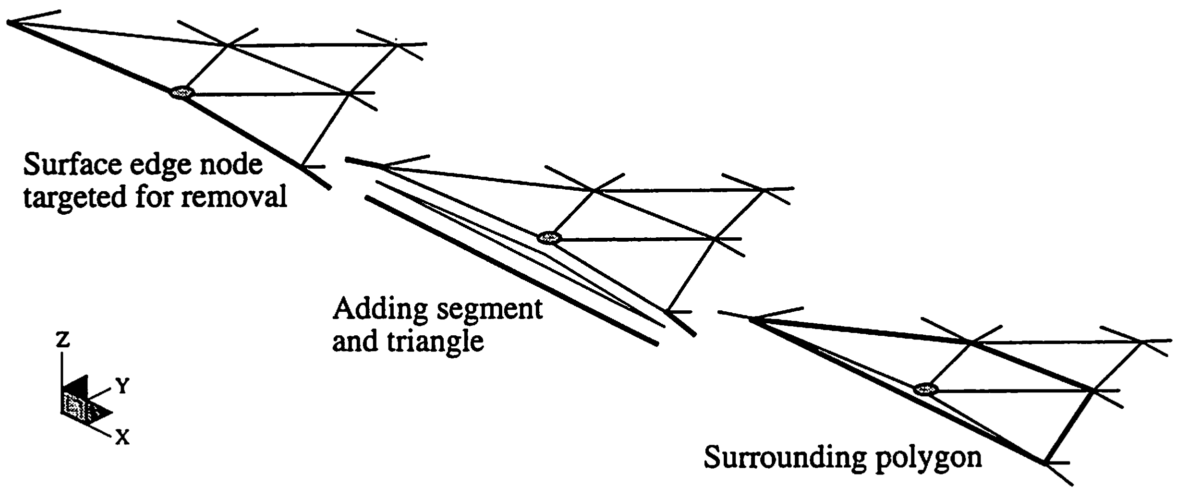


Figure 22 The preprocessing step of a surface edge node targeted for removal in the decimation process.

4.3.2 Procedure for Removing Node and Re-triangulation

Having found the surrounding polygon of a targeted node, the node can be removed and the surrounding polygon re-triangulated. This procedure makes use of two operations for modifying the mesh: 1 node merging (Sec. 2.4.1.1) and a constrained Delaunay triangulation (Sec. 3.2.2).

The procedure begins by finding the shortest segment surrounding the targeted node and merges the targeted node along the shortest segment into the surrounding polygon. Then the surrounding polygon is re-triangulated as described by the constrained Delaunay triangulation algorithm. This procedure is outlined in the following pseudo code and depicted in Fig. 23.

```
find the shortest segment surrounding the targeted Node
```

```
merge the Node into the node opposite the shortest segment

create a segment queue with the remaining segments within the
surrounding polygon

perform a constrained Delaunay triangulation on the segment
queue
```

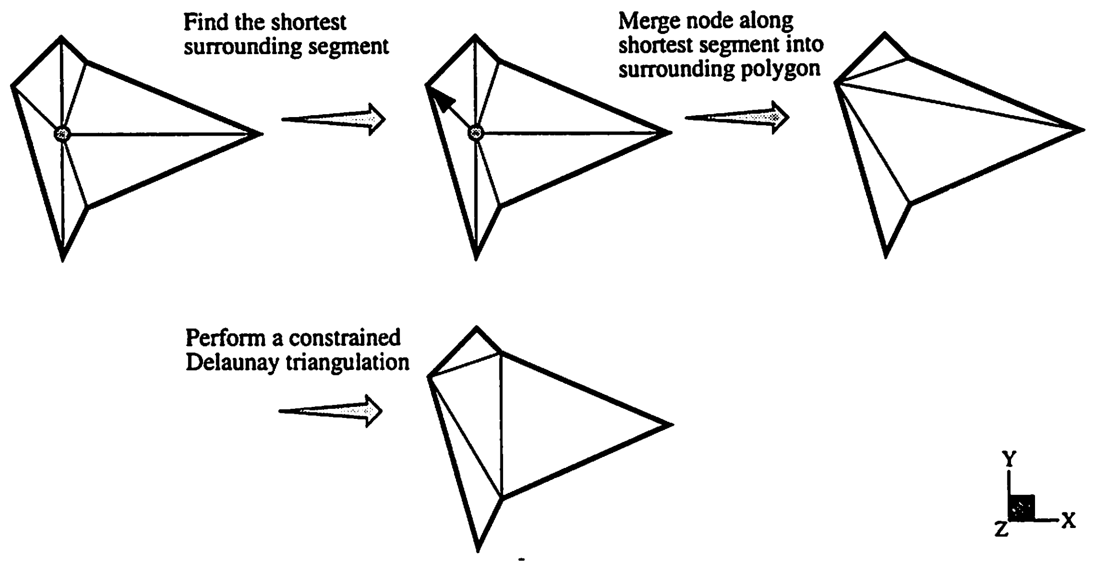

Find the shortest surrounding segment
Merge node along shortest segment into surrounding polygon
Perform a constrained Delaunay triangulation
Y
Z
X

Figure 23 Demonstration of the node removal and re-triangulation procedure within the surrounding polygon.

4.3.3 A Greedy and Incremental Decimation Scheme

The greedy decimation scheme follows the decimation algorithm exactly as outlined. It attempts to removal *all* nodes whose normalized coplanarity vector exceeds the threshold value. The incremental algorithm attempts to achieve a more uniform decimation by making several passes over the surface. In each pass, it retains selected nodes. After the re-triangulation step, the nodes of the remaining triangle whose center most closely overlaps the removal node are marked. This prevents their removal as the decimation algorithm marches across the rest of the surface. Figure 23 shows an illustration of this procedure beginning with the re-triangulation result of Fig. 23.

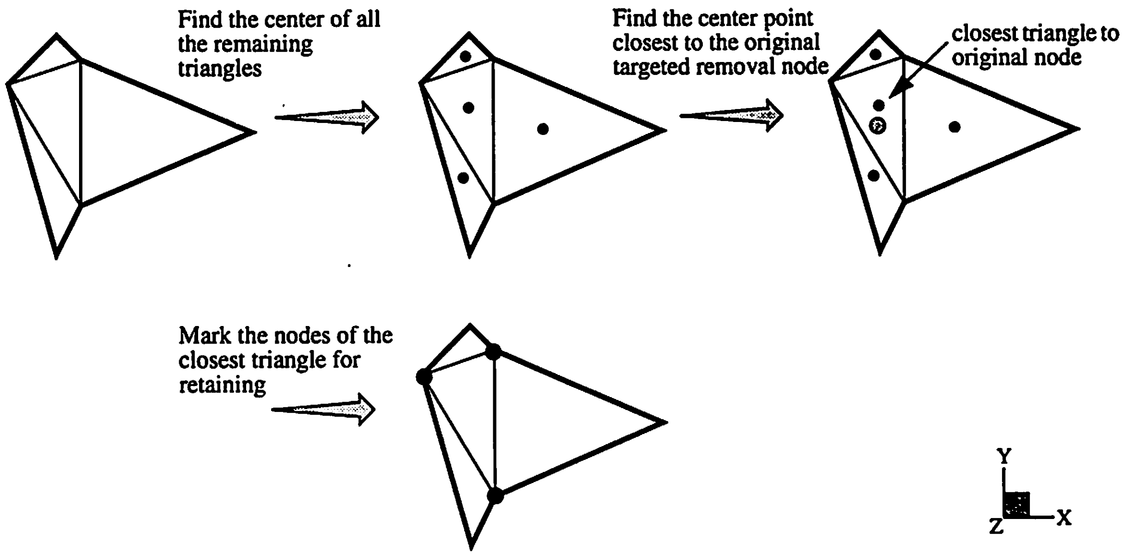

Find the center of all the remaining triangles
Find the center point closest to the original targeted removal node
closest triangle to original node
Mark the nodes of the closest triangle for retaining
Y
Z
X

Figure 24 How three of the surrounding nodes are selected for "retaining" during an incremental decimation scheme.

4.4 Progressive Examples of Triangle Decimation

To demonstrate results from the decimation algorithms, the following sections contain figures of the surface triangulations before and after greedy and incremental schemes are applied.

4.4.1 Decimation of a Simple Flat Mesh

One of the simplest examples to test the algorithms is to perform the decimation on a flat surface. Figure 25 contains results from this test case for both the greedy and incremental algorithm. Realize that a complete decimation of a triangulated surface of rectangular shape will result in two triangles.

4.4.2 Decimation of an Elbow Etched Surface

A more complicated example to demonstrate the progress of the greedy and incremental decimation schemes is shown in Fig. 26 and Fig. 25 respectively. Here, a finely tessellated elbow shaped surface resulting from an etching process in SAMPLE 3D

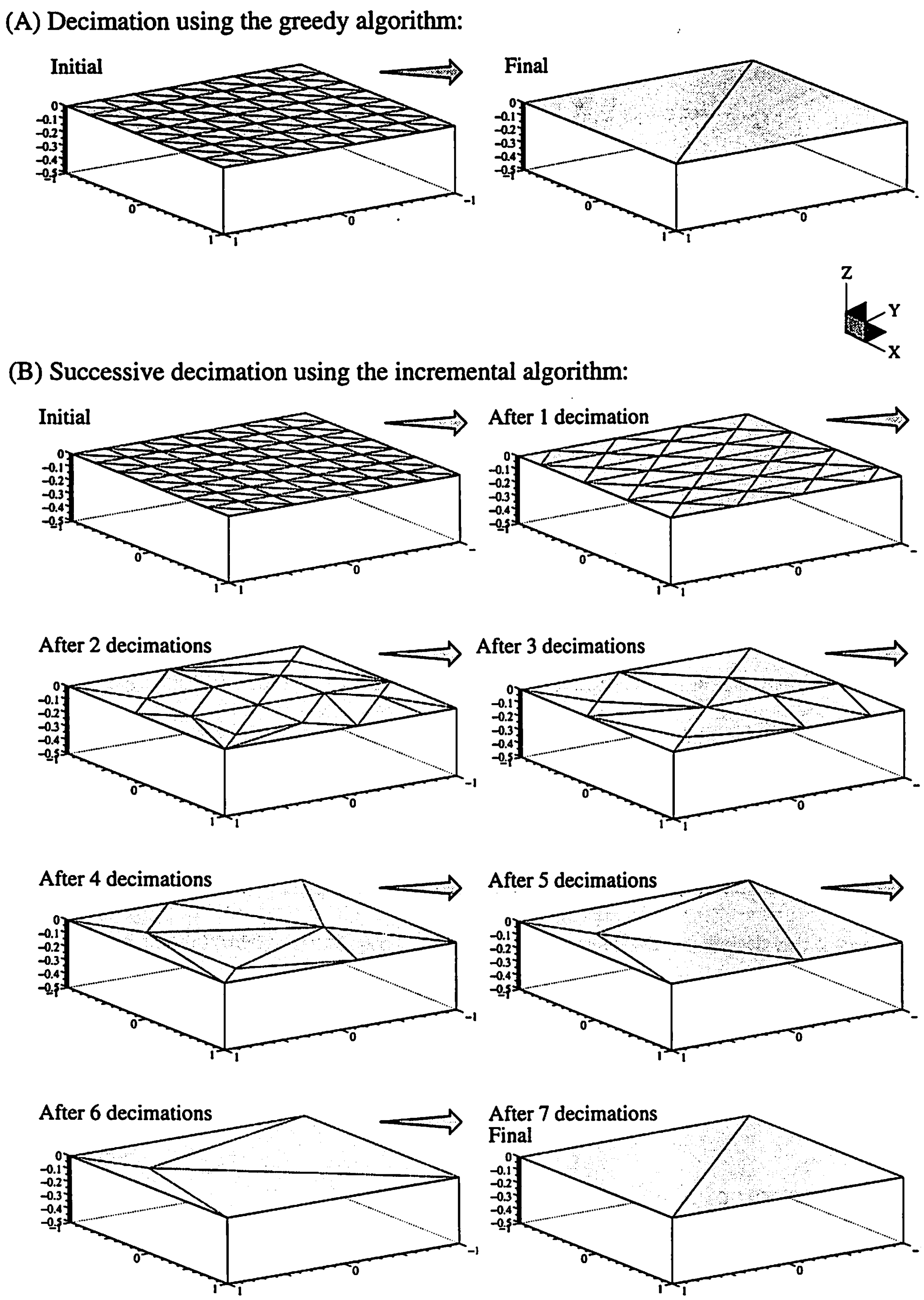


Figure 25 Decimation of a flat surface using the greedy scheme (A), and the incremental scheme repeatedly (B).

is decimated.

Using the greedy decimation scheme, the triangle count is reduced 45%.

After seven decimations using the incremental scheme, a steady state is reached (Fig. 25 H), and the triangle count is also reduced by 45%. At this step, the surface will likely contain thin triangles that require removal using the thin triangle removal algorithm described in Chapter 5. This reduces the triangle count even further and is shown in Fig. 28.

No matter which decimation algorithm is used, notice that the characteristic topology of the surface is unchanged despite the significant reduction in triangle count.

4.4.3 Decimation of an Etched and Sputter Deposited Surface

Similar to the previous example, Fig. 29 demonstrates results from the greedy decimation scheme applied to an etched and sputter deposited surface. In total, the triangle count is reduced by 75% through the processes of decimation and thin triangle removal. Notice in the cutaway plot of Fig. 29 D that the non-planarity of the deposited surface within the hole remains intact. Again, the characteristic topology of the surface is unchanged despite the significant reduction in triangle count.

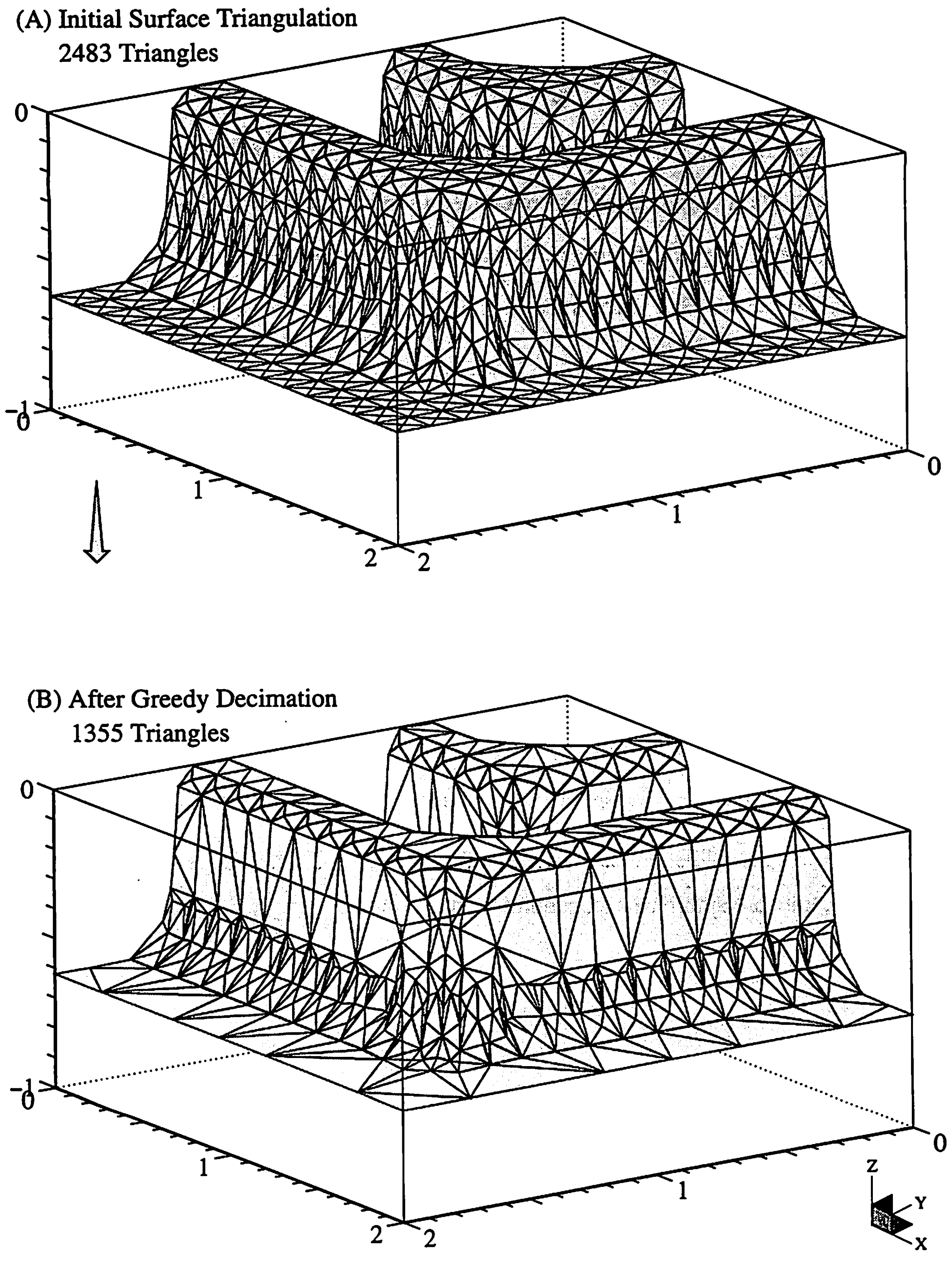
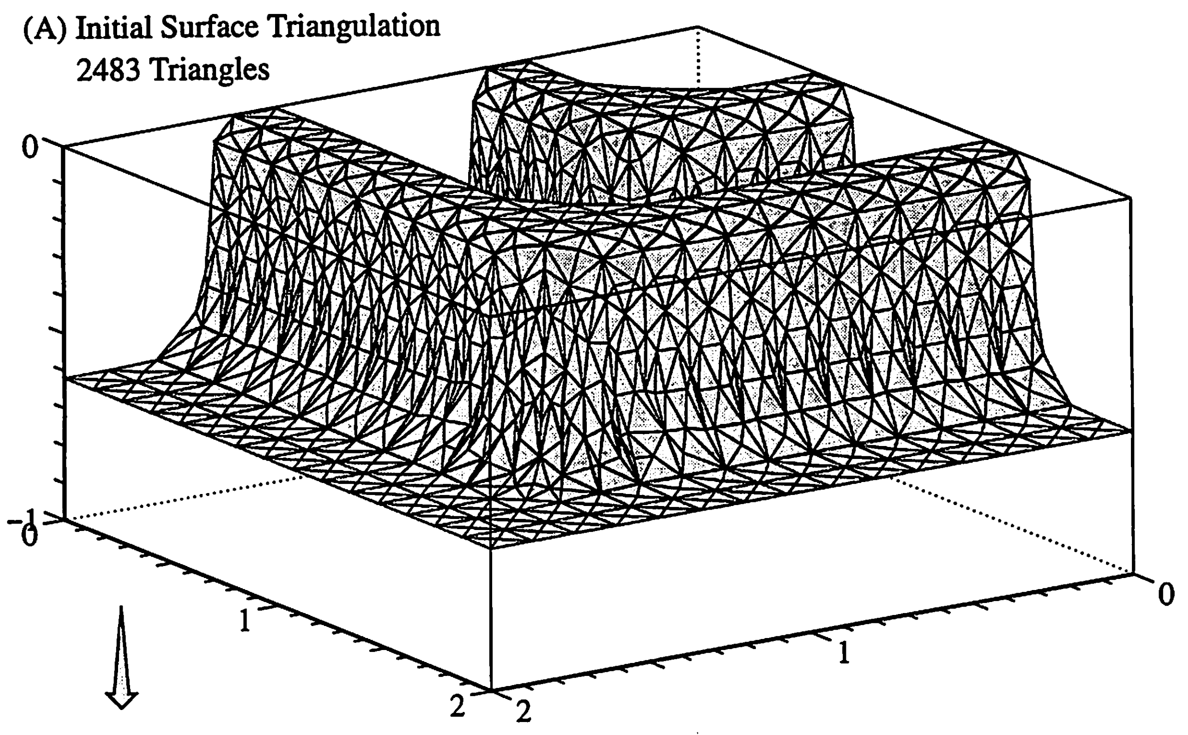

(A) Initial Surface Triangulation
2483 Triangles
0
-1
0
1
2
2
1
0
(B) After Greedy Decimation
1355 Triangles
0
-1
0
1
2
2
1
0
Z
Y
X

Figure 26 Decimation of an etched elbow shaped surface using the greedy removal scheme. Units are in microns.



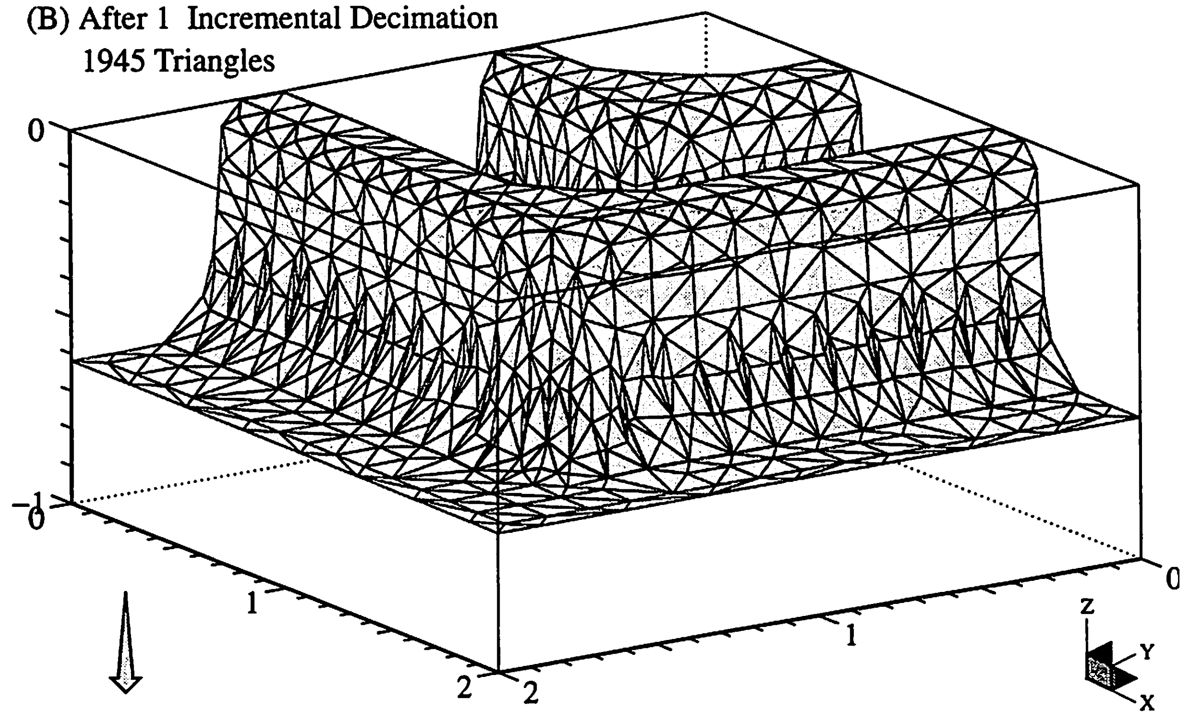
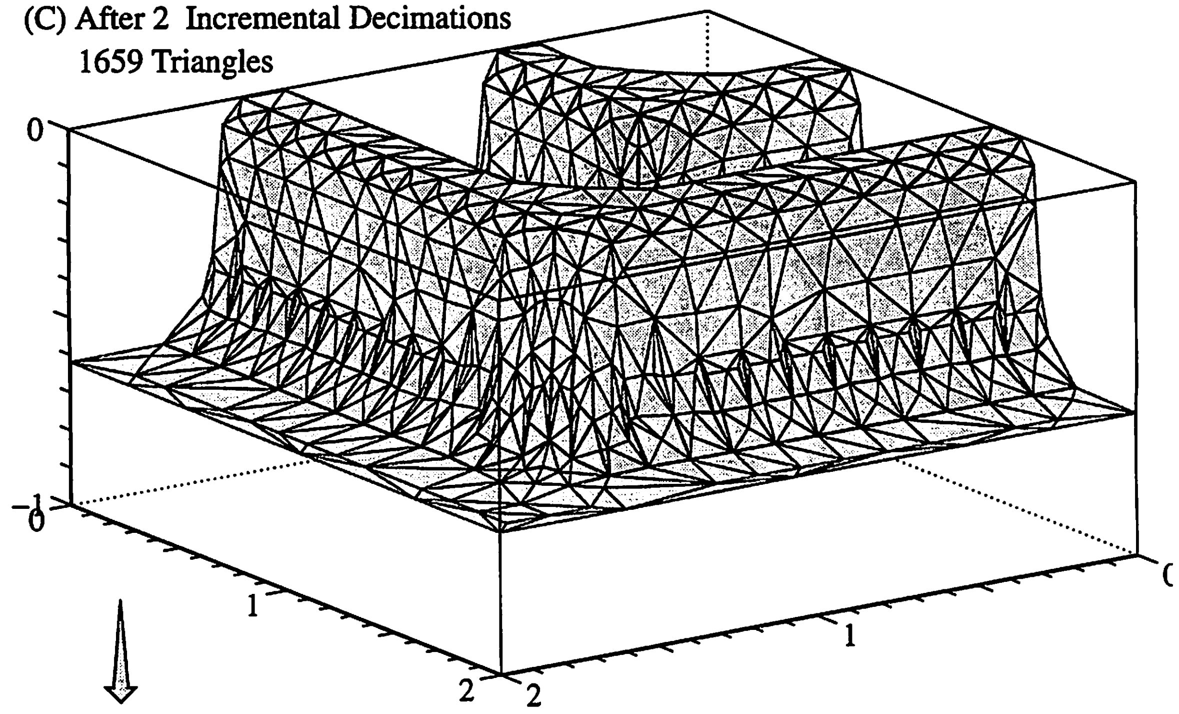


Figure 27 Decimation of an etched elbow shaped surface using the incremental removal scheme repeatedly.

(C) After 2 Incremental Decimations
1659 Triangles



(D) After 3 Incremental Decimations
1487 Triangles

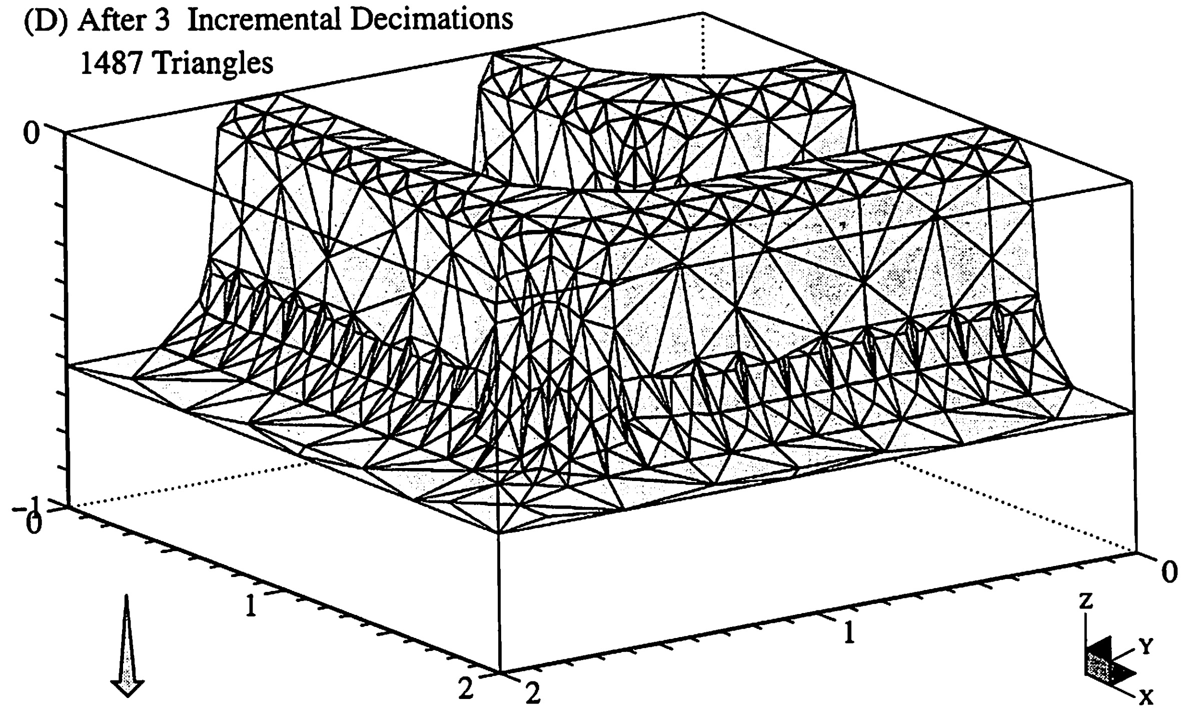
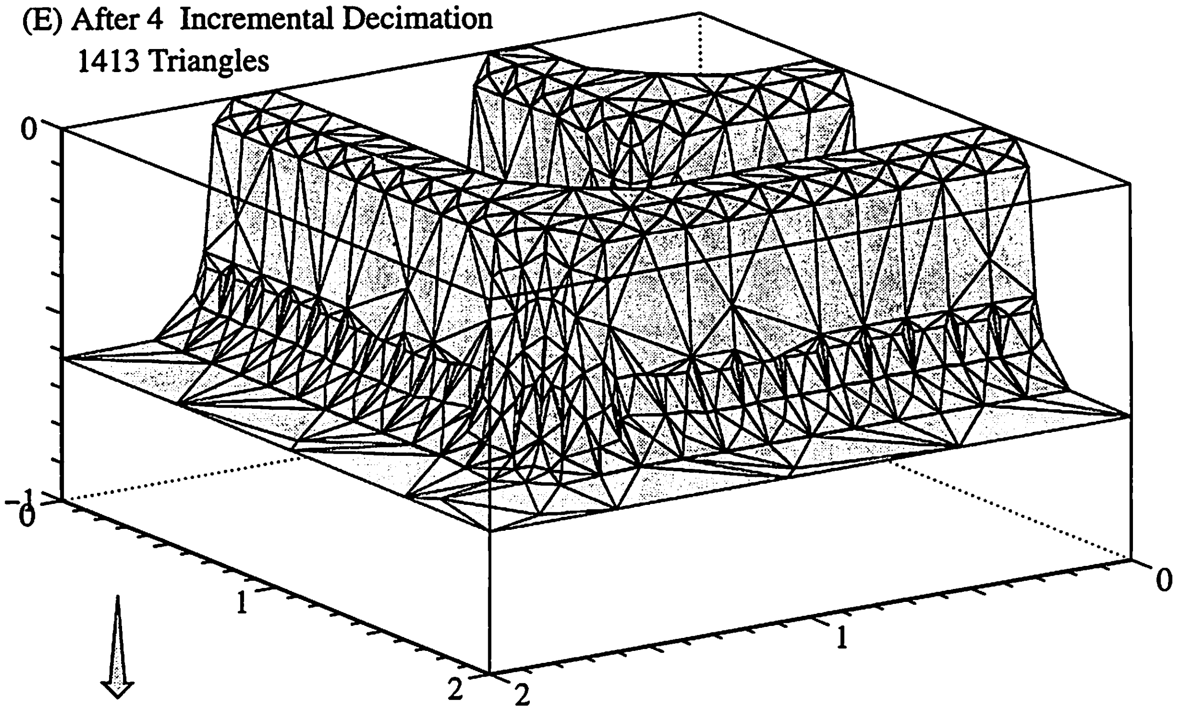


Figure 25 (continued)

(E) After 4 Incremental Decimation
1413 Triangles



(F) After 5 Incremental Decimation
1379 Triangles

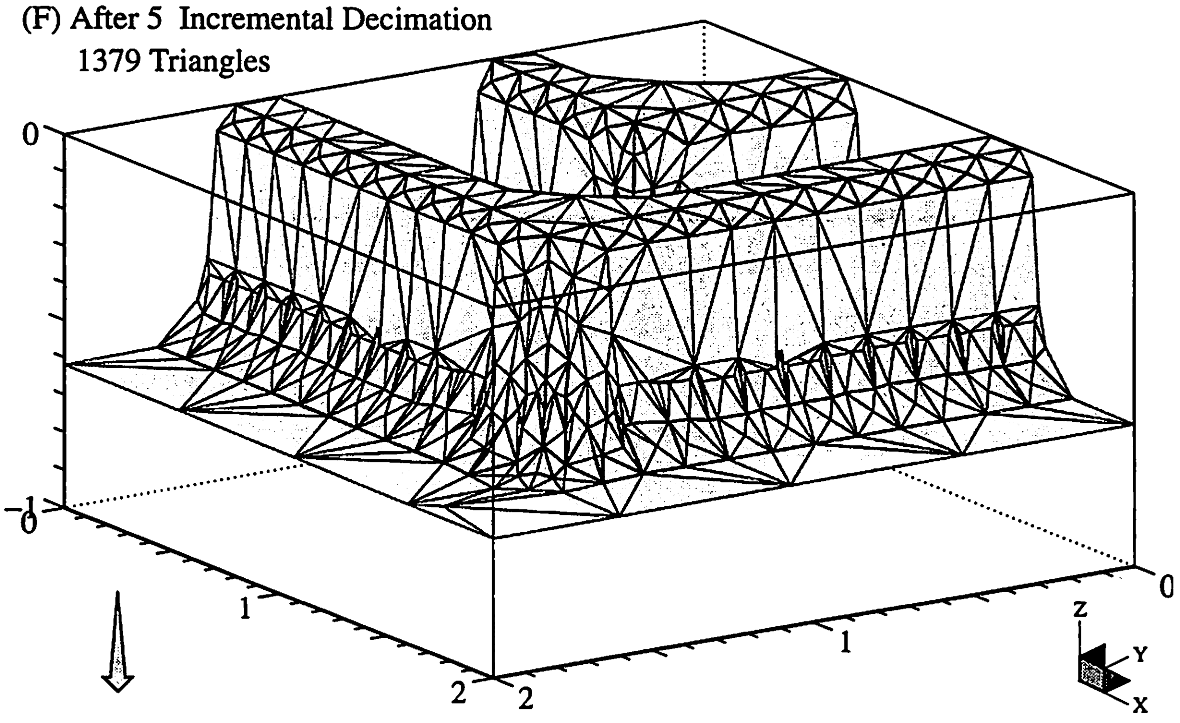
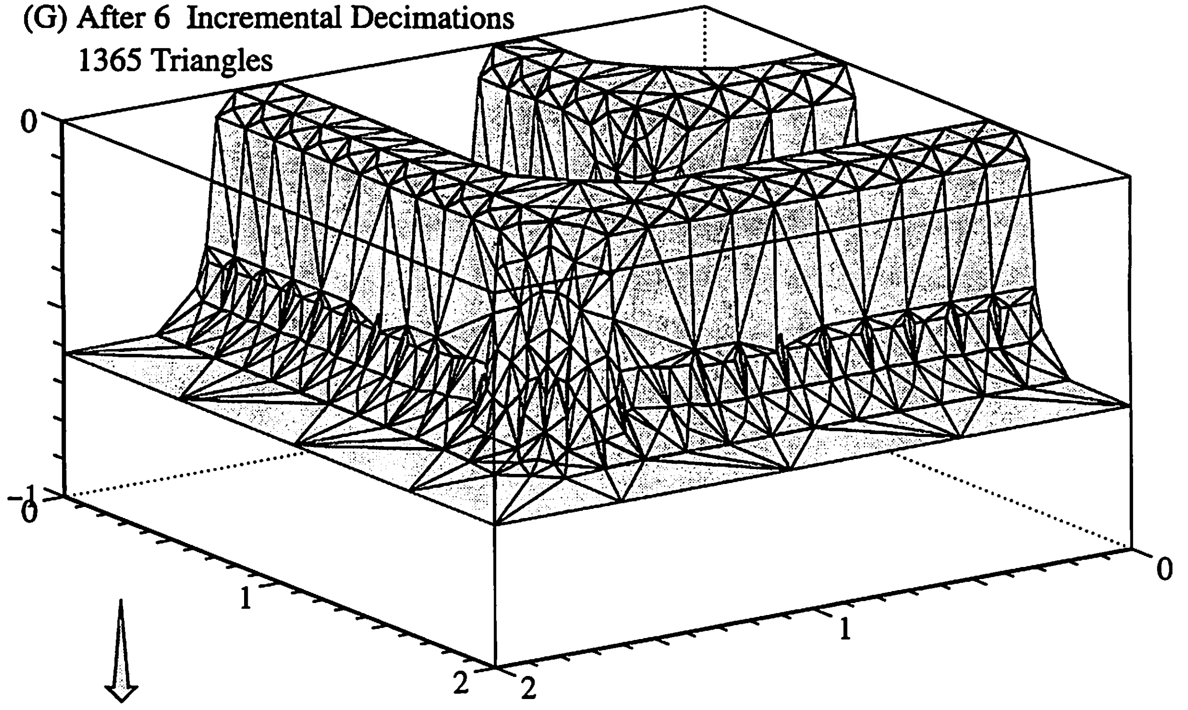


Figure 25 (continued)

(G) After 6 Incremental Decimations
1365 Triangles



(H) After 7 Incremental Decimations
1363 Triangles

Steady State

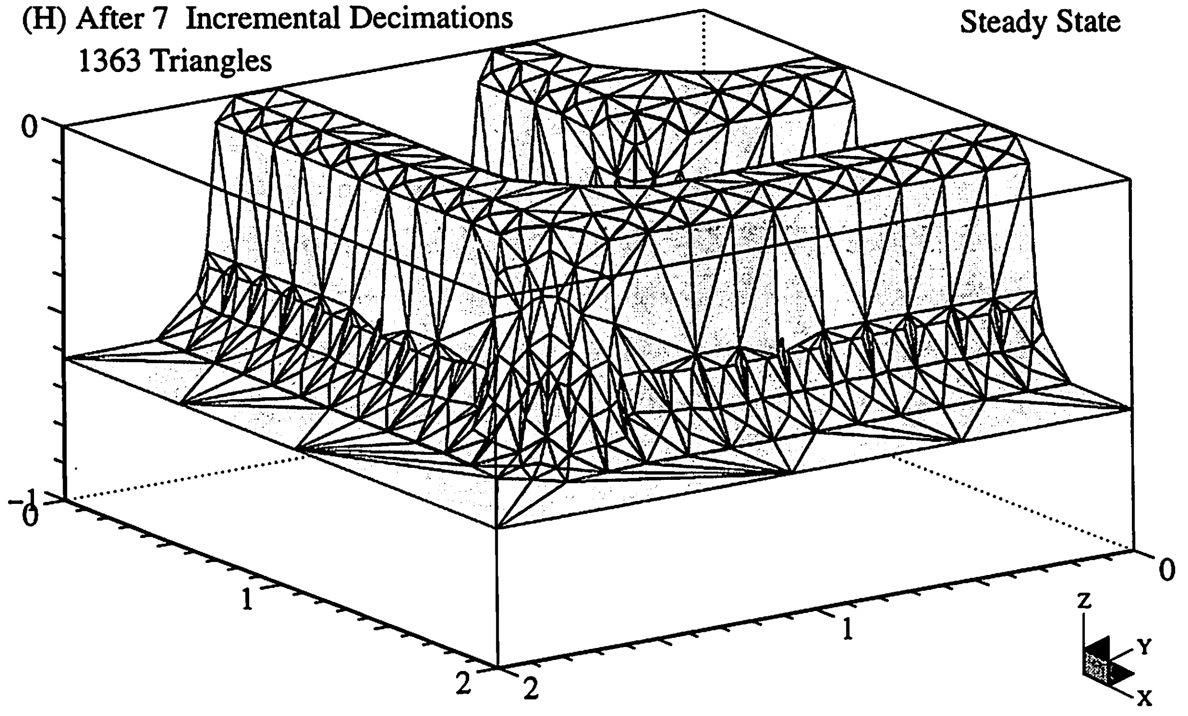


Figure 25 (continued)

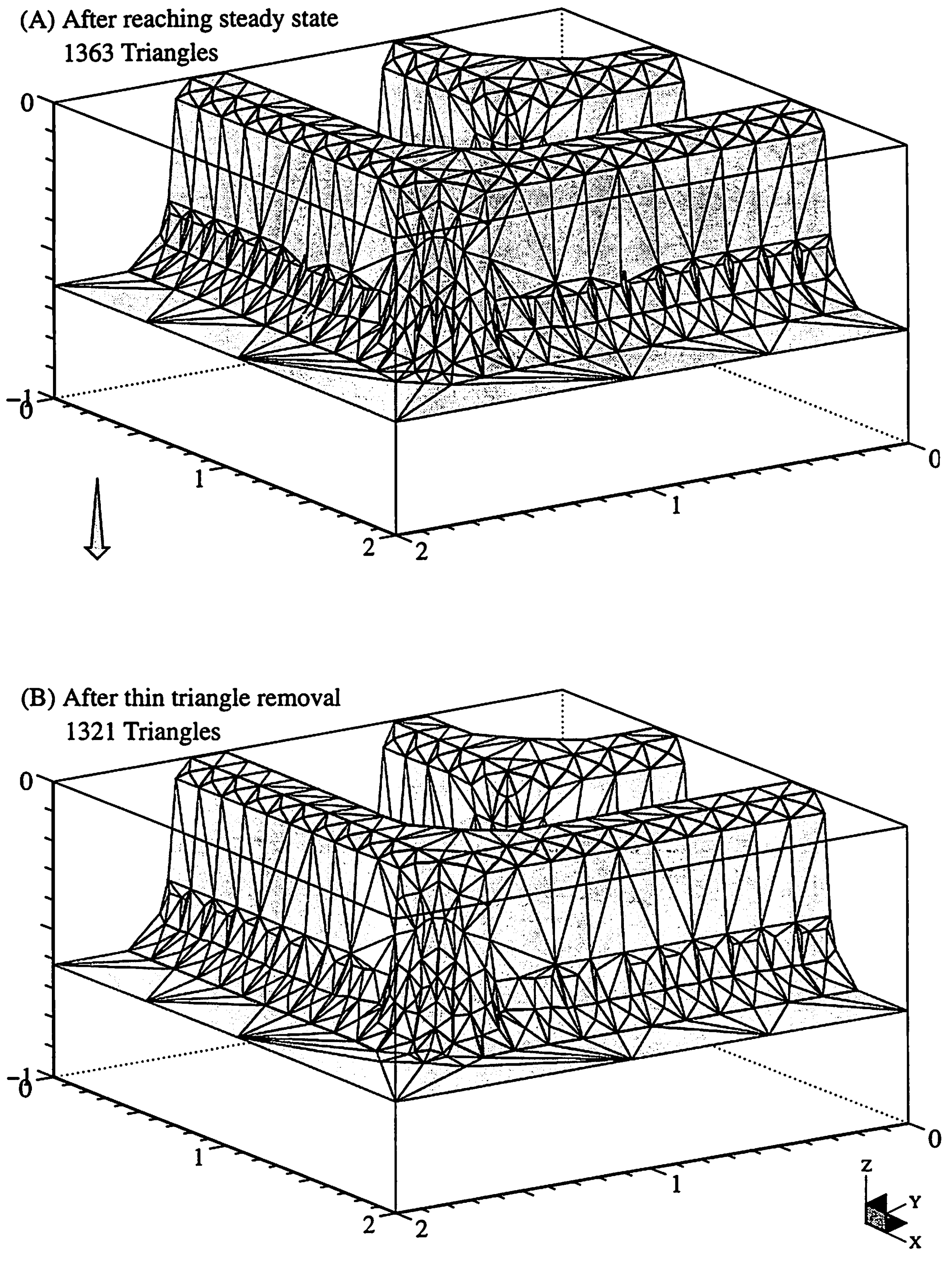
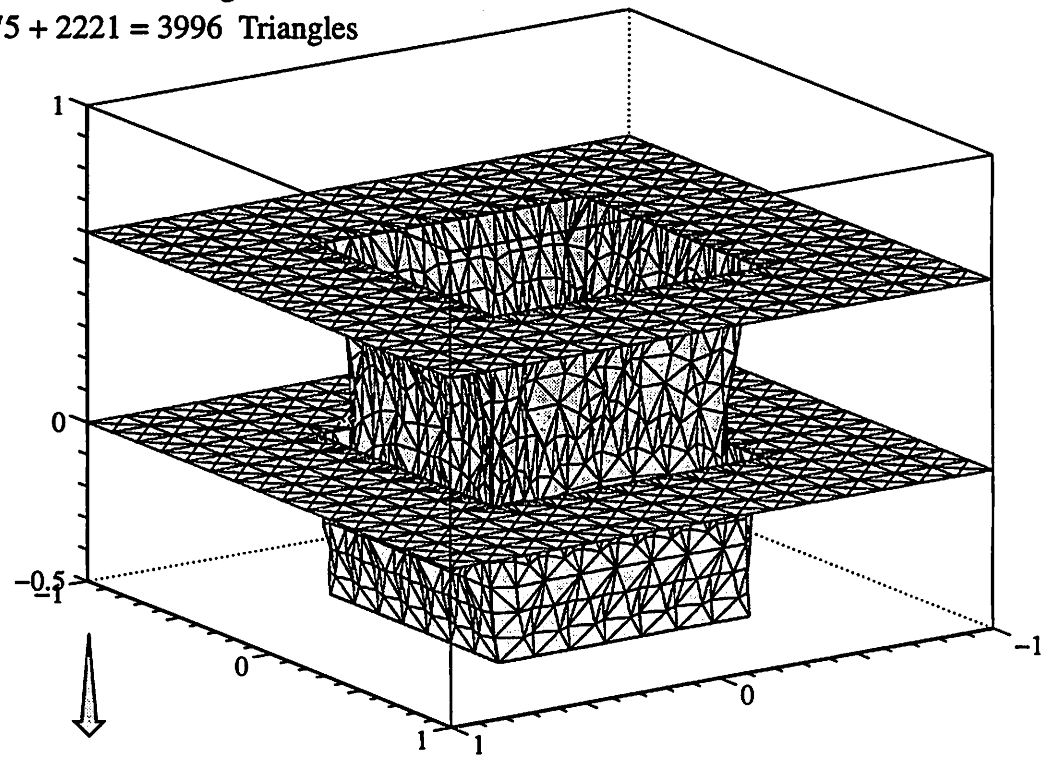

(A) After reaching steady state
1363 Triangles
0
-1
0
1
2
2
1
0
(B) After thin triangle removal
1321 Triangles
0
-1
0
1
2
2
1
0
Z
Y
X

Figure 28 Upon reaching steady state using the incremental decimation algorithm (A), the mesh likely requires removal of thin triangles (B).

(A) Initial Surface Triangulation
1775 + 2221 = 3996 Triangles



(B) After Greedy Decimation
408 + 844 = 1252 Triangles

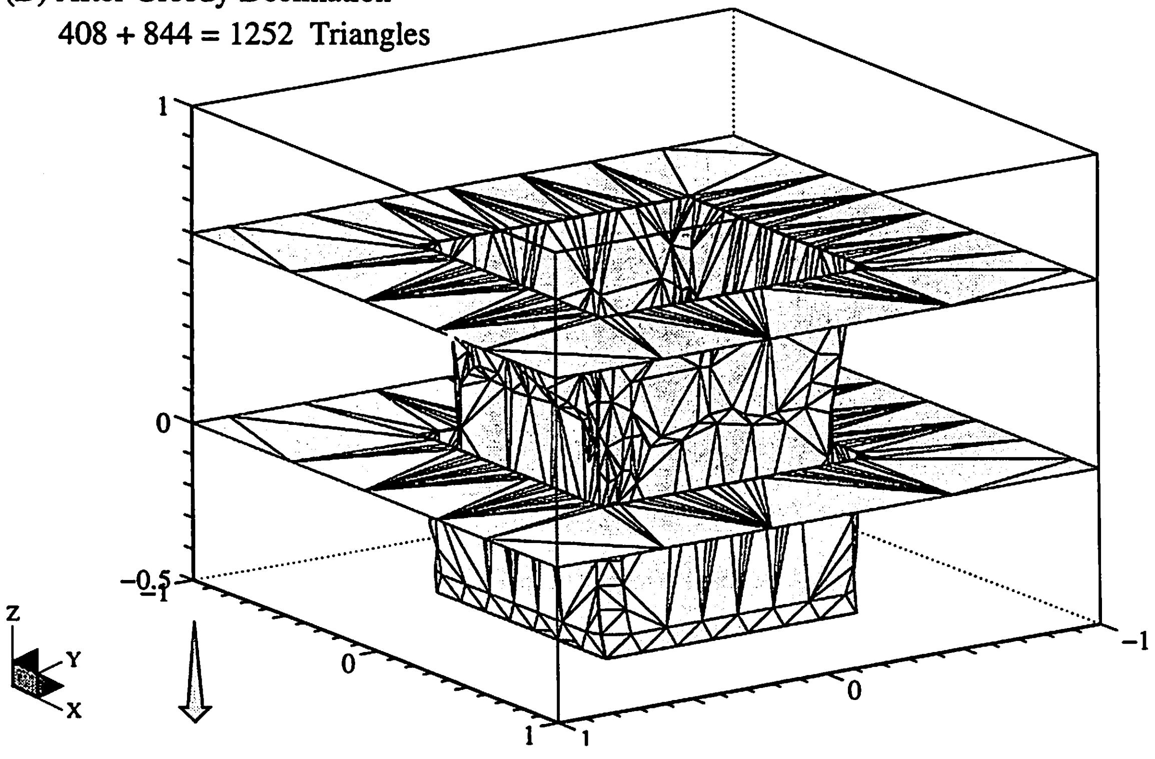
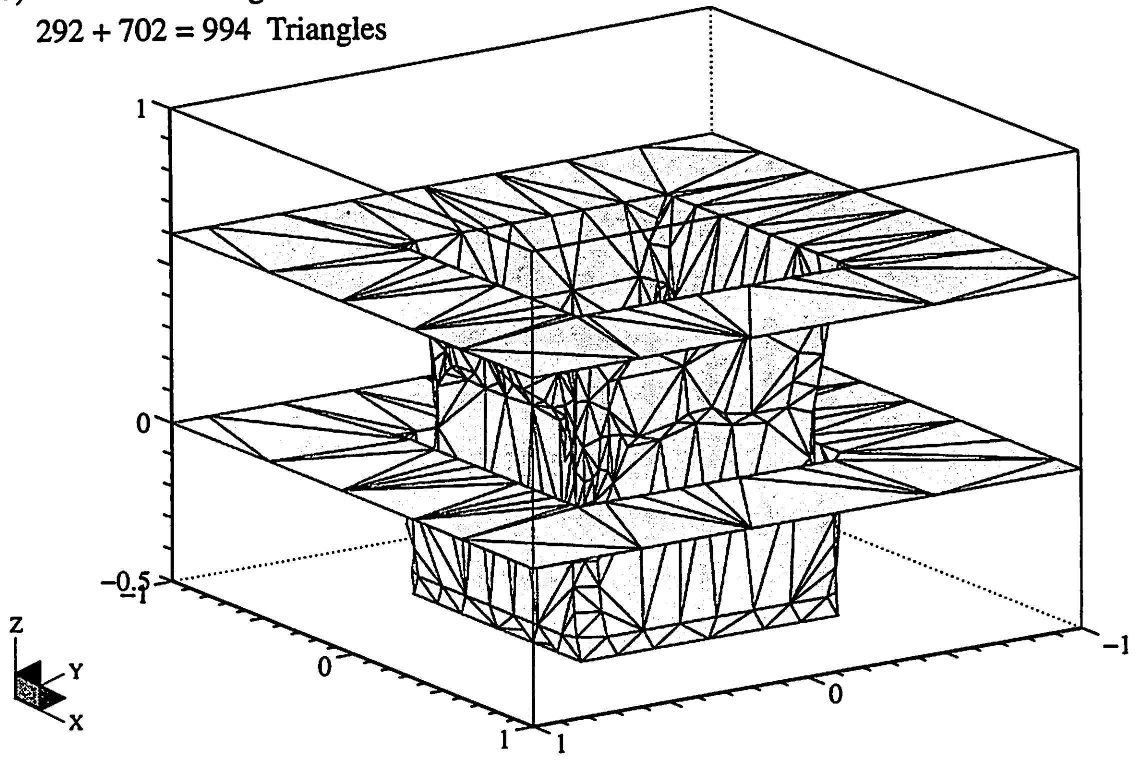


Figure 29 Decimation of an etched and sputter deposited surface using the fast greedy algorithm. Units are in microns.

(C) After Thin Triangle Removal
292 + 702 = 994 Triangles



(D) Cutaway Close-up

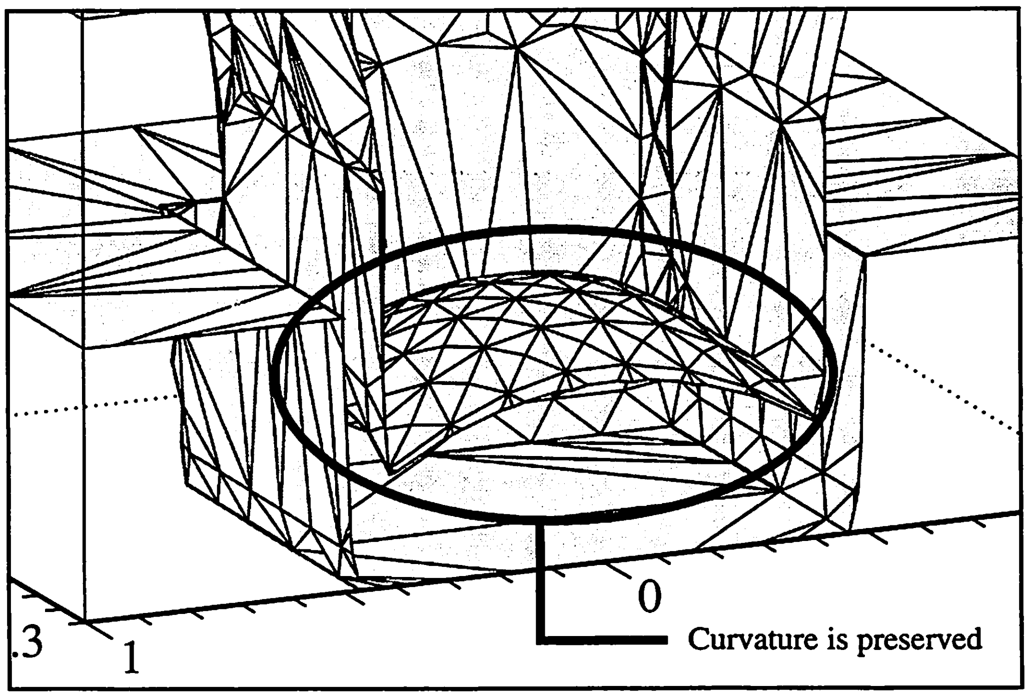


Figure 29 (continued)

Chapter 5:

Thin Triangle Removal

5.1 Thin Triangles

For engineering and mathematical applications that make use of triangulated surface meshes, thin triangles as defined in Sec. 2.3.2.3 (Is Triangle Thin?) are problematic. They often cause robustness and simulation difficulties especially when a triangle is acute to the point that its area approaches zero. Aside from robustness and simulation difficulties, thin triangles do little to aid in visualization of the surface topography; rather they wastefully consume the memory space of the computer. Thus it is advantageous to detect and remove thin triangles as needed.

5.2 Where do Thin Triangles come from?

Undoubtedly, there are many unique circumstances from which thin triangles can result. In integrated circuit topography simulators such as SAMPLE-3D where surface meshes are incrementally advanced in time based on pushing nodes (ray-tracing), the shape of triangles can change significantly between time steps. When nodes come together as in the case of deposition within a concave corner, triangles connected to the nodes can significantly shrink uniformly or affinely. This is illustrated in the contracting corner of Fig. 30. In this situation, removal of thin triangles in the corner can help avoid eminent looping problems [7].

Thin triangles also result as a consequence of intersecting surfaces. Photolithography simulators that move surfaces based on ray-trace methods are subject to loop formation. In order to remove the unwanted loops, the surface is intersected with itself followed by loop detection and removal. Performing this capability was the subject

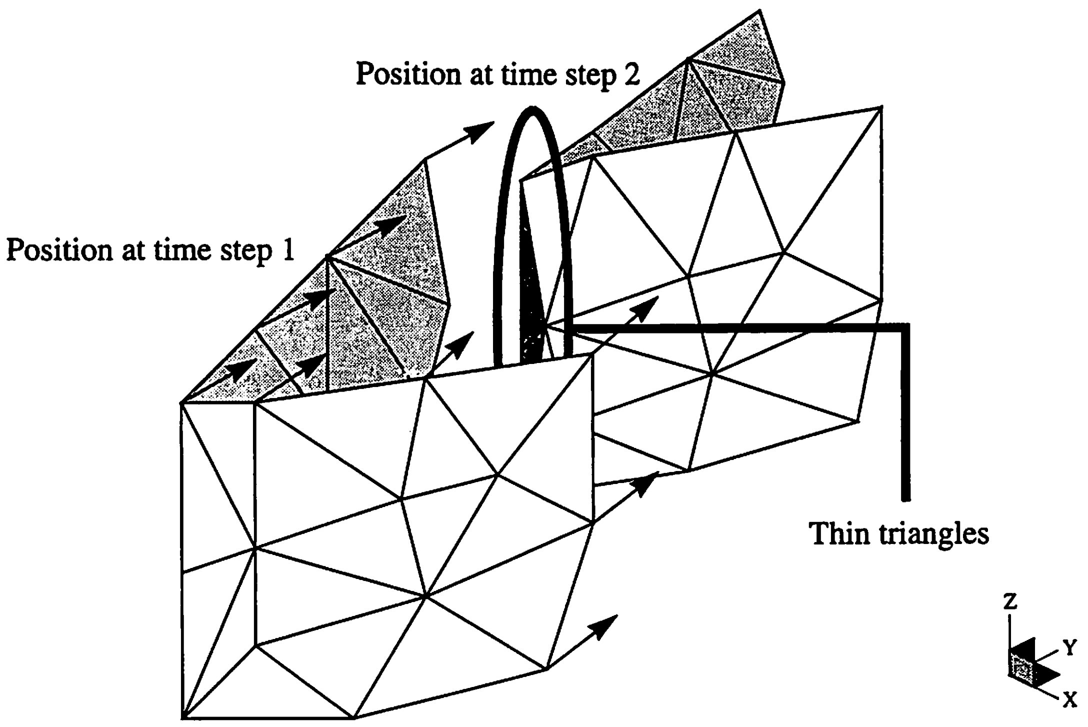


Figure 30 Formation of thin triangles along a contracting corner during a simulated deposition process.

of Helmsen's master's thesis [7]. To demonstrate how thin triangles are consequently formed during the surface intersection process, a brief illustration from Helmsen's thesis is helpful. Figure 31 contains a high level review of Helmsen's intersection procedure. The first part of the illustration simply shows two triangulated portions of a surface overlapping each other in three space. The second part shows a highlighted intersection line found by linking all the intersection segments found during the triangle-to-triangle intersection step. At this point in the procedure, each of the original triangles containing part of the intersection line is subdivided. The result of the triangle subdivision is shown in Fig. 31 C. Notice all the potential thin triangles that result.

5.3 Removing Thin Triangles without Disturbing the Topography

Several methods for re-tiling [23] and optimizing triangulated surface meshes [9] are available to remove thin triangles. However, these methods are often based on

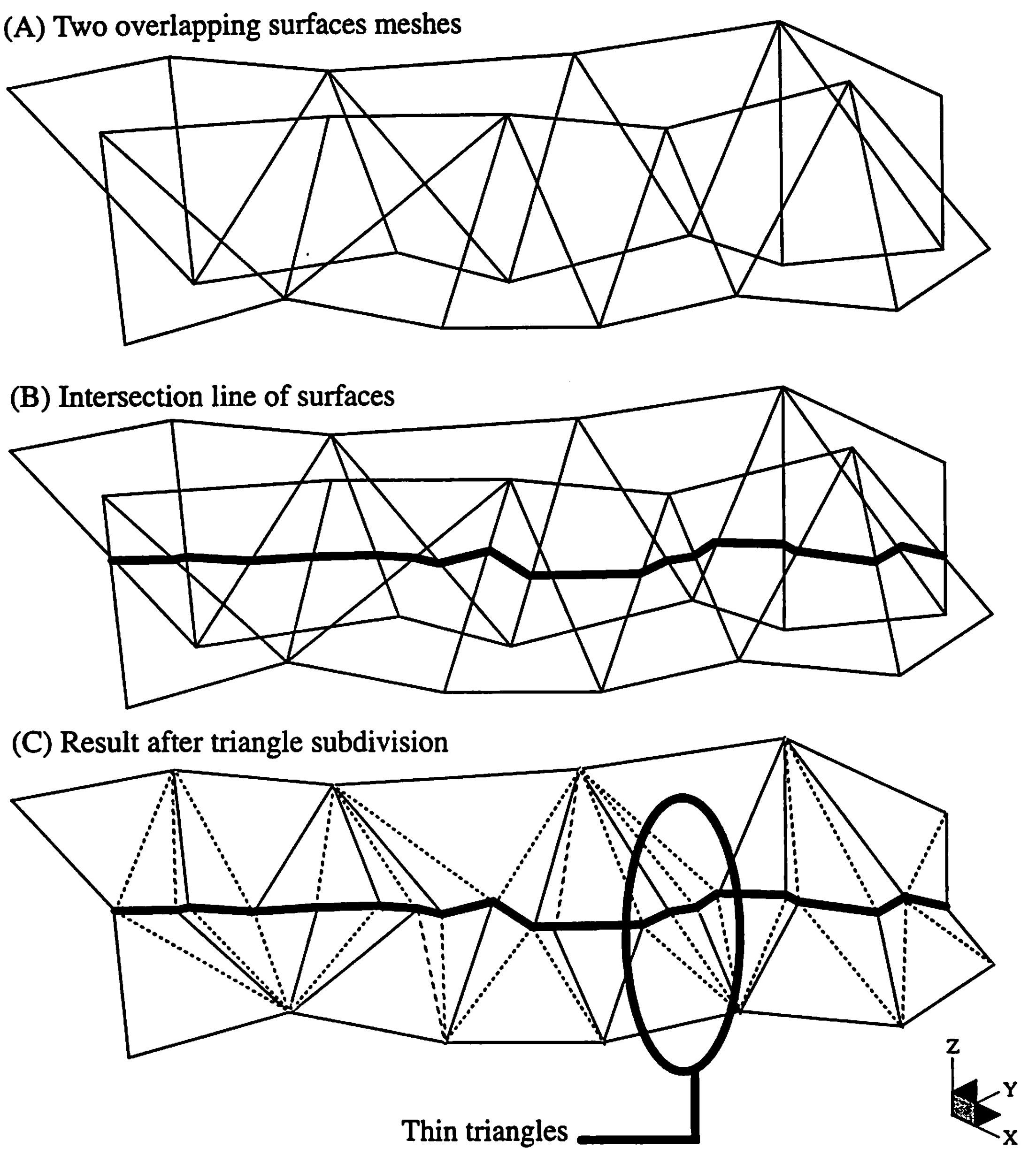

(A) Two overlapping surfaces meshes
(B) Intersection line of surfaces
(C) Result after triangle subdivision
Thin triangles
Z
Y
X

Figure 31 Illustration of how thin triangles are consequently formed during triangulated surface intersections.

minimizing an energy function dependent upon the distance between vertices and will either reposition the vertices of the original mesh or distribute a completely new set of vertices across the surface. For our purposes in TCAD, nodes in an evolving triangulated mesh carry more than positional information. They also carry simulation fields such as etch rates, deposition rates, visibility attributes, direction vectors, and others. For this reason, it is not desirable to significantly change the geometric position of nodes during a process step for the sake of thin triangle removal. In this light, the removal of thin triangles could be considered a goal-specific implementation of the decimation of triangulated meshes [19].

5.3.1 The Thin Triangle Removal Algorithm

In order to solve the problem of removing all thin triangles throughout a surface without altering its major topographic features, a flexible, query-based algorithm was implemented. This algorithm marches across the surface querying the individual triangles about their geometry and if their removal will alter the characteristic shape of the surface. When a thin triangle is encountered whose removal will not alter the macroscopic topography, surrounding triangles are either merged or collapsed together using one of several removal methods described in Chapter 2.

The algorithm is structured so as to remove the thin triangles as greedily as possible without altering the surface. For example, a thin triangle positioned in the center of a planar region could have all three of its vertices merged into one node thereby eliminating a total of four triangles; the thin triangle and its three adjacent triangles. This query and removal method is positioned at the top of the decision tree. Toward the middle of the decision tree are one and two node merge operations that eliminate two triangles; the thin triangle and its neighbor across the segment common to the merging nodes. At the bottom of the decision tree is the least greedy removal method; a simple edge flip that eliminates the occurrence of a thin triangle but doesn't reduce the surface's triangle count. In the form of a large flow chart, an outline of the algorithm is presented in Fig. 32.

5.3.2 Subtleties of the Removal Algorithm

When a thin triangle is encountered that cannot be eliminated by using any of the triangle removal methods of merging and collapsing, a simple edge flip is attempted. Realize that a simple edge flip doesn't actually remove any triangles, it simply

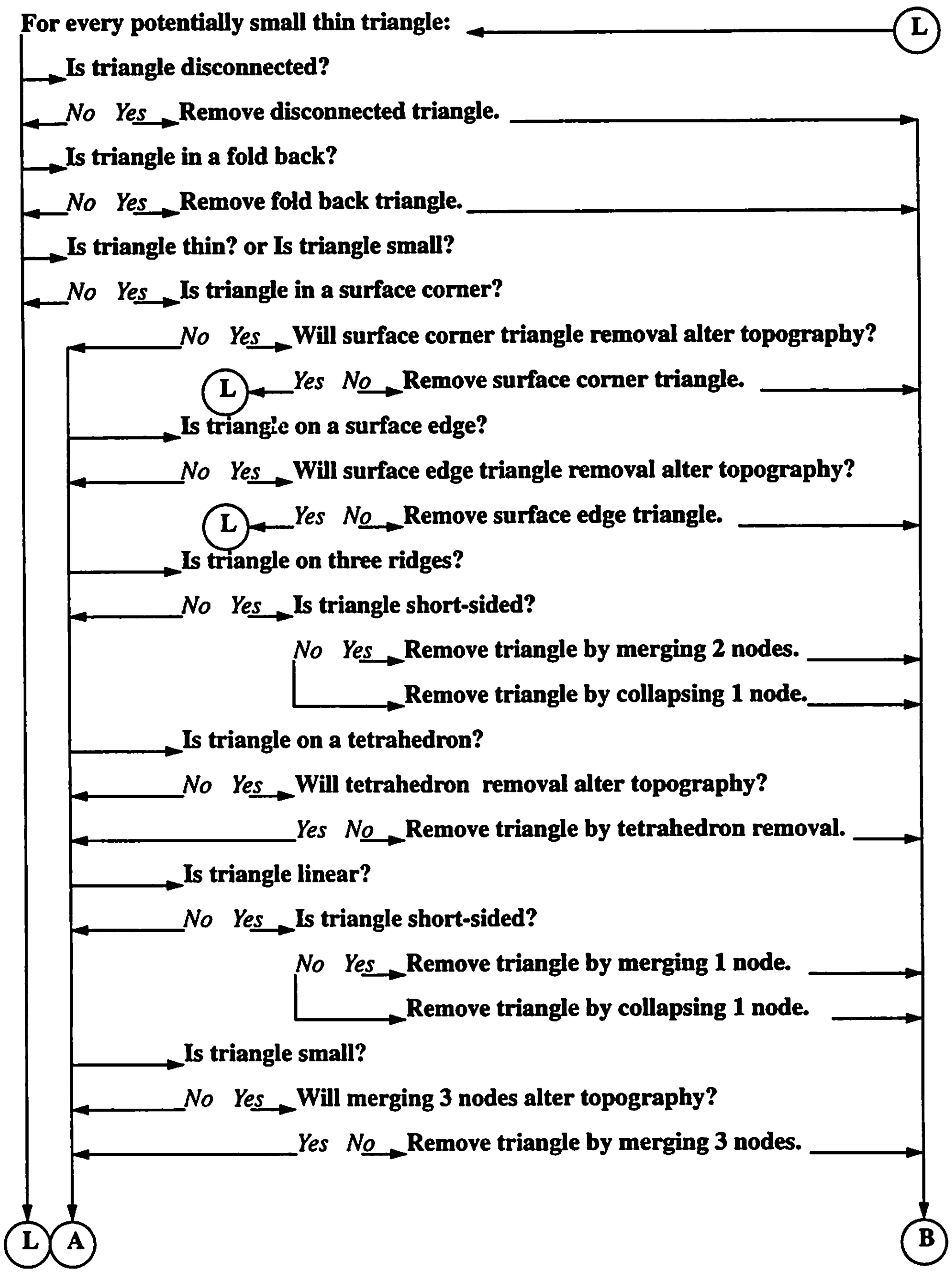

For every potentially small thin triangle:
L
Is triangle disconnected?
No Yes Remove disconnected triangle.
Is triangle in a fold back?
No Yes Remove fold back triangle.
Is triangle thin? or Is triangle small?
No Yes Is triangle in a surface corner?
No Yes Will surface corner triangle removal alter topography?
L Yes No Remove surface corner triangle.
Is triangle on a surface edge?
No Yes Will surface edge triangle removal alter topography?
L Yes No Remove surface edge triangle.
Is triangle on three ridges?
No Yes Is triangle short-sided?
No Yes Remove triangle by merging 2 nodes.
Remove triangle by collapsing 1 node.
Is triangle on a tetrahedron?
No Yes Will tetrahedron removal alter topography?
Yes No Remove triangle by tetrahedron removal.
Is triangle linear?
No Yes Is triangle short-sided?
No Yes Remove triangle by merging 1 node.
Remove triangle by collapsing 1 node.
Is triangle small?
No Yes Will merging 3 nodes alter topography?
Yes No Remove triangle by merging 3 nodes.
L
A
B

Figure 32 The thin triangle removal algorithm in the form of a flow chart.

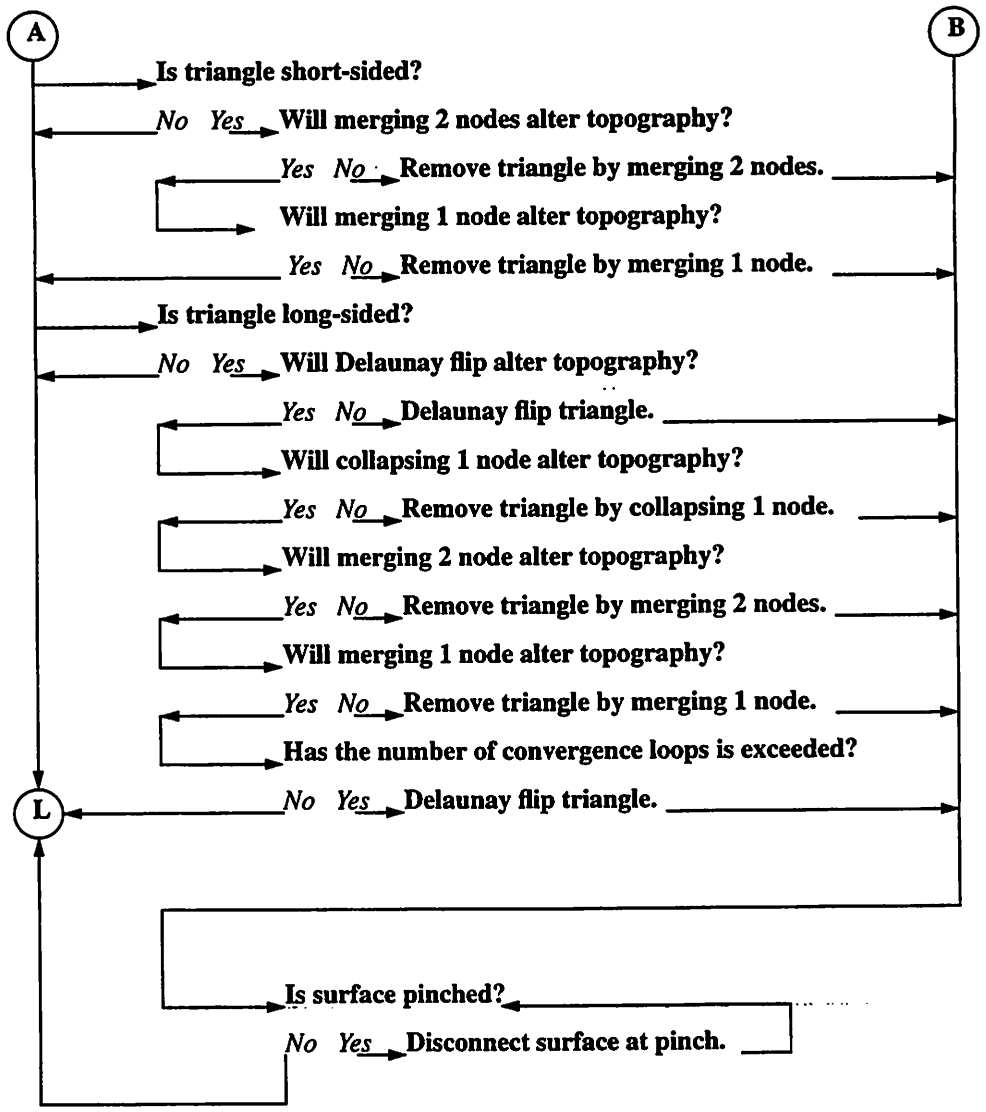

A
B
Is triangle short-sided?
No Yes Will merging 2 nodes alter topography?
Yes No Remove triangle by merging 2 nodes.
Will merging 1 node alter topography?
Yes No Remove triangle by merging 1 node.
Is triangle long-sided?
No Yes Will Delaunay flip alter topography?
Yes No Delaunay flip triangle.
Will collapsing 1 node alter topography?
Yes No Remove triangle by collapsing 1 node.
Will merging 2 node alter topography?
Yes No Remove triangle by merging 2 nodes.
Will merging 1 node alter topography?
Yes No Remove triangle by merging 1 node.
Has the number of convergence loops is exceeded?
L
No Yes Delaunay flip triangle.
Is surface pinched?
No Yes Disconnect surface at pinch.

Figure 32 (continued)

reestablishes the connectivity between adjacent triangles. Consequently, the potential situation of removing a thin triangle by edge flipping can push the problem from one triangle to another. This problem is compounded by the fact that triangles are visited in the order in which they appear in a linked list. The result of successive calls to the thin triangle removal algorithm could be a continuous exchange of thin properties between neighboring triangles. This problem is illustrated in Fig 33.

To help correct the problem of no net change when flipping thin triangles, the outermost loop in the thin triangles routine must process the triangles in both a forward and a reverse order. However, alternating the direction between forward and reverse is not sufficient; the direction of traversal must be randomly chosen. This will help avoid the occurrence of flipping thin properties between two, three, four, or more adjacent triangles.

The remaining subtlety was alluded to in the last paragraph; the need to process the outer loop of the removal algorithm more than once. In many cases, the removal of one thin triangle using any removal method can create a new thin triangle. Like the queue method used in the Delaunay triangulation algorithm (Sec. 3.2.1), triangles neighboring a thin triangle being removed are added to a queue. On succeeding traversals of the outer loop, it is more efficient to visit only the potentially thin triangles in the queue.

As noted in the flow chart of Fig. 32, a convergence limit for the number traversals of the outer loop is used. When a thin triangle is encountered after reaching the looping convergence limit, a Delaunay flip is used to get rid of the thin triangle despite the indication that it may alter the topography. This operation tends to work well since at the topography being altered at the late stage in the removal process is likely the creation of a ridge segment rather than the destruction of a ridge or corner. A looping convergence of 7 has worked well.

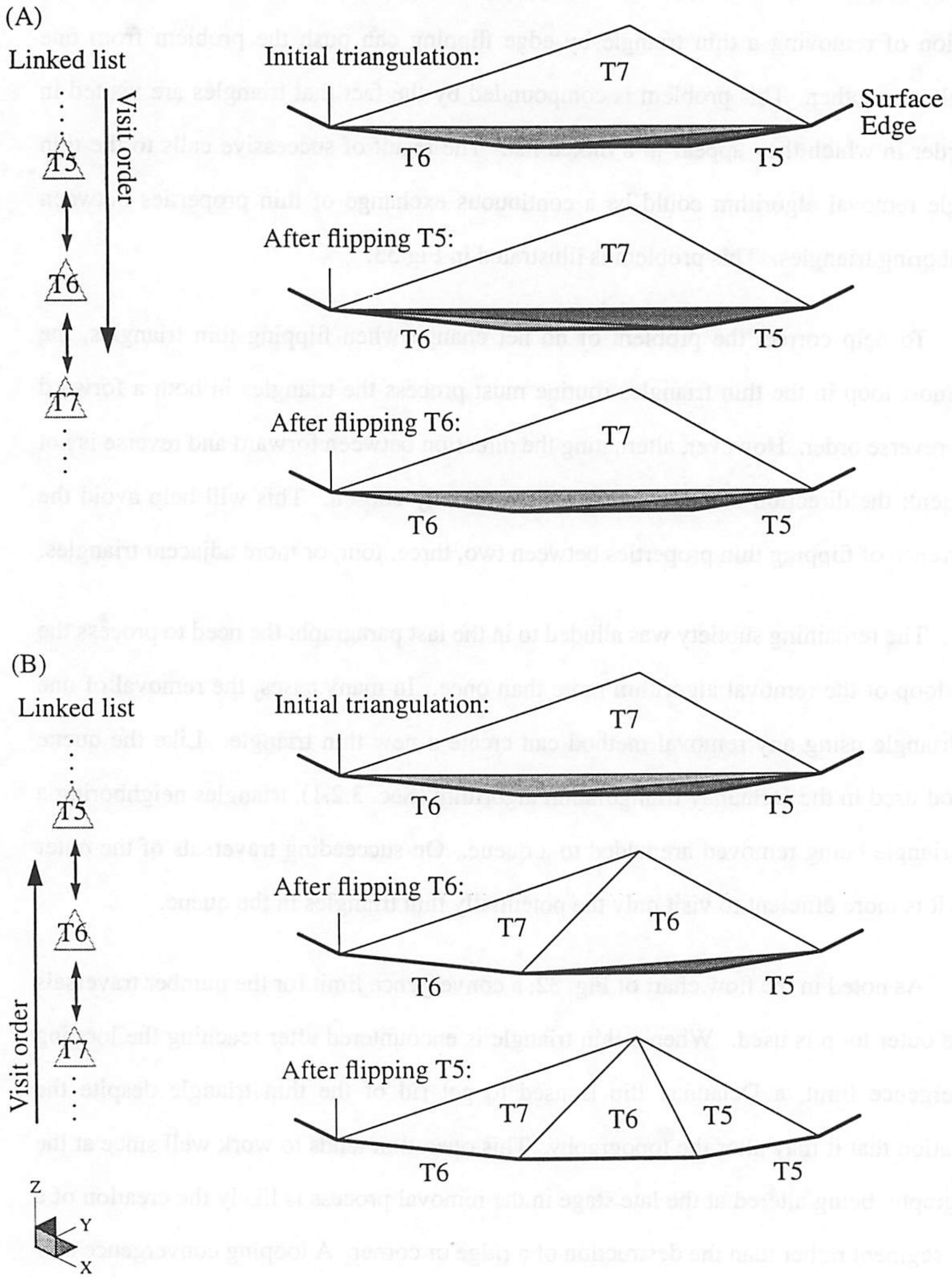


Figure 33 Traversing and flipping triangles T5 and T6 in a forward direction, thin properties are exchanged with no net result (A). Traversing and flipping triangles T5 and T6 in reverse order corrects this subtlety (B).

5.3.3 Results of the Thin Triangle Removal Algorithm

To demonstrate the success of the thin triangle removal algorithm, Fig. 34 shows close-ups of an intermediate stage of solid extraction from Chapter 7. before and after the thin triangles are removed. The presence of thin triangles in this example are due to surface intersections. Notice that the device topology is preserved despite the removal of thin triangles along ridges and corners. In this example, a surface of 1133 triangles was reduced to 731 triangles in 19 seconds on an IBM RISC 6000 workstation.

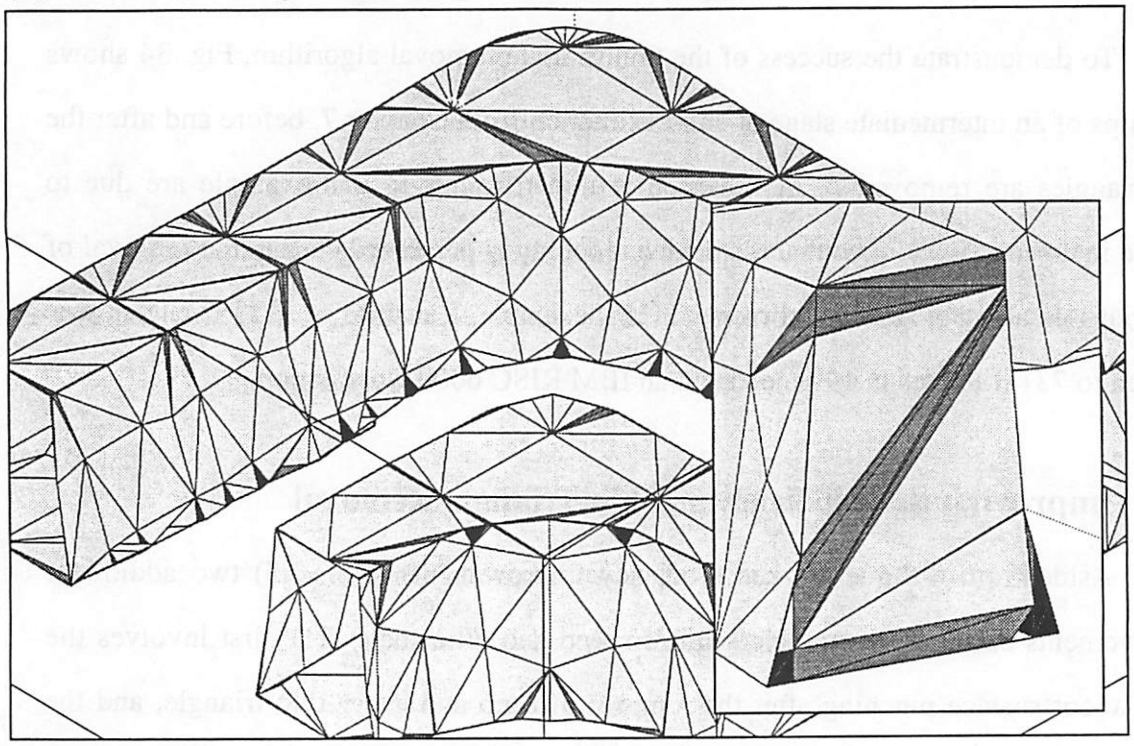
5.4 Improving the Efficiency of Thin Triangle Removal

Aside from the query-based efficiency improvements (Sec. 2.5) two additional improvements can increase the algorithm's speed and efficiency. The first involves the query about surface pinching after the removal of each and every thin triangle, and the second involves the search for inadvertently created thin triangles on successive loops of the removal algorithm.

As noted in the outline of the thin triangle removal algorithm (Sec. 5.3.1) a query of the surface is made for the presence of a pinch caused by the recent triangle removal. Because this query (Sec. 2.4.2.1) traverses the entire surface, most of the cpu time is wastefully spent checking for pinches far from the location of the recent triangle removal. A more efficient algorithm will only check for a surface pinch in the vicinity of the recently removed triangle.

Along the same lines of thought, searching for thin triangles throughout the entire surface following the first loop of the removal algorithm is also a waste of cpu time. On successive loops of the removal algorithm, only triangles in the vicinity of a previously removed triangle need to be checked for their acuity.

(A) Before thin triangle removal



(B) After thin triangle removal

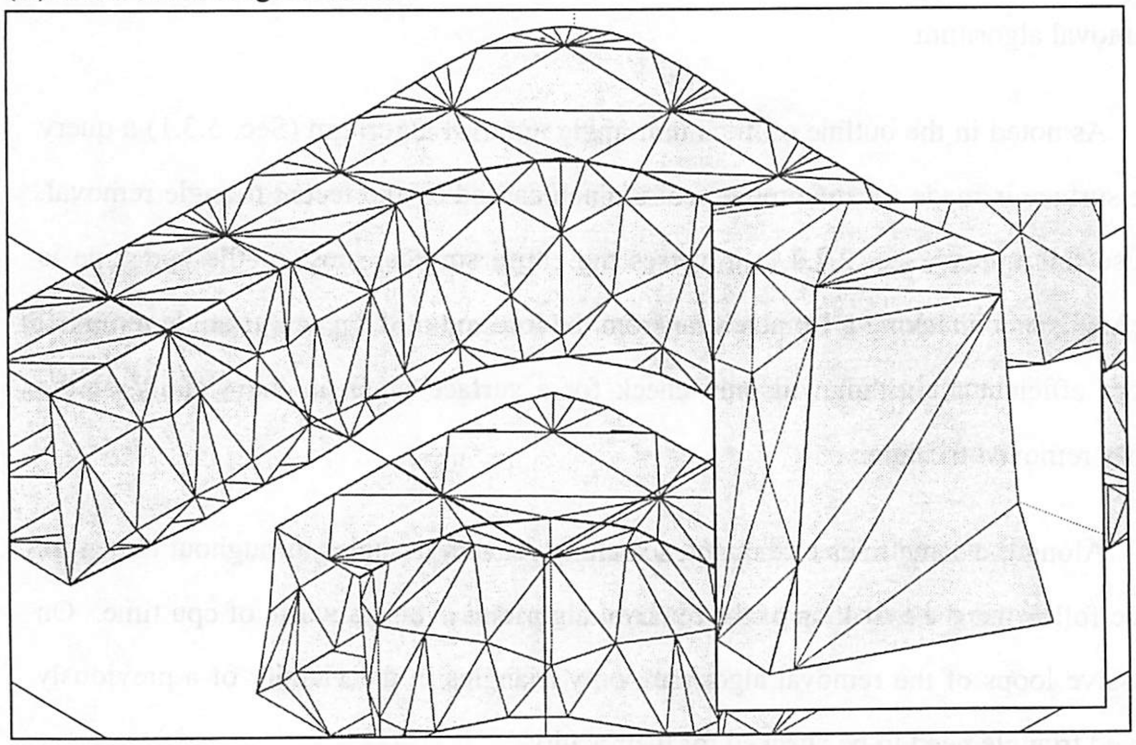


Figure 34 Close-ups showing darkened thin triangles resulting from surface intersections (A) and the result after thin triangle removal (B). (Due to the painters algorithm used for plotting, some back facing triangles near horizon ridges are improperly rendered atop of visible triangles.)

Chapter 6:

Surface Smoothing

6.1 What is a Rough Surface?

Smoothing surface data can have many different interpretations. In this chapter, two different algorithms for smoothing a surface mesh, triangle smoothing and node smoothing, are described and results are given. But before the algorithms are presented, a brief discussion of what constitutes a "rough" surface is necessary. This will establish a perspective whereby the smoothing techniques can be objectively reviewed.

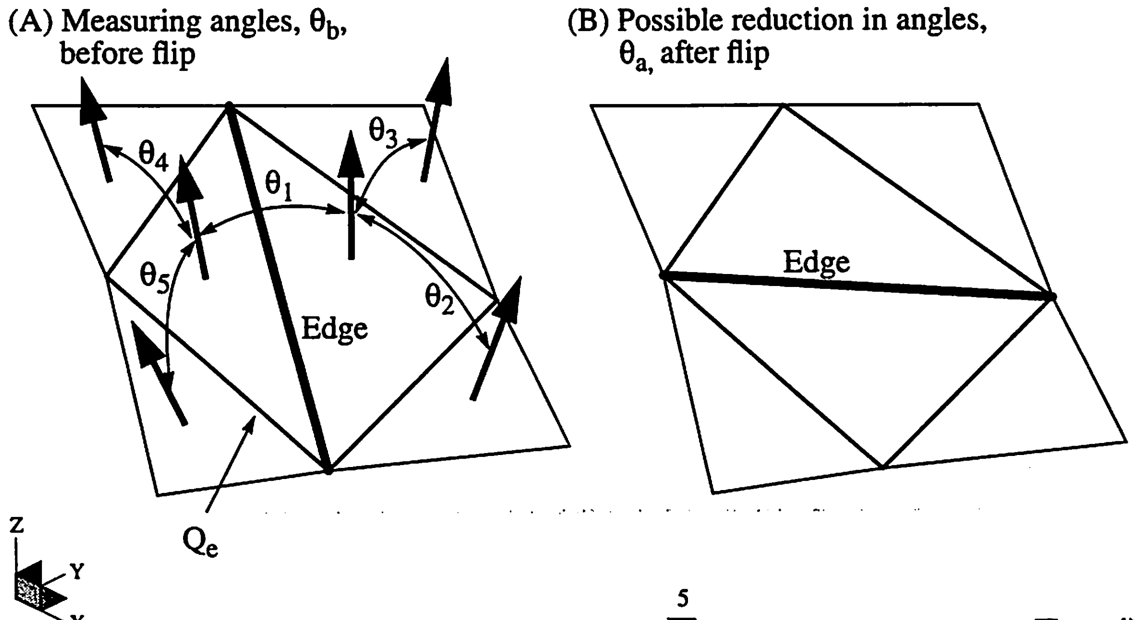
One interpretation of a rough surface, crenulation, was discussed by Helmsen's [8] and will be reused here. The easiest way to understand Helmsen's concept of a crenulated surface is through the use of an analogy. Imagine crumpling a sheet of paper into a ball and then unwrapping it. The crinkled sheet of paper is no longer perfectly planar. It contains many non-planar facets. To reduce the surface roughness, two algorithms taking different approaches are discussed below. An important distinction between the two algorithms is that one permits repositioning of the nodes while maintaining the mesh connectivity; the other maintains the mesh connectivity while reestablishing the node positions.

6.2 Smoothing Triangles

In the triangle smoothing algorithm, the position of surface nodes remain fixed. Nodes are *not* removed or repositioned. Therefore, the only freedom given to smoothing a surface is to manipulate the connectivity of the surface elements. By measuring the angle between normal vectors of all pairs of adjacent triangles and averaging, we have a way to quantify the surface roughness. This is the measure that triangle smoothing minimizes.

6.2.1 The Triangle Smoothing Algorithm

The triangle smoothing algorithm minimizes the average angle between normal vectors of adjacent triangles across a surface. To accomplish this goal, a procedure similar to the Delaunay edge flipping algorithm is employed. For all interior edges across a surface, a summation of the angles between the normal vectors of triangles adjacent to the two triangles connected to the edge in question is made. If the total measure for the angle is lower once the edge has been flipped, then the local roughness has been reduced, otherwise the edge is flipped back. Figure 35 shows a schematic of how the angles are measured and the result after an edge has been flipped.


(A) Measuring angles, θ_b, before flip
θ_4
θ_1
θ_3
θ_5
θ_2
Edge
Q_e
Z
Y
X
(B) Possible reduction in angles, θ_a, after flip
Edge

$$\theta_a, \theta_b = \sum_{i=1}^{5} \theta_i \qquad \textbf{(Equ. 4)}$$

Note: If an adjacent triangle is NULL, the angle between normals is taken as zero.

Figure 35 A measure of the local roughness (non-planarity) can be made by summing the angles between triangle normals. Using an algorithm based on the edge flipping algorithm, the roughness can be reduced.

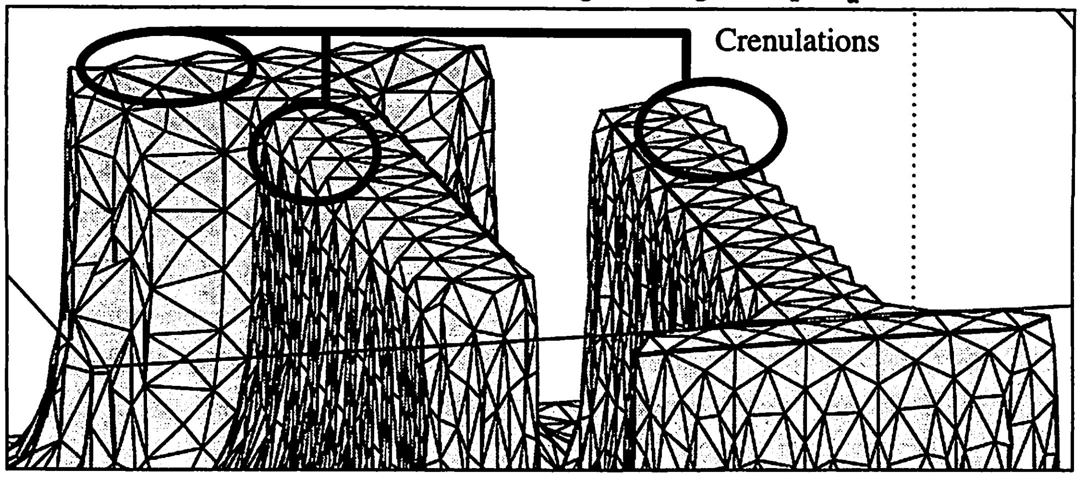
With reference to the nomenclature of Fig. 35 and Equ. 4 for calculating the local roughness of a surface, the triangle smoothing algorithm is outlined in the following pseudo-code:

```
for each interior surface Edge
{
   add Edge to a Queue
}

while Queue is not empty
{
   remove the first Edge from Queue

   calculate the summation of the local angles $\theta_b$ of
   Quadrilateral $Q_e$ before flipping Edge

   flip Edge

   calculate the summation of the local angles $\theta_a$ of
   Quadrilateral $Q_e$ after flipping Edge

   if ($\theta_a < \theta_b$) then
   {
      add to Queue each side of $Q_e$ not already in the Queue and
      not on the surface edge
   }
   else
   {
      flip Edge back to its original state
   }
}
```

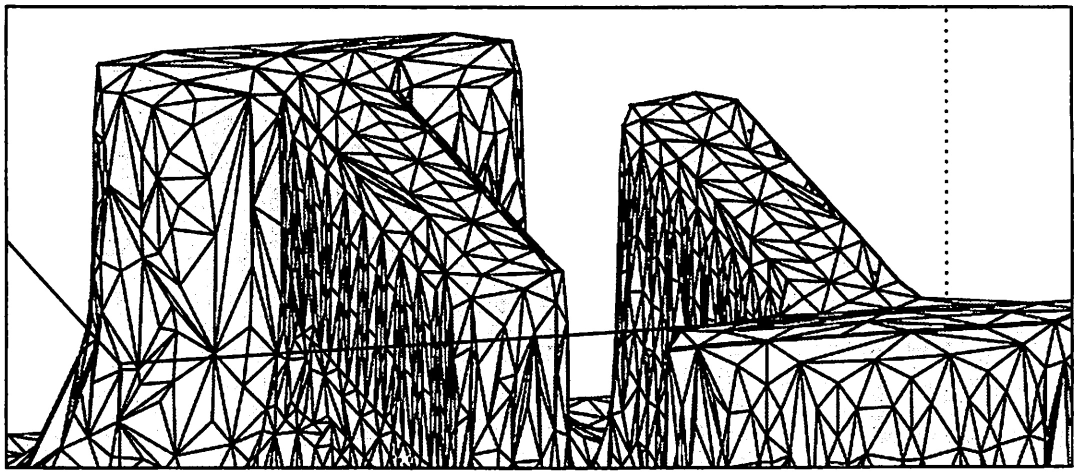
6.2.2 Example of Smoothing Triangles

To demonstrate how well the triangle smoothing utility works, a surface with adjacent T-shaped etchings was simulated using SAMPLE-3D. At the top of the etchings shown in Fig. 36 A, the triangles appear rough. In fact, many of the segments surrounding the triangles along the top satisfy the ridge segment criteria. Recall Sec. 2.3.3.6 (Is Segment on a Ridge?). Because many of the triangles are surrounded by ridge segments, the triangle smoothing algorithm is a good way to redefine the ridge. Having smoothed the triangles as shown in Fig. 36 B, the ridge segments are consequently redefined, and a Delaunay triangulation, presented in Chapter 3, can be performed.

(A) Original surface with rough (crenulated) triangles along the top. $\theta_a = 14.39°$.



(B) After smoothing triangles, the crenulation is gone. $\theta_b = 6.37°$.



(C) Having reestablished the ridge line, a Delaunay operation improves the triangulation.

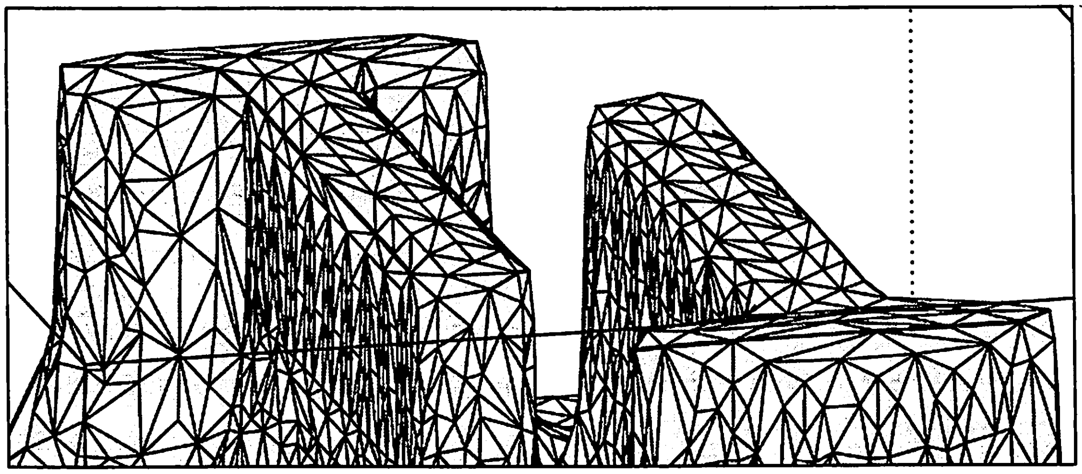


Figure 36 The triangle smoothing algorithm is used to remove the rough crenulated edges along the top of the T-shaped etchings. The average angle between triangle normals of adjacent triangles is reduced 56%.

6.3 Smoothing Nodes

In the node smoothing algorithm, surface nodes are free to be repositioned. This is a precautionary point for TCAD simulation tools. Recall that these tools often keep auxiliary information within the structure of a node that is likely dependent on the current geometric coordinates of the node. This is true for lithography simulators that use "ray-trace" algorithms for advancement. Therefore a smoothing algorithm that repositions nodes can be dangerous. On the other hand, an advancement algorithm based on pushing facets (triangles) for various etching processes [18] will tolerate node relocation.

6.3.1 The Node Smoothing Algorithm

The node smoothing algorithm is very simple and is outlined in pseudo-code below. The primary idea is to relocate the nodes to the geometric center of its surrounding nodes while paying particular attention to nodes located on corners, ridges, and surface edges whose topology must be preserved.

```
do
{
   for each Node NOT connected to a ridge or an edge segment
   {
      relocate Node to the geometric center of its surrounding
      nodes
   }

   for each Node on a surface edge
   {
      relocate Node to an interpolated location between the
      Node's two neighbor nodes on the same surface edge
   }

   for each Node on a surface ridge
   {
      relocate Node to an interpolated location between the
      Node's two neighbor nodes on the same ridge
   }
} while (the maximum node relocation distance > a tolerance)
```

6.3.2 Choice of Interpolation Methods

Of interest in the node smoothing algorithm is the choice of interpolation methods for smoothing nodes on a ridge or surface edge. Two methods were investigated: circular arc interpolation, and a convex hull interpolation method. The second method is an ad-hoc technique designed to overcome the flaws that occur in the circular arc method. After reviewing the two methods of interpolation that follow, inspect Fig. 41 which compares results from both interpolation methods on a surface containing a well defined ridge line.

6.3.2.1Circular Arc Interpolation

The circular arc interpolation method was first chosen as the interpolation method for smoothing nodes along surface ridges and edges. In the plane created by three consecutive nodes along a surface ridge or edge, this method fits a circle to nodes and then bisects the subtended arc by relocating the center node to the bisecting location. Figure 37 demonstrates how the interpolated node is found. When the three nodes are colinear, the center node is relocated to the midpoint.

This method works satisfactorily when the radius of the circle is somewhat larger than the distances between the interpolating nodes. However, when this condition is not satisfied and the radius of the circle is comparable to the distances between the interpolating nodes, then the circular arc method can backfire. The ridge or edge line can roughen. See Fig. 38 for a schematic description of the circular arc interpolation method and how it can roughen rather than smooth the nodes.

6.3.2.2Convex Hull Interpolation

If the interpolated node lies within the convex hull of the three interpolating nodes, the flawed results depicted in Fig 38 C can be overcome. Making this observation leads one to believe that using a quadratic b-spline for interpolation would be a good idea. The

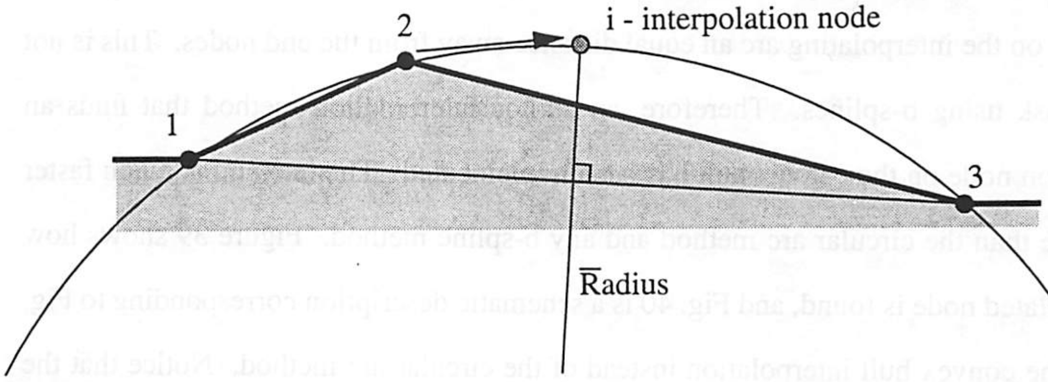


Figure 37 Demonstration of how to find an interpolation node on the interpolating circle of three successive ridge or edge segments that is equidistant from the end nodes.

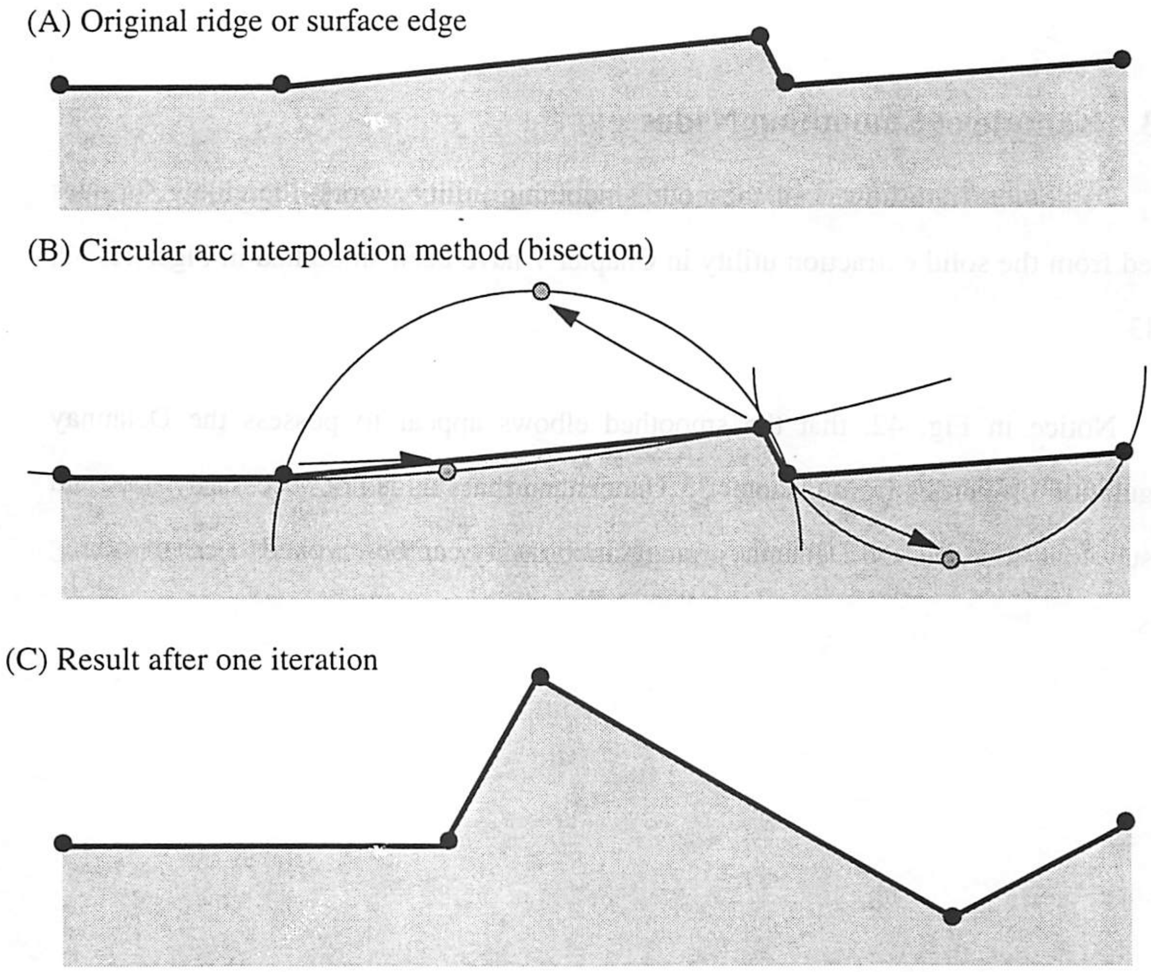

(A) Original ridge or surface edge
(B) Circular arc interpolation method (bisection)
(C) Result after one iteration

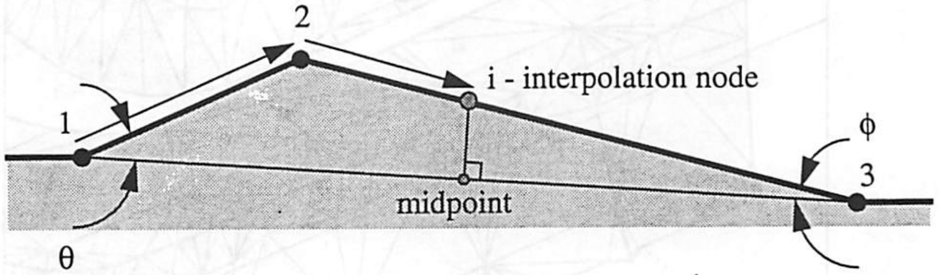
Figure 38 Schematic of the circular arc interpolation method used on nodes along a ridge or surface edge. Notice that this interpolation method can actually roughen rather than smooth the line.

only difficulty of using b-splines in this circumstance is that we would like the interpolated node to lie on the interpolating arc an equal distance away from the end nodes. This is not a trivial task using b-splines. Therefore, an ad-hoc interpolation method that finds an interpolation node on the convex hull has been implemented. This method is much faster to compute than the circular arc method and any b-spline method. Figure 39 shows how the interpolated node is found, and Fig. 40 is a schematic description corresponding to Fig. 37 using the convex hull interpolation instead of the circular arc method. Notice that the convex hull methods does not backfire, the ridge line is successfully smoothed. Although not demonstrated in a figure, the convex hull and circular arc interpolation method will produce nearly equivalent results when the radius of the interpolating circle is sufficiently large.

6.3.3 Example of Smoothing Nodes

To demonstrate how well the node smoothing utility works, the elbow surfaces created from the solid extraction utility in Chapter 7 have been smoothed in Figs. 41, 42, and 43.

Notice in Fig. 42, that the smoothed elbows appear to possess the Delaunay triangulation properties from Chapter 3. Understand that this is not necessarily true, and to insure that it is true, the Delaunay triangulation utility can be invoked after smoothing nodes.



$$|r_{12}|\cos\theta + |r_{2i}|\cos\theta = \frac{1}{2}|r_{13}| \qquad \textbf{(Equ. 5)}$$

Figure 39 Demonstration of how to find an interpolation node on the convex hull of three successive ridge or edge segments that is equidistant from the end nodes.

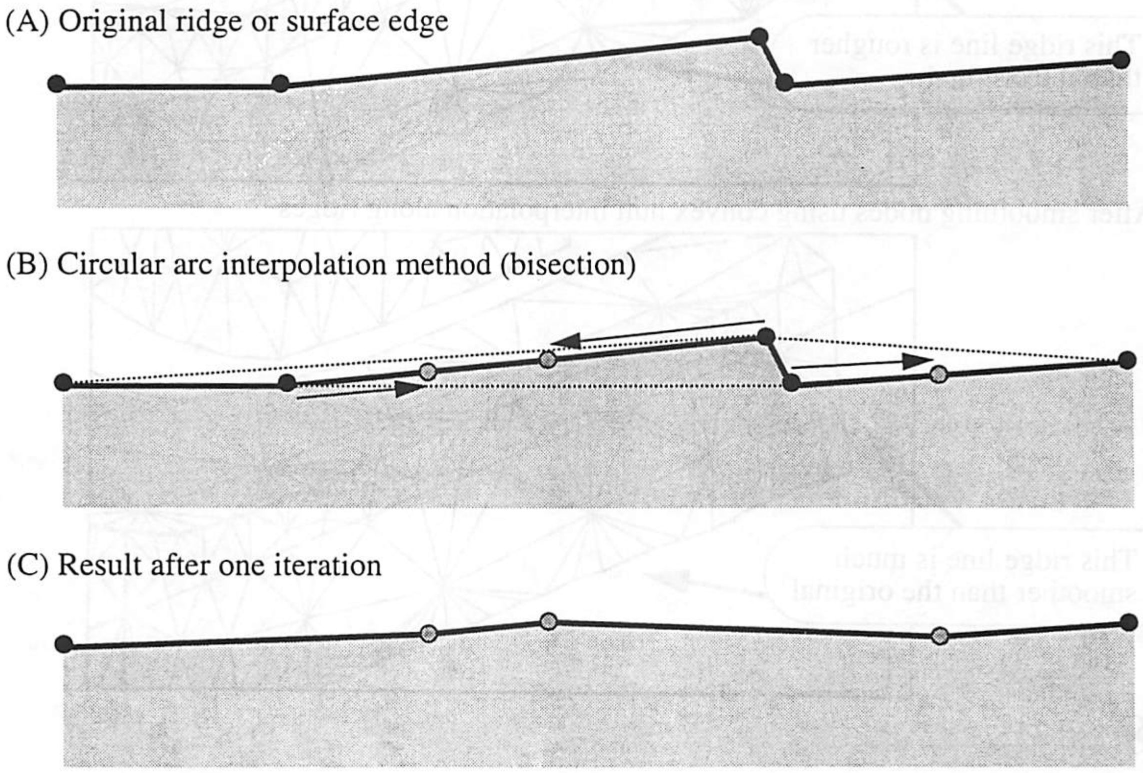

(A) Original ridge or surface edge
(B) Circular arc interpolation method (bisection)
(C) Result after one iteration

Figure 40 Schematic of the convex hull interpolation method used on nodes along a ridge or surface edge. Notice that this interpolation method is an improvement from the circular arc method in Fig. 38.

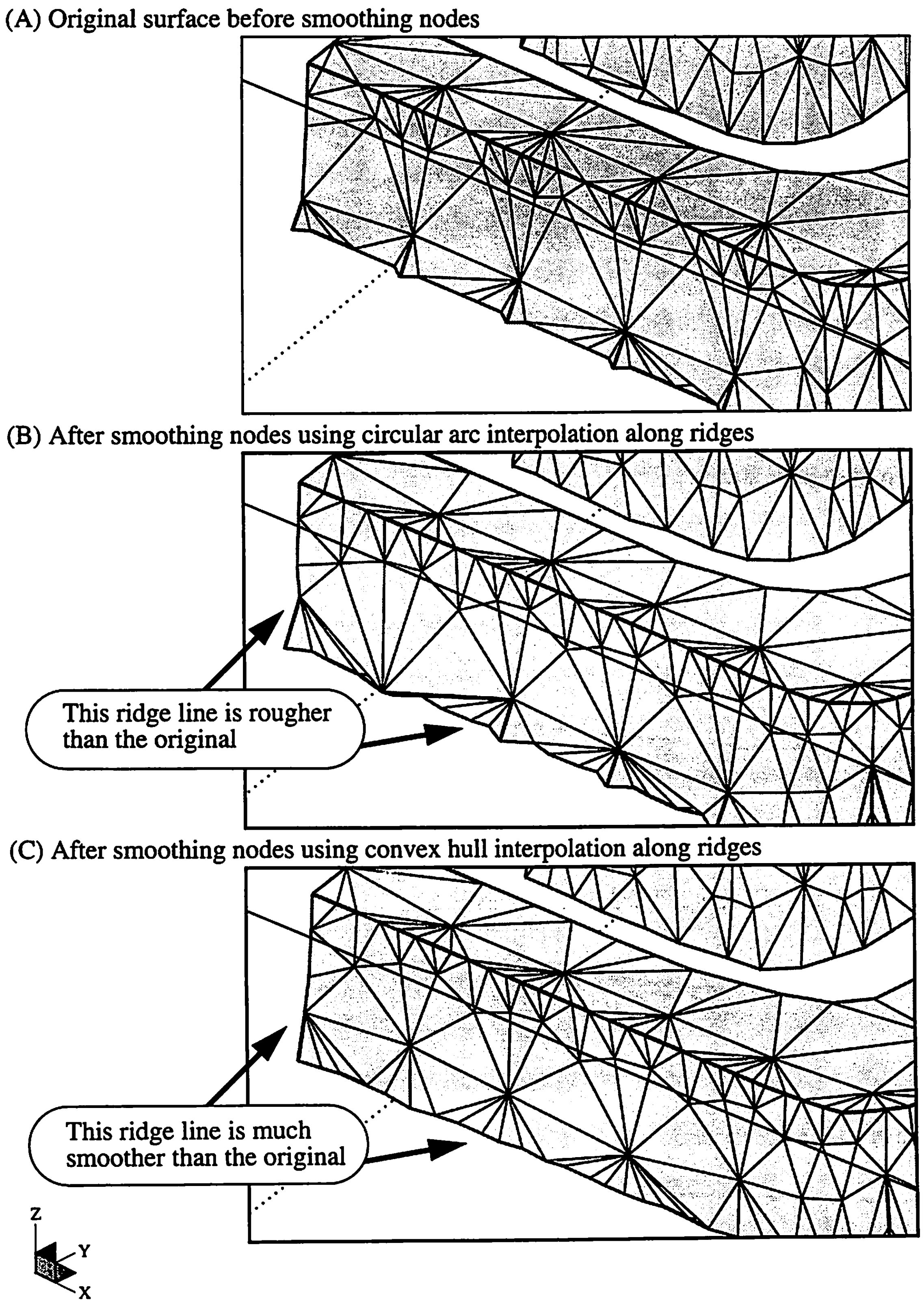
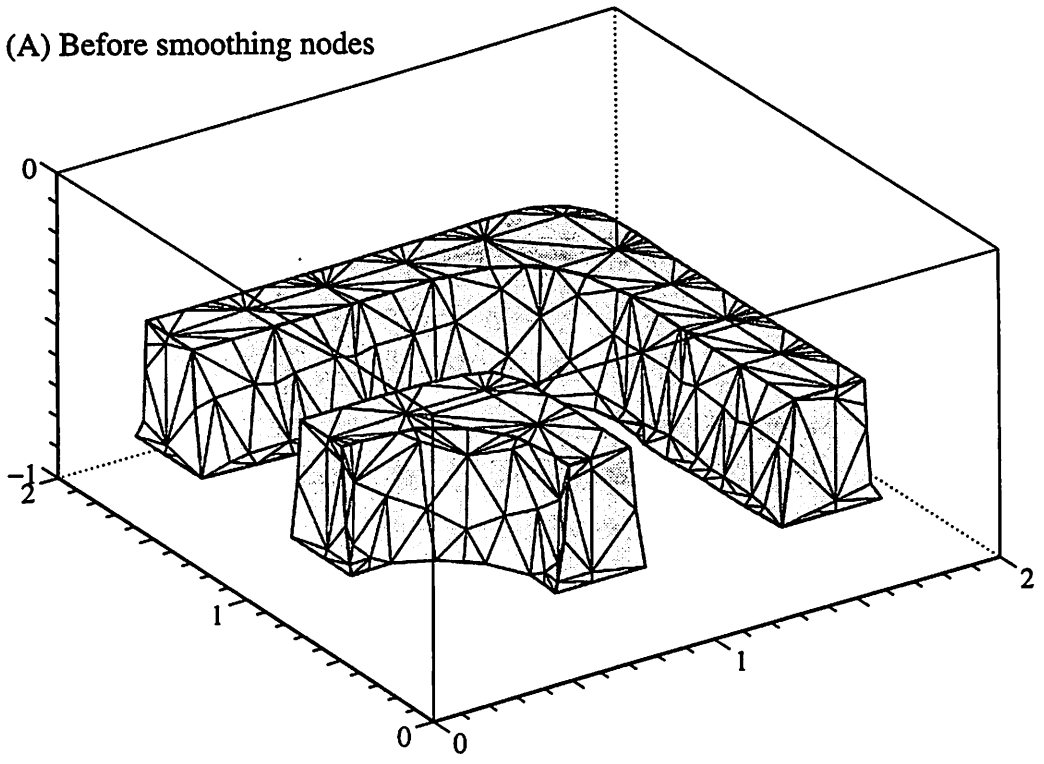

(A) Original surface before smoothing nodes
(B) After smoothing nodes using circular arc interpolation along ridges
This ridge line is rougher than the original
(C) After smoothing nodes using convex hull interpolation along ridges
This ridge line is much smoother than the original
Z
Y
X

Figure 41 Close-ups of a surface with a well defined ridge line showing a comparison of the interpolation methods for smoothing nodes. Notice the circular arc interpolation method can roughen rather than smooth.



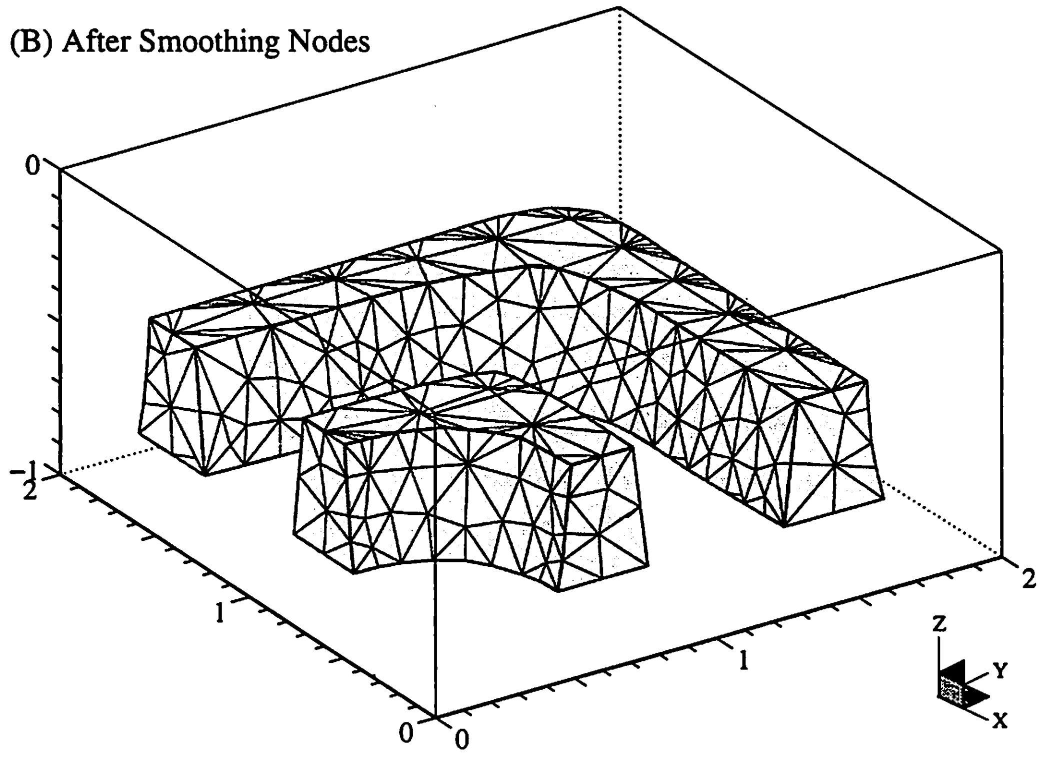


Figure 42 Before (A) and after (B) node smoothing. Notice that the smoothed surfaces appear to posses the Delaunay triangulation properties. Units are in microns.

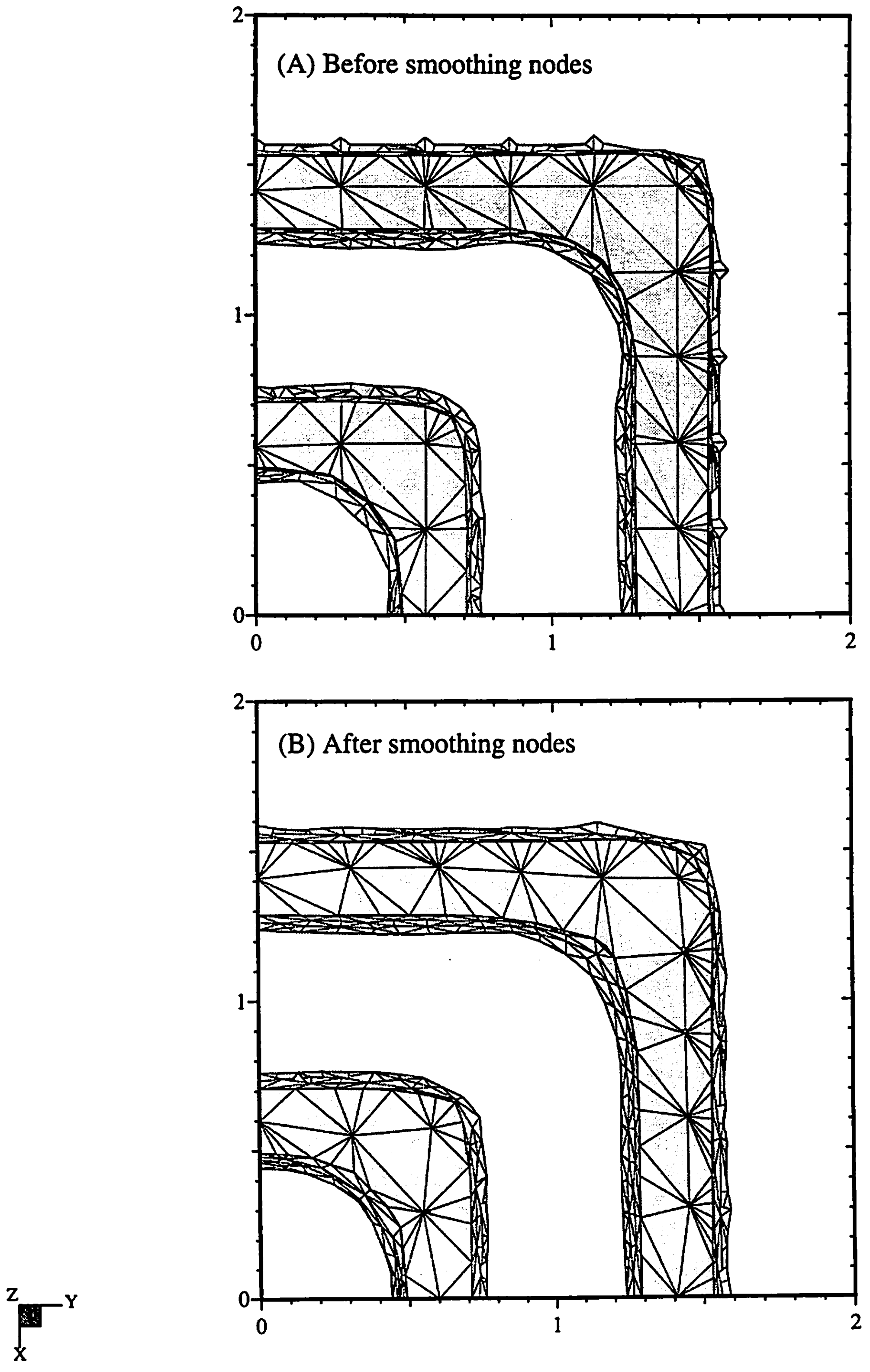


Figure 43 An aerial view of the elbow shaped surfaces before (A) and after (B) node smoothing. Notice that bumps along the outer ride of the larger elbow have been smoothed out. Units are in microns.

Chapter 7:

Extraction of Solids from a Single Surface

7.1 Introduction to the Solid Elbow Extraction Problem

The subject of this chapter, the extraction of solid geometries from a single triangulated surface, details a solution to the original problem that inspired and required the implementation many of the surface modeling utilities presented in this thesis. This chapter is organized in a story-like manner whereby the procedure to extract solids is outlined and applied to a real example; the extraction of solid elbows. At intermediate stages in the procedure, necessary algorithms are discussed and applied to the elbow example. The intended purpose for solid extraction could vary from one application to another. The application that required this work is discussed in Chapter 8.

Presented in Fig. 44 is an introduction to the solid elbow extraction problem. The

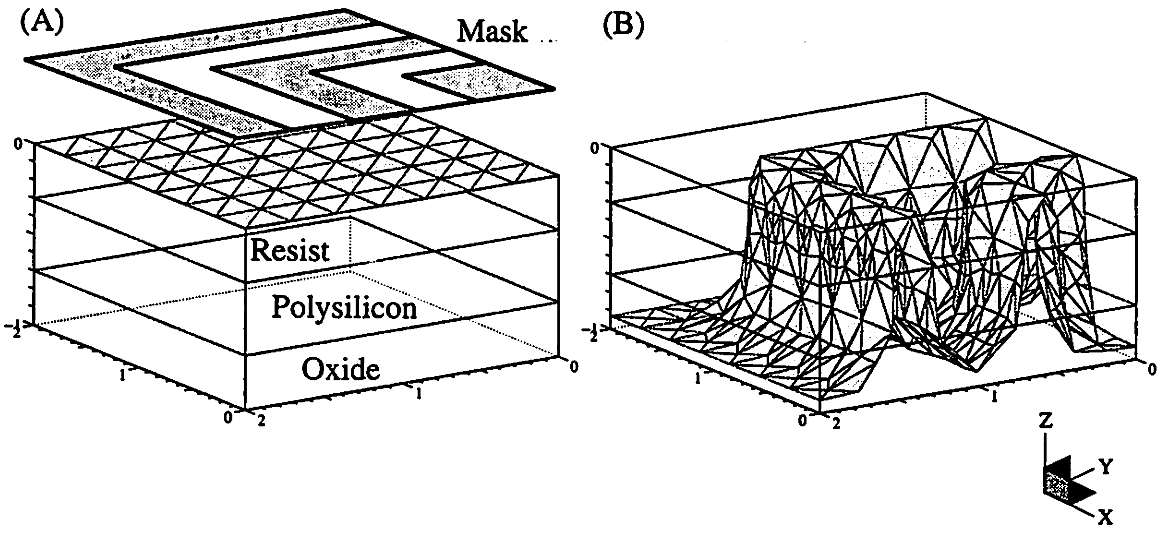


Figure 44 The formation of the solid polysilicon elbows represented by a single advanced surface generated from a lithography and etching process using SAMPLE-3D. Units are in microns.

elbow shaped mask and simulation volume containing a layer of negative resist, polysilicon, and oxide are given in Fig. 44 A. A simple lithography and etching process using SAMPLE-3D was used to advance the initially flat surface into the conformal elbow shape of the wafer materials represented in Fig. 44 B. The process began by calculating an aerial image, bleaching the resist, and developing the resist in a lithography step. The resulting resist pattern was then transferred to the polysilicon by a reactive ion etching simulation. The surface advanced in time and was influenced by self-shadowing and bombardment [18]. The layer of oxide was used as a substrate allowing for a complete etch through the polysilicon layer. The elbow pattern emerged in the single triangulated surface representation shown in Fig. 44 B. This result is the starting point for the extraction procedure. The objective is to extract the volumes of remaining polysilicon by representing them as enclosed triangulated surfaces.

7.2 Overview of the Extraction Procedure

To extract the solid geometry from the polysilicon layer, a constructive procedure of intersecting the simulated surface with "cutting planes" is used. Altogether, six different cutting planes are used. Two of them represent the interface between the layer of resist with the polysilicon and the polysilicon with the layer of oxide. The remaining four cutting planes are the simulation boundary planes in the X and Y axis directions. Although this example makes use of planar cutting planes, non-planar surfaces could also be used for cutting.

After intersecting the simulated surface with the initial two cutting planes, unwanted surface corresponding to the layer of resist and oxide is identified and removed. To do this, deloop algorithms are employed [7]. This removal procedure is physically analogous to the chemical reaction of stripping off the resist and the removal of oxide by chemical means or sputter etching. To complete the procedure, the same deloop algorithms are again employed to remove the unwanted surface after intersection with the simulation boundary planes.

The reason for not simultaneously intersecting the simulated surface with all six cutting planes is two fold. First, separating the intersection and unwanted surface removal into three steps avoids the three plane intersection problem. Second, after intersecting two triangulated surfaces, irregular thin shaped triangles result that must be corrected before proceeding with the intersection of the next set of cutting planes.

Finally, the enclosed solids are represented by a single linked list of triangles and must be separated into individual lists corresponding to each of the formed solids. Following this step, each enclosed surface is oriented with outward pointing normals.

Below is a simplified account of the extraction steps:

Intersection Loop

1 Introduce 6 new flat surfaces (2 at a time) that represent the resist/poly interface, poly/oxide interface, and the 4 simulation boundaries.

2 Intersect the new surfaces with the original etched surface.

3 Combine multiple intersecting surfaces into one surface.

4 Remove the unwanted surface.

5 Identify and remove thin triangles without altering topographical features.

6 Identify the enclosed surfaces that remain.

7 Orient triangles to correct for a proper outward surface normal.

7.2.1 Initial Surface Intersections and Combinations

In Fig. 45, the first three steps of the extraction procedure are illustrated. In Fig. 45 A, the original surface is simultaneously plotted with the cutting planes. Accompanying the plot is a representation of the linked list of triangles corresponding to the three individual surfaces. In Fig. 45 B, the surfaces are intersected. The corresponding linked list representation shows that the triangles from each of the three surfaces have

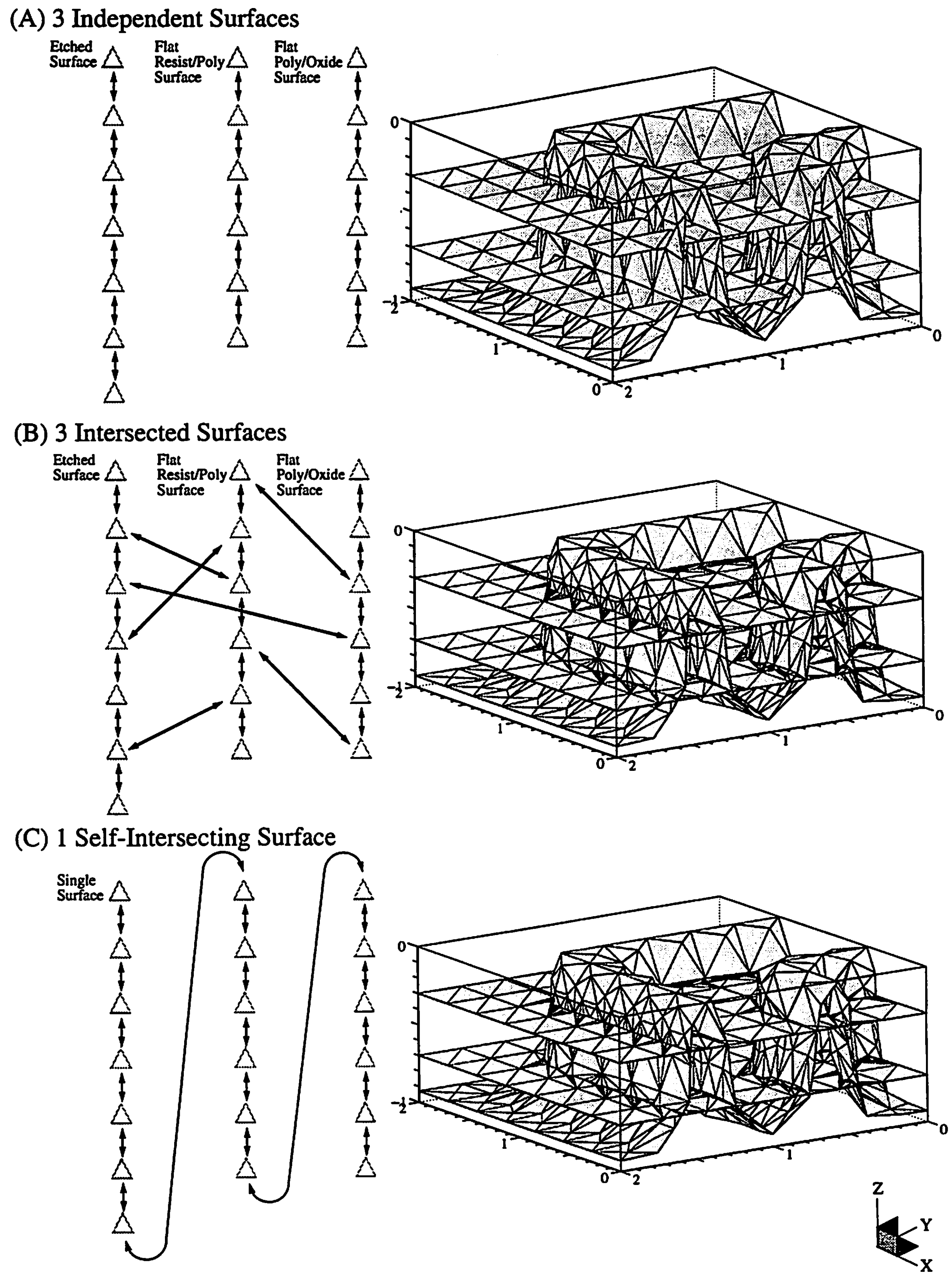


Figure 45 Illustration of how the original surface is intersected and combined with the first two cutting planes (steps 1,2,3, in the extraction procedure).

connectivity relationships to the other surfaces' triangles. Finally in Fig. 45 C, the three linked lists are concatenated into one linked list. At this point, the plot of Fig. 45 C represents a single self-intersecting surface containing many thin triangles resulting from the surface intersections.

7.2.2 Removing Unwanted Surface

Having intersected and combined multiple surfaces into a single self-intersecting surface as shown in the plot of Fig. 45 C, the next step in the extraction process is to remove unwanted portions of the surface. To accomplish this step, a variation of the deloop algorithms developed by John Helmsen for lithography simulation [7] are employed.

The first step in delooping a self-intersecting surface is to identify at least one triangle that will either remain with the surface or will be removed with the loops. In Helmsen's work, he assumed that the list head triangle, positioned in the corner of the simulated surface, was always positioned on a valid portion of the surface. Therefore the list head triangle was marked for "keeping". In the solid extraction example, this assumption is not valid. In fact, the list head triangle could be positioned anywhere. Identifying a triangle for "keeping" or "removal" is where this variation in Helmsen's algorithm lies.

A simple method, although not a fully robust method, to find triangles for keeping or removing is to locate nodes at a corner of the simulation region. In a corner, each node is positioned equally in X and Y but has different Z coordinates. By comparing the value of the Z coordinate of the node corresponding to the original surface with the cutting plane nodes, three triangles (each adjacent to one of the corner nodes) can be marked for keeping or removal. Recognize that the cutting planes bound the layer of material we wish to extract. Therefore, if the corner node corresponding to the original surface is positioned above the extraction layer, it is marked for removal while the cutting plane nodes are marked for keeping. If the corner node corresponding to the original surface is positioned between the cutting planes, then it is marked for keeping while the upper cutting plane node is marked for removal, and the lower cutting plane node is marked for keeping. Lastly, if the corner node .corresponding to the original surface is positioned below the lower cutting plane node, then all three nodes are marked for removal. These three cases are presented in the 2 D illustration of Fig. 46.

Having identified three triangles (each adjacent to one of the corner nodes) for keeping or removing, the delooping algorithm employs a breadth-first search method to remove or keep triangles across the surface until segments on the intersection line is reached. Crossing the intersection line, a new breadth-first search proceeds and does the opposite keep or remove operation to the triangles. A complete description of the delooping algorithm is found in Helmsen's work [6,7].

7.2.3 Removing Thin Triangles and Repeating the Intersection Loop

Now that the process for intersecting, combining, and removing unwanted surface has been established, thin triangles resulting from the surface intersections are removed. Because the thin triangle removal procedure has many interesting details, it is the subject of Chapter 5. For now, let us assume that thin triangles throughout the surface are removed while preserving the characteristic features of the surface such as edges, ridges, and corners.

The process is now repeated twice; once with two new cutting planes representing the X simulation boundaries, and once with two new cutting planes representing the Y simulation boundaries. Intermediate steps in the intersection loop are given in Fig. 47. It is important to note that the surface preparation is handled slightly different during the second and third pass of the intersection loop just before intersecting and delooping.

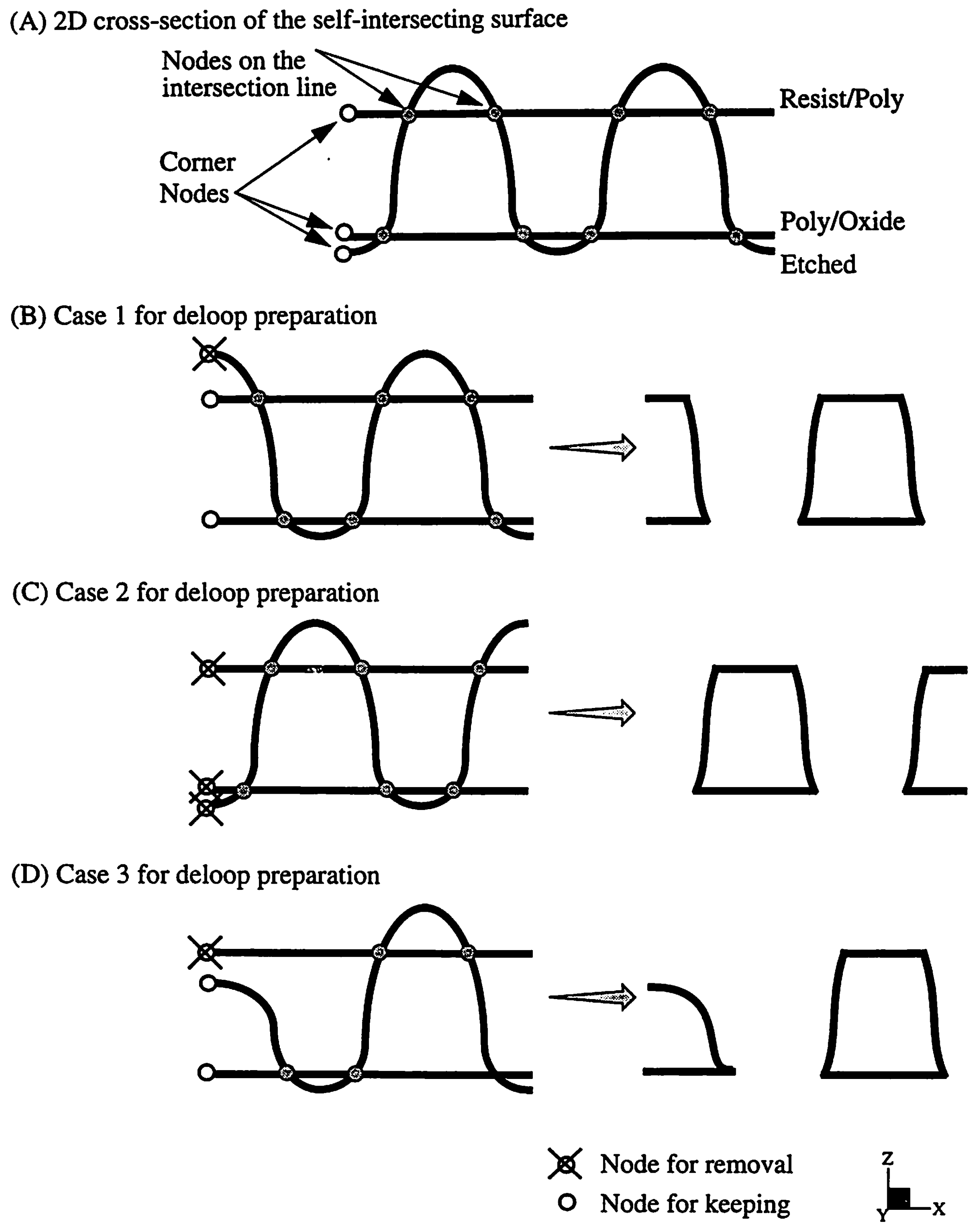

(A) 2D cross-section of the self-intersecting surface
Nodes on the intersection line
Corner Nodes
Resist/Poly
Poly/Oxide
Etched
(B) Case 1 for deloop preparation
(C) Case 2 for deloop preparation
(D) Case 3 for deloop preparation
Node for removal
Node for keeping
Z
Y
X

Figure 46 Identifying three triangles/nodes for "keeping" or "removal". These are used in a preparation step for removing unwanted surface using Helmsen's deloop algorithms.

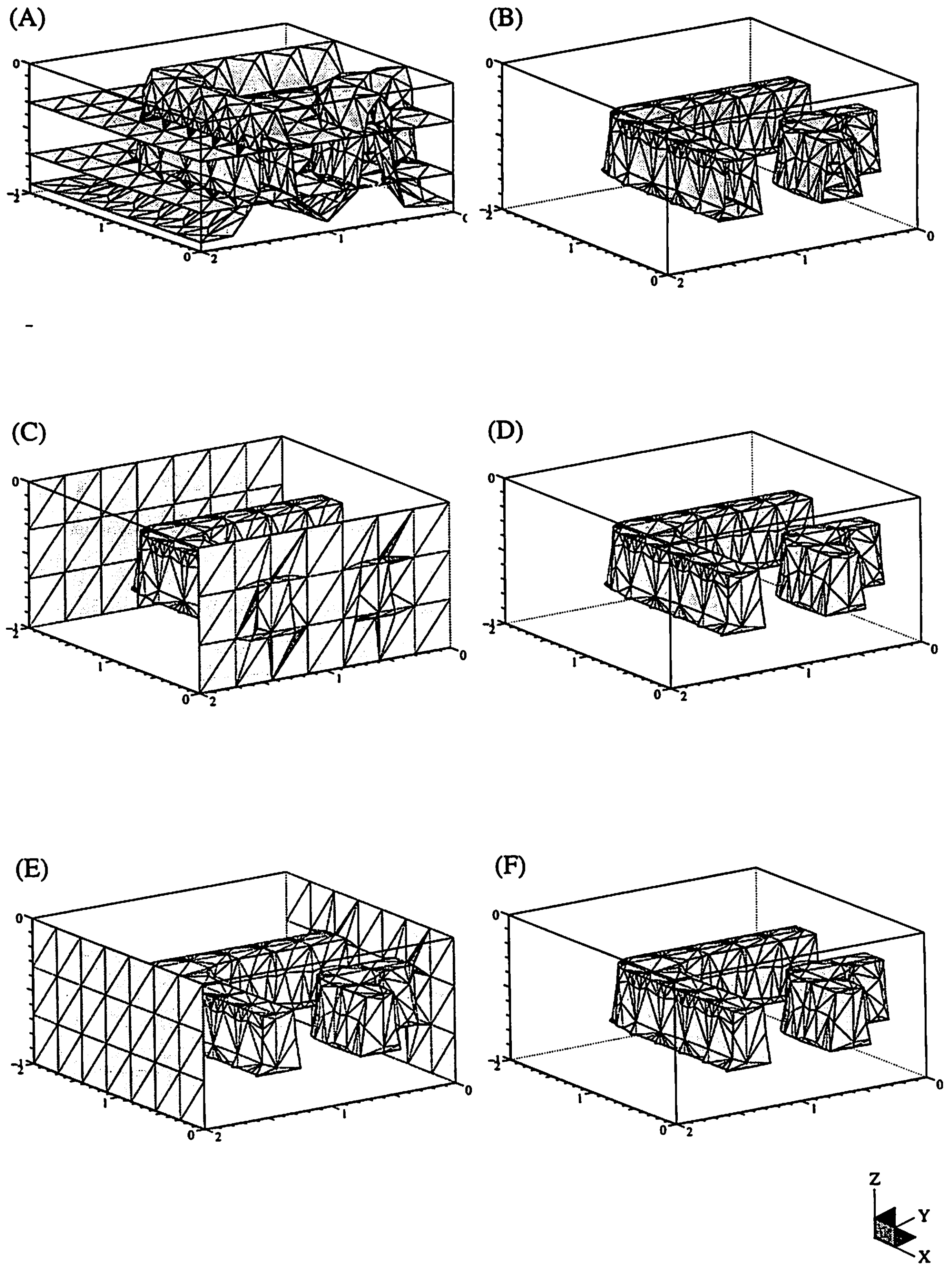


Figure 47 Intermediate stages in the intersection loop of the extraction procedure.

Because the cutting planes are coplanar with the simulation boundaries and triangles that intersect along the edge of another triangle are not considered true intersections, the nodes along the surface edge of the non-cutting plane surface are slightly pushed outside the simulation boundaries. This has two benefits; it permits true triangle intersections, and it facilitates the selection of a triangle for keeping or removal prior to delooping (any triangle positioned outside of the simulation boundary suffices).

7.2.4 Separation of a Discontinuous surface into Individual Surfaces

At the point in the extraction procedure depicted in Fig. 47 F, a single surface containing multiple disconnected groupings remains (Fig. 47 F shows two groups). The linked list of triangles, segments, and nodes need to be separated into individual surfaces representing each grouping as a separate surface. This objective is depicted in Fig. 48.

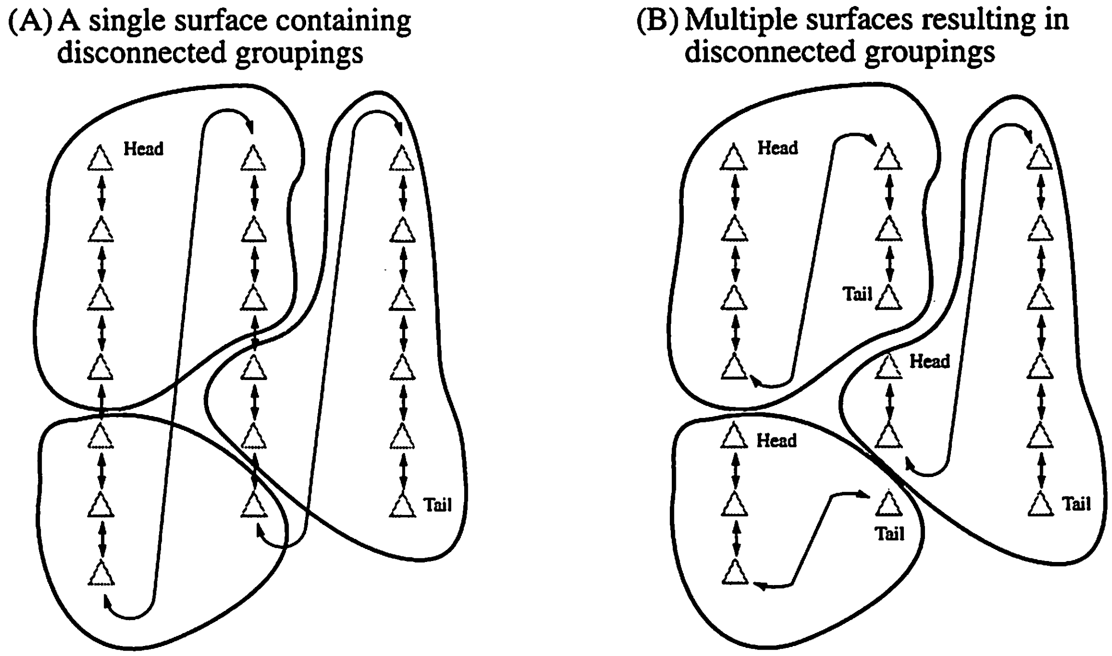

(A) A single surface containing disconnected groupings
Head
Tail
(B) Multiple surfaces resulting in disconnected groupings
Head
Tail
Head
Head
Tail
Tail

Figure 48 A linked list representation of a single surface containing multiple groupings (3) that are separated into multiple individual surfaces.

To determine the number of groupings in a surface and decompose it into multiple surfaces, the following pseudo code procedure is followed:

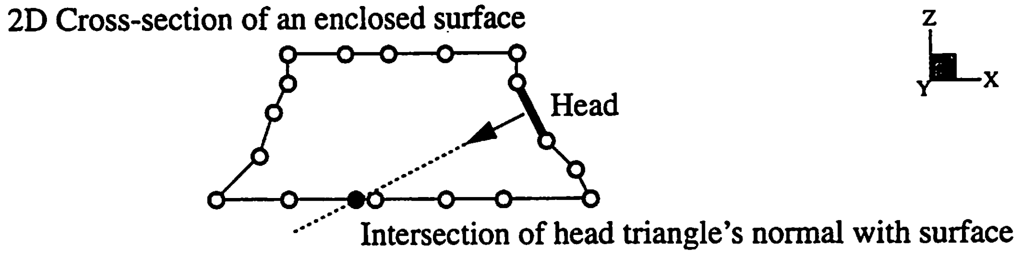
```
reset flags within the triangle, segment, and node list to 0

initialize a group counter to 0

while (a triangle exists whose flag is set to zero)
{
   increment the group counter

   flag the triangle with the current value of the group counter

   perform a breadth-first search marking adjacent triangles
   with the same value of the group counter
}

flag all the segments connected to the triangles with the same
value as the their corresponding triangle's flag

flag all the nodes connected to the segments with the same value
as the their corresponding segment's flag

initialize "group counter" number of new surfaces

for each surface element (triangles, segments, and nodes) in
the flagged surface
{
   delete the element from the original surface and append it to
   a new surface identified by the flag of the surface element
}

destroy the original surface which should now be empty
```

7.2.5 Orienting Triangles of an Enclosed Surface

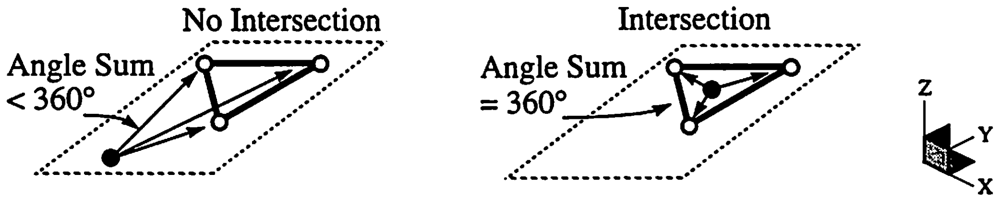
In pursuit of extracting enclosed surfaces from a single surface, many operations have taken place including the creation of cutting planes, surface intersections, thin triangle removal, and regrouping. Unfortunately, consistency in the direction of the triangles' outward normals are not maintained by all of these operations. Thus the final step of reorienting the triangles can be regarded as a clean-up step.

Figure 49 shows an illumination plot where triangles are individually painted with a grayscale value proportional to the angle between the triangle's outward normal direction and direction to the illumination source. If the triangle's outward normal direction opposes the direction toward the illumination plot, the triangle is painted black. As seen in Fig. 49 A, many triangles alternate in their shading intensity. This indicates that many adjacent triangles do not share a consistent outward normal direction. In Fig. 49 B, this problem has been corrected. To correct triangles for consistency in the direction of their outward facing normals, the following illustrated steps are followed:

1 Examine the outward normal of the enclosed surface's head triangle by extending a directed line originating at the center of the head triangle.

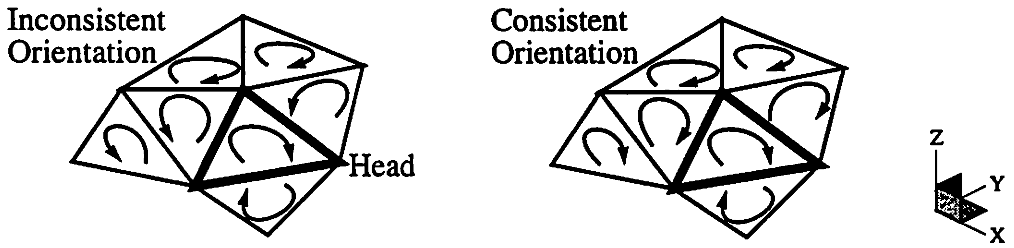

2D Cross-section of an enclosed surface
Head
Intersection of head triangle's normal with surface
Z
Y
X

2 Check every triangle in the surface for an intersection.



3 If an odd number of intersections are found, reorient the head triangle, otherwise do nothing.

4 In a breadth-first search manner, orient the triangles adjacent to the head triangle to have consistent outward pointing normals.



Note the above procedure can be used to orient triangles of a non-enclosed surface (has at least two surface edge segments) by replacing steps 1-3 with a decision about the proper direction for the head triangle's normal direction. Then step 4 is repeated.

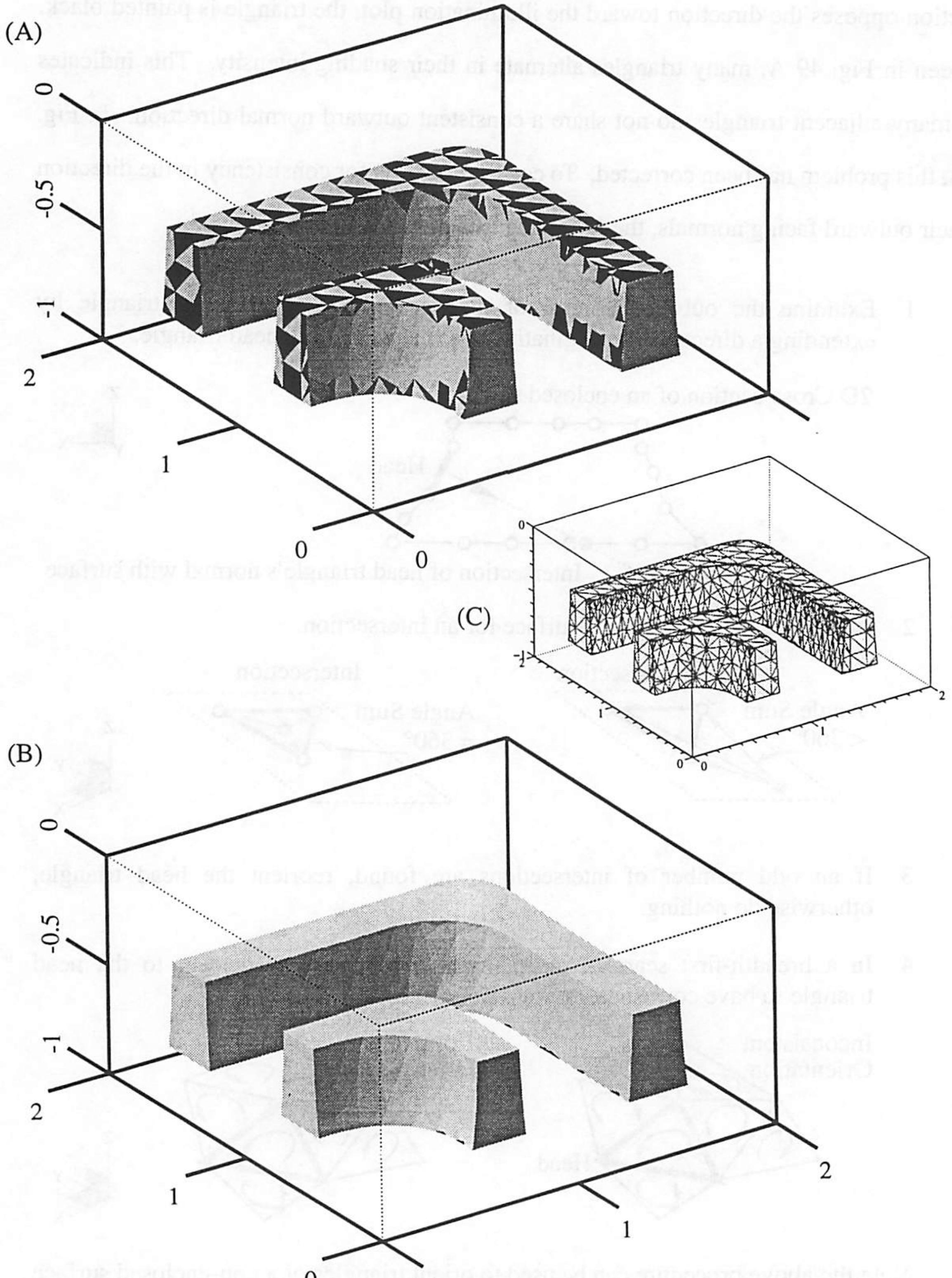


Figure 49 Two enclosed surfaces containing disoriented triangles (A), and the result after being corrected for orientation (B). Note these elbows were simulated with a finer mesh (C) than those presented earlier in this chapter.

7.3 Improving the Efficiency of Solid Extraction

Although the solid-extraction utility has been designed to handle non-planar cutting planes, future use may show that planar cutting planes like those in the elbow example are most common. If this is the case, efficiency in the spatial partitioning and intersection steps can be improved. During the extraction step depicted in Fig. 47 A, triangles in the non-planar simulation surface whose nodes do not straddle either cutting plane need not be inserted into the octtree since they are not candidates for intersections. Likewise, only triangles bordering the simulation boundaries are intersection candidates during the extraction steps depicted in Fig. 47 C and 47 E. These enhancements should have a noticeable run-time improvement since the number of triangles inserted into the octtree is reduced. Consequently this also reduces the number of triangle intersection tests.

Although less significant, another source for improving run-time speed is in the triangle reorientation step. As stated in Sec. 7.2.5 and illustrated in Fig. 49 A, triangle orientation can be adversely affected during the solid extraction procedures. To correct the inconsistent orientation and give the enclosing surface an outward pointing normal, a four step procedure with a running time of *O(2n)* was given. In step 2 of the procedure, *every* triangle is tested for an intersection with the directed normal vector of the head triangle. Because the current implementation has already spatially partitioned *all* triangles into an octtree, intersections with the head triangle's normal vector need only be checked with triangles in the octtree cells pierced by the vector. This would reduce the running time for reorientation by approximately 50%.

Although not necessarily an efficiency improvement, it would be nice to eliminate the need for reorientation by moving the responsibility for initialization and maintenance of the triangles' orientation to the initialization and re-triangulation operations. Inspection of Fig. 49 A shows that the cutting planes were not oriented before they were used.

Provided the cutting planes are planar, as depicted throughout this chapter, the direction of their outward normals could have been properly initialized without error. Then subdivided triangles introduced during the surface intersection steps would have to be inserted while maintaining consistent orientation with its neighbors. With the exception of the triangle flipping operations, the thin triangle removal routines would not have to be modified since the orientation of triangles surrounding a removed triangle are not altered. With these changes, the presence of disoriented triangles can be eliminated.

Chapter 8:

Interconnect Analysis

8.1 An Application of the Surface Modeling Utilities - Interconnect Analysis

The driving force that inspired the development of most of the surface modeling utilities in this thesis was the need for computing parasitic interconnect parameters on rigorously simulated three dimensional surfaces. FASTCAP and FASTHENRY are both programs developed by Professor Jacob White at M.I.T. [13,11] that can respectively compute capacitance and inductance matrices on 3D topologies. In addition, many device simulators [21,17,18,3,1] have been (are being) developed that can rigorously simulate 3D topology results from integrated circuit manufacturing processes. Having both of these tools available, it is a natural desire to use them for prediction of parasitic parameters such as mutual capacitance on the simulated topologies. To accomplish this task however, it is necessary to first extract solid geometries from the surface simulators as presented in Chapter 7.

In this chapter, a comparison is made between SAMPLE-3D's rigorously simulated polysilicon elbows presented in Sec. 7.1 (Introduction to the Solid Elbow Extraction Problem) and a pair of first order approximation elbows generated using CUBEGEN, one of a few primitive solid model generators accompanying the FASTCAP package. We will see from the polysilicon elbow example that parasitic capacitance calculations based on topologies that deviate from a realistic profile can produce significant overestimates.

8.2 Generating an Elbow Model using CUBEGEN

For the sake of comparing capacitance results between different methods of estimating the geometry of etched polysilicon elbows, CUBEGEN is used to create a solid model and compare the resulting capacitance analysis with that of the SAMPLE-3D results. To model the elbows using CUBEGEN, one could approximate the elbow geometry as a direct extrusion of the polysilicon through the mask, or for more accuracy one can use information from the aerial image generated during the SPLAT [5] simulation shown in Fig. 50. Using the 30% percent threshold model from the aerial image to closer

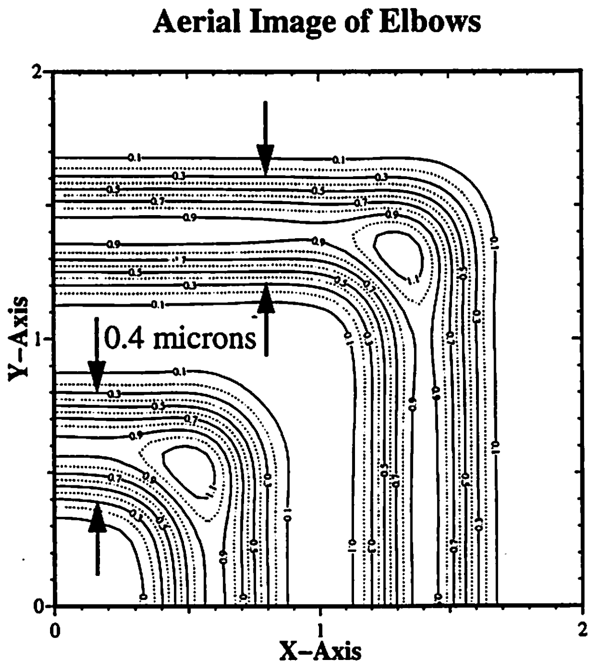


Figure 50 Aerial image through the elbow mask using SPLAT.

approximate the actual feature widths of the elbows, a 0.4 micron feature width along the longitudinal portions is measured. Coincidently, the width of the mask features are also 0.4 microns. Thus the direct extrusion approximation and the 30% threshold model yield the same width approximation along the longitudinal portion of the polylines away from the elbow corner. In Fig. 51, the rigorously simulated and extruded polysilicon elbows are shown.

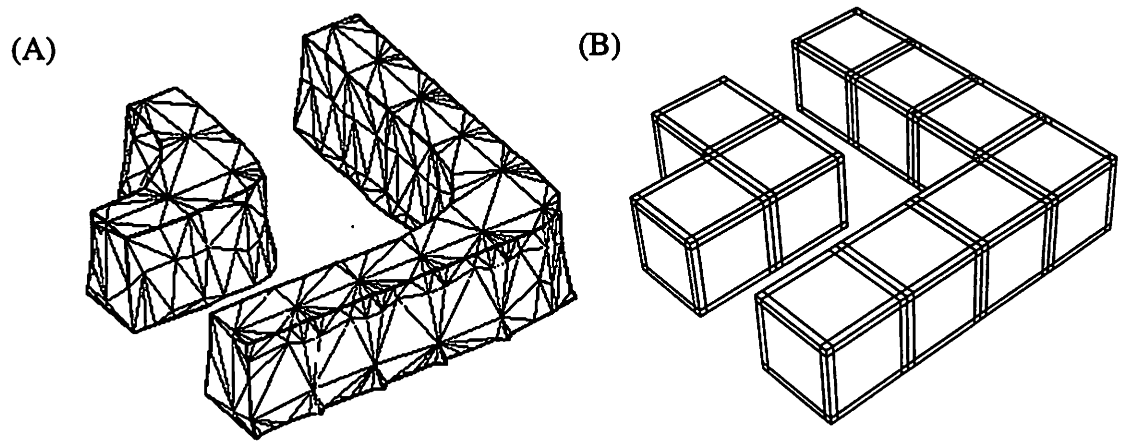


Figure 51 3D representations of polysilicon elbows generated from a rigorous topography simulator (A) and a first order mask extrusion approximation (B).

8.3 Comparison of Capacitance Analyses

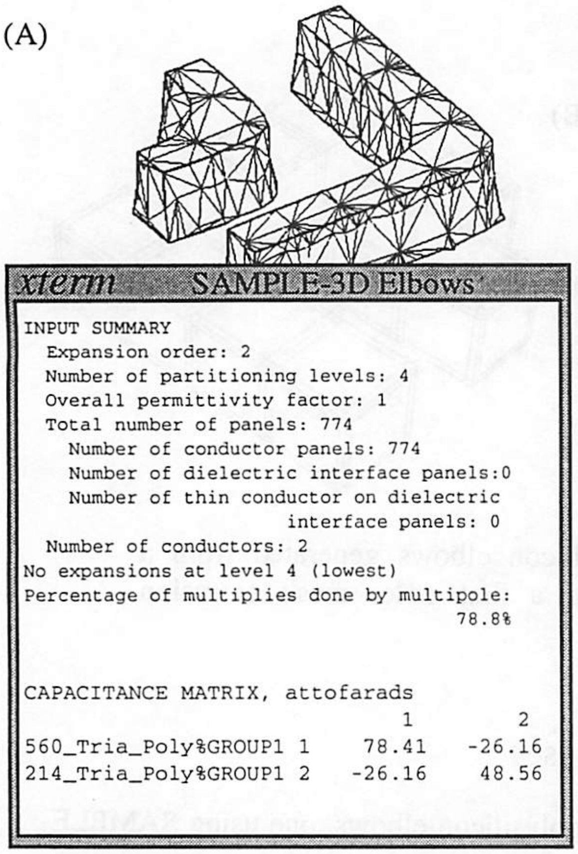
Having created two models of the solid polysilicon elbows, one using SAMPLE-3D and one using CUBEGEN, input files are created for FASTCAP and the capacitance analysis is done. Figure 52 shows the final few lines of FASTCAP output from the analysis of both models.

Of interest from the FASTCAP results is the capacitance matrix which shows the self capacitance of each solid with infinity and the mutual capacitance between the elbows. Table 1 lists the difference between the results as a percent difference with respect to the SAMPLE-3D elbows. Notice the significant 44% difference in mutual capacitance.

Table 1 Percent Difference of capacitance matrices with respect to the SAMPLE-3D matrix.

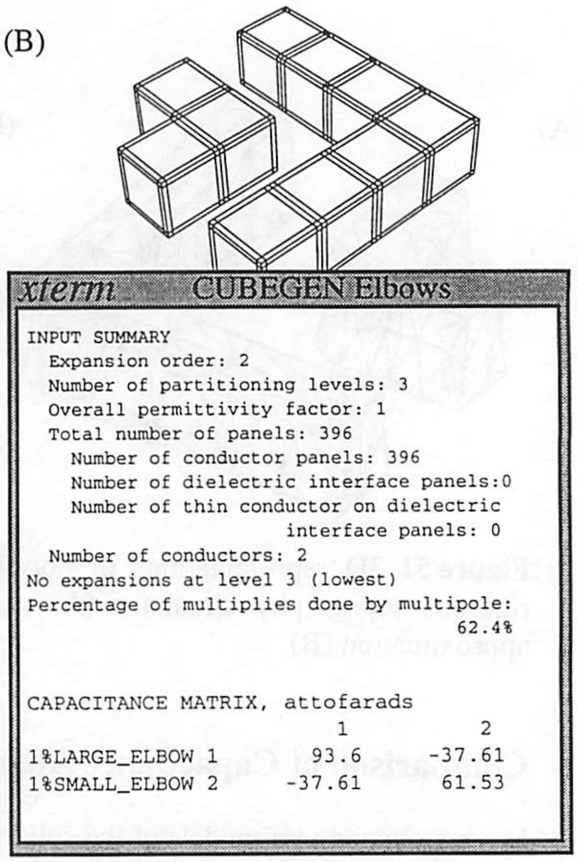
	Large Elbow 1	Small Elbow 2
Large Elbow 1	+19%	+44%
Small Elbow 2	+44%	+27%

(A)



```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 4
  Overall permittivity factor: 1
  Total number of panels: 774
    Number of conductor panels: 774
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                      interface panels: 0
  Number of conductors: 2
No expansions at level 4 (lowest)
Percentage of multiplies done by multipole:
                                        78.8%


CAPACITANCE MATRIX, attofarads
                                  1          2
560_Tria_Poly%GROUP1 1       78.41     -26.16
214_Tria_Poly%GROUP1 2      -26.16      48.56
```

(B)



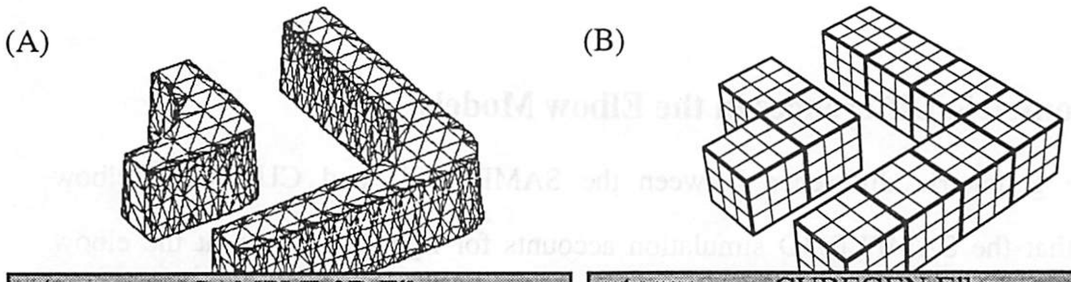
```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 3
  Overall permittivity factor: 1
  Total number of panels: 396
    Number of conductor panels: 396
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                      interface panels: 0
  Number of conductors: 2
No expansions at level 3 (lowest)
Percentage of multiplies done by multipole:
                                        62.4%


CAPACITANCE MATRIX, attofarads
                          1            2
1%LARGE_ELBOW 1        93.6       -37.61
1%SMALL_ELBOW 2      -37.61        61.53
```

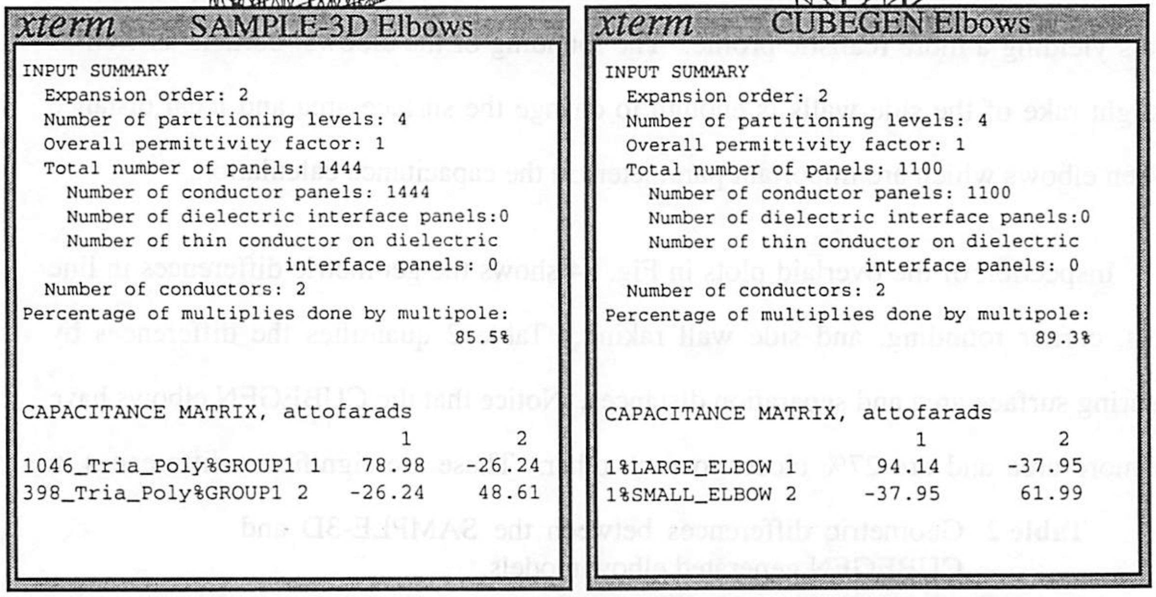
Figure 52 Comparison of capacitance analysis results using FASTCAP.

8.3.1 Accounting for the Difference in Mutual Capacitance

Because the difference in capacitances appears so large, one may suggest it is due to numerical issues caused by the coarse tiling of the solids. This is a valid issue for the following reason. FASTCAP assumes that the charges are uniformly distributed across a panel, and when this assumption is inadequate, inaccurate results occur. In fact, it is known that charges across the elbow structures will accumulate along the structure's edges. The authors of FASTCAP realized this and consequently built their primitive solid modelers to automatically discretize the panels along the surface edges. By making this discretization, the uniformity of charge distribution across individual panels is improved. The discretization is apparent in Fig. 51 B and 52 B.

With the concept of uniform charge distribution in mind, it would make sense that finer triangulations and finer panels would lead to more accurate results. Of course the accurate results will be at the expense of cpu time. To test this hypothesis and establish the adequacy of the capacitance results in Fig. 52, both the SAMPLE-3D and CUBEGEN elbows were regenerated with finer panels (1444 triangles and 1100 panels respectively) and FASTCAP was re-executed. The results are shown in Fig. 52. The resulting





```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 4
  Overall permittivity factor: 1
  Total number of panels: 1444
    Number of conductor panels: 1444
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                     interface panels: 0
  Number of conductors: 2
Percentage of multiplies done by multipole:
                                       85.5%


CAPACITANCE MATRIX, attofarads
                                1          2
1046_Tria_Poly%GROUP1 1     78.98     -26.24
398_Tria_Poly%GROUP1 2     -26.24      48.61
```

```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 4
  Overall permittivity factor: 1
  Total number of panels: 1100
    Number of conductor panels: 1100
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                     interface panels: 0
  Number of conductors: 2
Percentage of multiplies done by multipole:
                                       89.3%


CAPACITANCE MATRIX, attofarads
                                1          2
1%LARGE_ELBOW 1             94.14     -37.95
1%SMALL_ELBOW 2            -37.95      61.99
```

Figure 53 Comparison of capacitance analysis results using FASTCAP with finer panels than analyzed in Fig. 52.

capacitance analysis was within 1% of the results shown in Fig. 52. Thus the adequacy of the previous results are established.

One may also question the effect of removing thin triangles on the capacitance results. Again a capacitance analysis was conducted on the elbow structures before and after thin triangle removal only to find that the FASTCAP simulation was three times faster after thin triangles were removed (likely due to the reduced triangle count), and the capacitance matrix results were within 1.5% of each other.

Having disputed the reasons above as major contributors to the difference in the models' mutual capacitances, the 44% capacitance difference reported in Table 1 is most likely due to appreciable differences in geometry, not the coarseness or regularity of the tiling.

8.3.2 Geometric Differences in the Elbow Models

One geometric difference between the SAMPLE-3D and CUBEGEN elbow models is that the SAMPLE-3D simulation accounts for light diffraction at the elbow corners yielding a more realistic profile. The rounding of the elbows' corners as well as the slight rake of the side walls is enough to change the surface area and local distance between elbows which are important parameters in the capacitance calculation.

Inspection of the overlaid plots in Fig. 54 shows the geometric differences in line widths, corner rounding, and side wall raking. Table 2 quantifies the differences by measuring surface area and separation distances. Notice that the CUBEGEN elbows have 20% more area and are 27% closer to each other. These are significant differences in

Table 2 Geometric differences between the SAMPLE-3D and CUBEGEN generated elbow models.

	SAMPLE-3D	CUBEGEN	% Difference
Surface Area (μm^2)	1.77 / 4.09	2.24 / 4.80	+20%
Separation Distance (μm)	0.55	0.40	-27%

geometry and will undoubtedly lead to significant differences in mutual capacitance.

8.3.3 Incorporating a Bias in the CUBEGEN Model

Thus far we have established that a CUBEGEN model generated as an extrusion through the mask and a CUBEGEN model generated with 30% threshold information

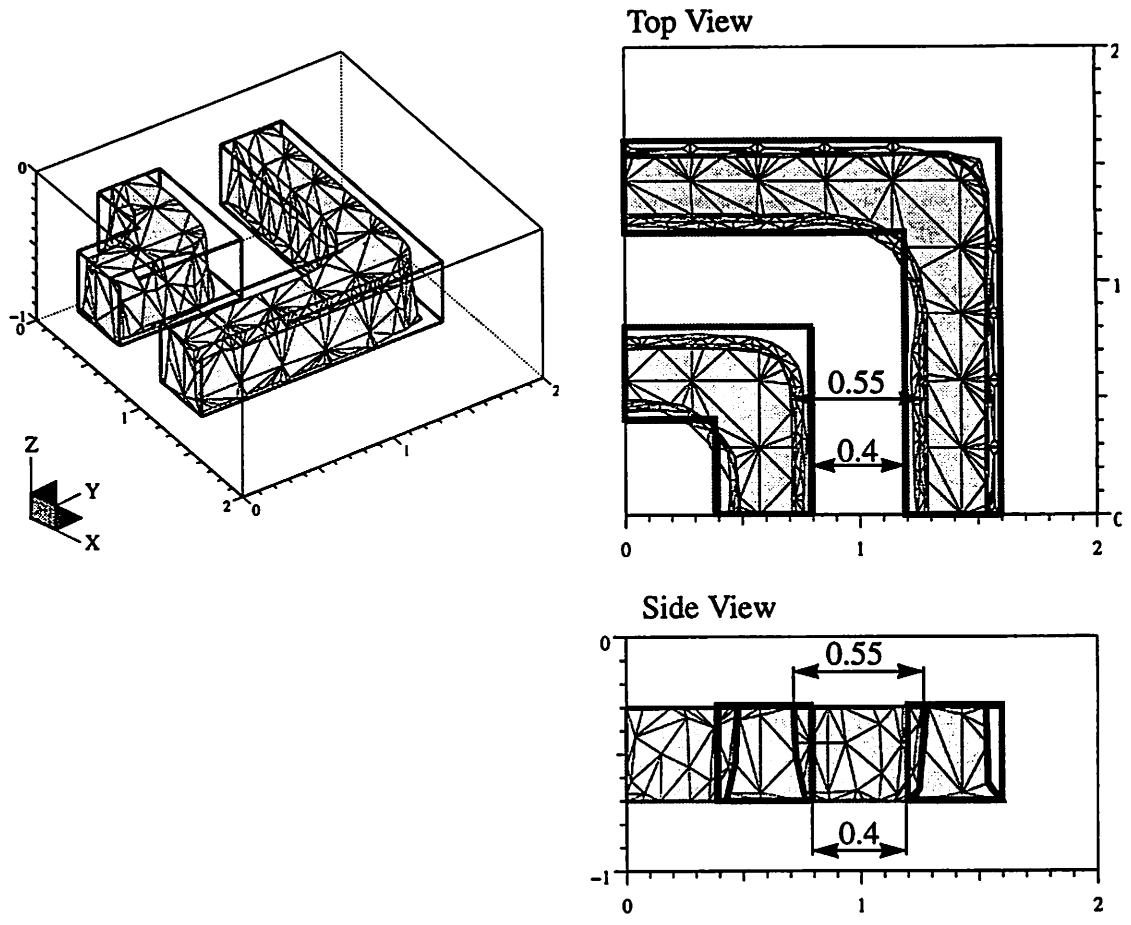


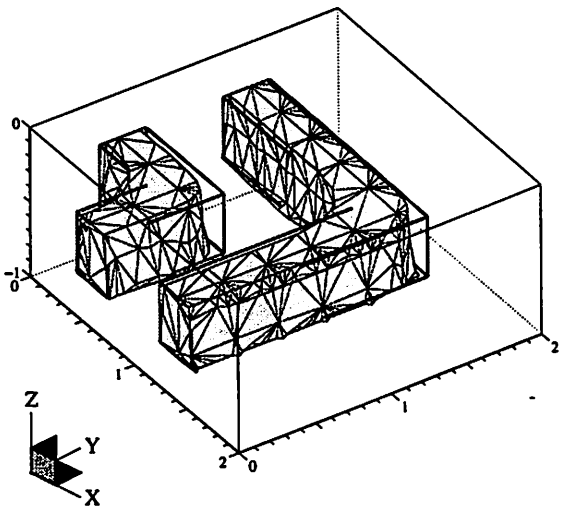
Figure 54 Overlaid plots showing geometric differences in the elbow models. Units are in microns.

from the aerial image will both lead to capacitance analysis results that are 44% greater than capacitance results from a more realistic model. At this point, a circuit designer may ask, "If I incorporate enough bias in the CUBEGEN model to closely approximate the actual surface area and separation distance, how realistic will the capacitance be?" This question is quite relevant since this is the approach circuit designers take when predicting capacitances. The difficulty in this method is knowing how much bias to introduce. Despite this difficulty, another CUBEGEN model is generated to test the circuit designer's question.

To generate a CUBEGEN model that closely approximates the surface area and separation distance of the SAMPLE-3D model, a 0.5 micron mask bias is required. The small geometric differences are given in Table 2., and overlaid plots are given in Fig. 54.

Table 3 Geometry tabulation of the SAMPLE-3D and CUBEGEN elbows generated with a 0.5 micron mask bias to reduce geometric differences.

	SAMPLE-3D	CUBEGEN	% Difference
Surface Area (μm^2)	1.77 / 4.09	1.83 / 4.07	+1%
Separation Distance (μm)	0.55	0.50	-9%



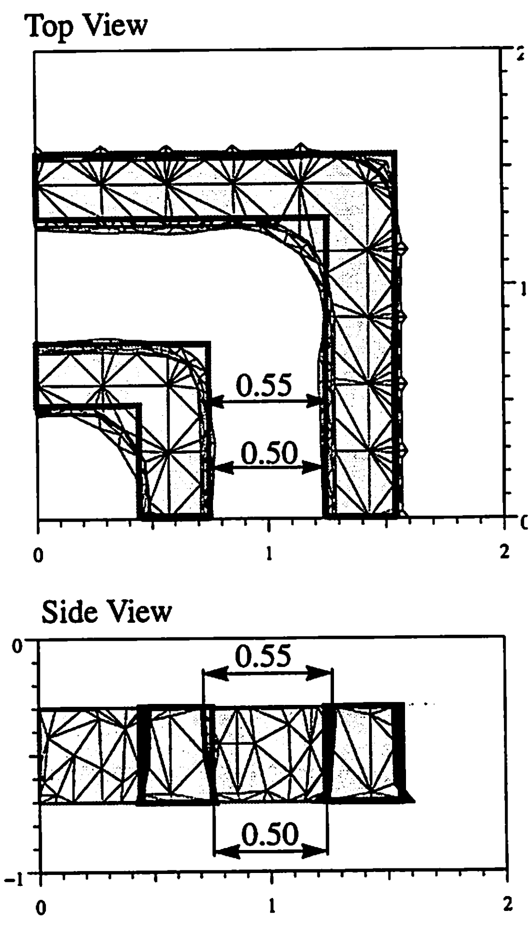
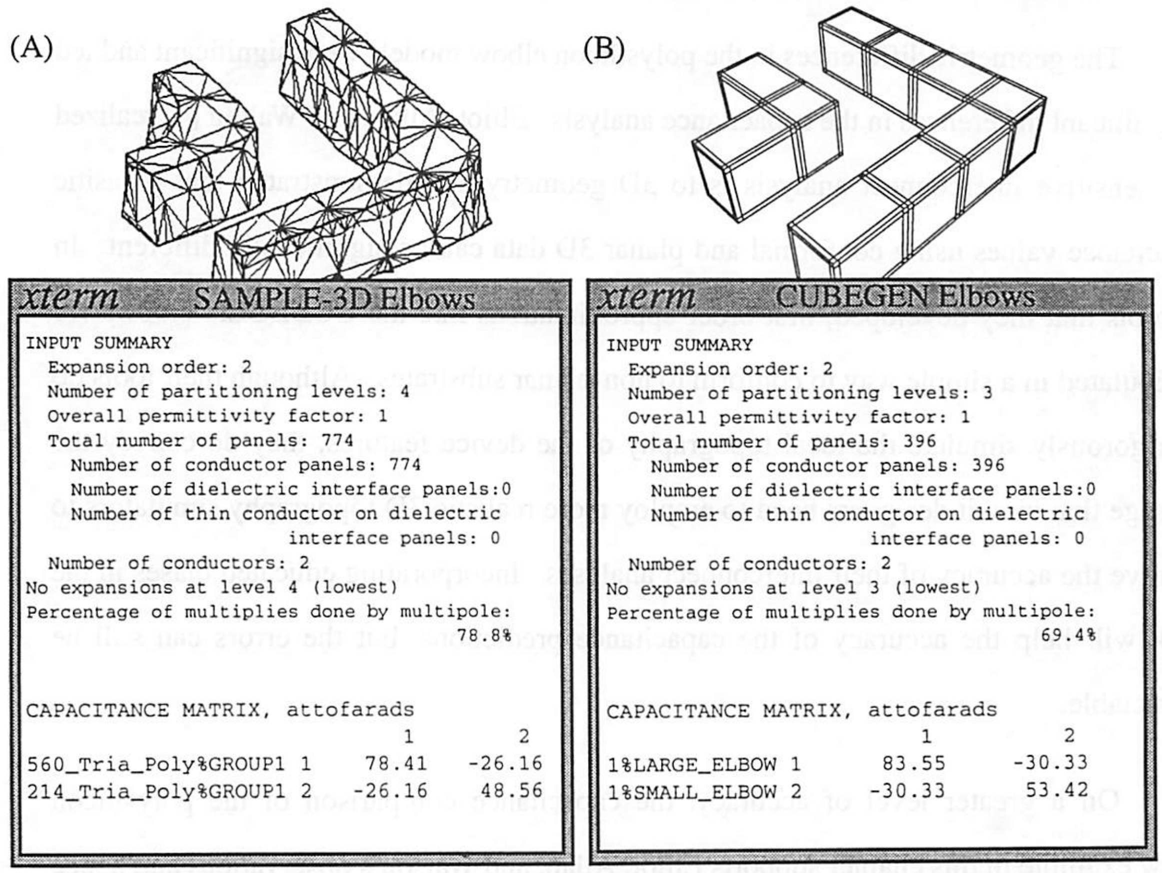


Figure 55 Overlaid plots showing small geometric differences between the SAMPLE-3D model and CUBEGEN model generated with a 0.5 micron mask bias. Units are in microns.

The capacitance analysis results of this test are shown in Fig. 56, and Table 1 contains the percent differences in capacitance. Notice that the mutual capacitance difference has



```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 4
  Overall permittivity factor: 1
  Total number of panels: 774
    Number of conductor panels: 774
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                     interface panels: 0
  Number of conductors: 2
No expansions at level 4 (lowest)
Percentage of multiplies done by multipole:
                                      78.8%


CAPACITANCE MATRIX, attofarads
                                 1          2

560_Tria_Poly%GROUP1 1       78.41     -26.16
214_Tria_Poly%GROUP1 2      -26.16      48.56
```

```
INPUT SUMMARY
  Expansion order: 2
  Number of partitioning levels: 3
  Overall permittivity factor: 1
  Total number of panels: 396
    Number of conductor panels: 396
    Number of dielectric interface panels:0
    Number of thin conductor on dielectric
                     interface panels: 0
  Number of conductors: 2
No expansions at level 3 (lowest)
Percentage of multiplies done by multipole:
                                      69.4%


CAPACITANCE MATRIX, attofarads
                              1          2

1%LARGE_ELBOW 1           83.55     -30.33
1%SMALL_ELBOW 2          -30.33      53.42
```

Figure 56 FASTCAP analysis comparison between the SAMPLE-3D elbows model and the CUBEGEN model incorporating a 0.5 micron mask bias to reduce the geometric differences.

reduced to 16%. This answers the circuit designers question.

Table 4 Percent Difference of capacitance matrices with respect to the SAMPLE-3D matrix.

	Large Elbow 1	Small Elbow 2
Large Elbow 1	+7%	+16%
Small Elbow 2	+16%	+10%

Provided the circuit designer can exactly predict the required mask bias, the capacitance result will be closer to that of a more realistic geometry model, but it is still a considerable over-estimate.

8.4 A Final Word on Interconnect Analysis

The geometric differences in the polysilicon elbow models were significant and led to significant differences in the capacitance analysis. Elliot, Allan, and Walton [4] realized how sensitive interconnect analysis is to 3D geometry and demonstrated that parasitic capacitance values using conformal and planar 3D data can be significantly different. In the tools that they developed, first order approximations like the CUBEGEN elbows are manipulated in a simple way to conform to non-planar substrates. Although their tools do not rigorously simulate the local topography of the device features, they do convey the message that circuit designers need to employ more realistic 3D topography simulators to improve the accuracy of their interconnect analyses. Incorporating educated biases in the mask will help the accuracy of the capacitance predictions, but the errors can still be appreciable.

On a greater level of accuracy, the capacitance comparison of the polysilicon elbow example in this chapter supports Elliot, Allan, and Watson's observations and hence the importance of the relationship between manufacturing and chip performance.

Chapter 9:

Summary

In this thesis, a number of surface modeling utilities have been presented for use in correcting triangle irregularities that develop during surface advancement by process simulation tools of integrated circuits. Although these utilities are intended to support TCAD tools, they were developed in general enough context that they could be used to address a broader audience of scientific triangulation problems.

Of notable interest in this thesis was the introduction of a new means by which integrated circuit designers can perform interconnect analyses using results from rigorously simulated 3D topologies. An example comparing the capacitance matrices from a rigorously simulated surface employing the new solid extraction utility was compared to a commonly used first-order method for device approximation. Interestingly, the first-order model lead to a considerable (44%) over-estimate of the capacitance matrix calculated by FASTCAP. The significant difference in capacitance was largely attributable to geometric differences such as surface area and separation distance between the conductors. This result reestablishes the need for more accurate 3D semiconductor device simulators and the supporting surface modeling utilities needed to improve the robustness of the simulators.

While 3D semiconductor topography simulators are still somewhat slow and require extensive memory for entire wafer layouts, future projects to simulate portions of a layout such as lines, elbows, and T-shaped conductors followed by solid extraction and capacitance analysis are feasible. Results from these experiments can be tabulated with corresponding correction factors. In this way, circuit designers can apply these factors to crude models while obtaining more realistic capacitance analyses in a fast time-frame.

This vision that has already prompted Alex Quezada's tabulation of capacitance correction factors for a set of isotropic etched T-shaped conductors [16].

Regarding the state of the software used to implement the surface modeling utilities, there are many ways in which efficiency can be improved. At the end of many chapters, a brief discussion was included about how and where significant improvements can be made. Before continuing development of these utilities, the issues presented in these sections should be considered.

To date, at least five different people have added functionality to the software (NETCH). Because few have been members of the Berkeley TCAD Research Group at the same time, little communication to preserve a consistent software methodology has occurred. Consequently, the software organization and implementation of NETCH suffers from a lack of structured software engineering practices. A number of modules are sparsely commented and few header files are used to declare the type and usage of public functions. Before continuing development of NETCH, I would suggest that a serious effort be made to restructure, or rewrite some of the current modules. In this way, software maintenance and extensibility can be made much easier. This project may also entail the development of a new framework in which the algorithms currently housed within NETCH should reside. Refer to Robert Wang's thesis [24] for a detailed discussion of this framework. This restructuring project is well suited for a computer science graduate student interested in software engineering of scientific applications.

References

[1] E. Bär, J. Lorenz, "3-D Simulation of LPCVD using Segment Based Topography Discretization", To appear in *IEEE Transactions on Semiconductor Manufacturing*, submitted May 3, 1995.

[2] M. Bern, D. Eppstein, "Mesh Generation and Optimal Triangulation", chapter 2, Edited by Ding-Zhu Du, Frank Hwang, Computing in Euclidean Geometry, World Scientific, vol. 1, 1992.

[3] T.S.Cale, T.H. Gandy, M.K. Jain, M. Ramaswami, and G.B. Raupp, "A General Model for PVD Deposition," *Proceedings Eight International VLSI Multilevel Interconnection Conference*, pp. 350-352, Santa Clara, June 11-12, 1991.

[4] J.P. Elliot, G.A. Allan, A.J. Walton, "The Automatic Generation of Conformal 3D Data for Interconnect Capacitance Simulation," *1995 Proceedings Twelfth International VLSI Multilevel Interconnection Conference*, June 27-29, 1995, Santa Clara, CA., 95 ISMIC - 104, pp. 664-666.

[5] P. Flanner, K. Toh, D. Newmark, D. Lee, *SPLAT v5.0 Users' Guide*, University of California, Berkeley, Jan. 19, 1995.

[6] J.J. Helmsen, E.W. Scheckler, A.R. Neureuther, C.H. Séquin, "An Efficient Loop Detection and Removal Algorithm for 3D Surface-based Lithography Simulation," *Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV*, pp. 3-8, June 1992.

[7] J.J. Helmsen, *An Efficient Loop Detection and Removal Algorithm for 3D Surface-Based Lithography Simulation*, Memo. No. UCB/ERL M92/125, M.S. Thesis, University of California, Berkeley. Nov. 1992.

[8] J.J. Helmsen, *A Comparison of Three Dimensional Photolithography Simulators*, Memo. No. UCB/ERL M95/25, Ph.D. Thesis, University of California, Berkeley. April. 1995.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization", *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH '93)*, Anaheim, California, August 1993.

[10] R.E. Jewett, P.I. Hagouel, A.R. Neureuther, and T. Van Duzer, "Line-Profile Resist Development Simulation Techniques", *Polymer Engineering and Science*, vol. 17, no. 6, pp. 381-384, June 1977.

[11] M. Kamon, M.J. Tsuk, and J. White, "FastHenry, A Multipole-Accelerated 3-D Inductance Extraction Program," *Proceedings of the ACM/IEEE Design Automation Conference*, Dallas, June 1993.

[12] J.P.McVittie, J.C.Rey, A.J.Bariya, M.M.IslamRaja, M.M, and others, "SPEEDIE: a profile simulator for etching deposition." Conference: (Advanced Techniques for Integrated Circuit Processing, Santa Clara, CA, USA, 1-5 Oct. 1990). *Proceedings of the SPIE - The International Society for Optical Engineering*, vol.1392, pp. 126-38, 1991

[13] .K. Nabors, and J. White, "FastCap: A Multipole Accelerated 3-D Capacitance Extraction Program," *IEEE Transactions On Computer-Aided Design*, vol. 10, no. 11, pp. 1447-1459 Nov. 1991

[14] W.G.Oldham, S.N.Nandgaonkar, A.R.Neureuther, and M.M.O'Toole, "A General Simulation for VLSI Lithography and Etching Processes: Part I - Application to Projection Lithography," *IEEE Transactions on Electron Devices*, vol. ED-26, no. 4, pp. 717-722, April 1979.

[15] W.G.Oldham, A.R.Neureuther, C. Sung, J.L.Reynolds, and S.N.Nandgaonkar, "A General Simulation for VLSI Lithography and Etching Processes: Part II - Application to Deposition and Etching," *IEEE Transactions on Electron Devices*, vol. ED-27, no. 8, pp. 1455-1459, August 1980.

[16] A. Quezada, "Capacitance Analysis Using SAMPLE-3D Simulation," University of California, Berkeley, EECS Dept., SUPERB Program 1995, Mentor A.R.Neureuther, Aug. 9, 1995.

[17] E.W. Scheckler, A.R. Neureuther, "Models and Algorithms for Three-Dimensional Topography Simulation with SAMPLE-3D", *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 2, pp. 219-230 Feb. 1994.

[18] E.W.Scheckler, *Algorithms for Three-Dimensional Simulation of Etching and Deposition Processes in Integrated Circuit Fabrication*, Memo. No. UCB/ERL

M91/99, Ph.D. Dissertation, University of California, Berkeley. Nov. 1991.

[19] W.J. Schroeder, J.A. Zarge, W.E. Lorenson, "Decimation of Triangle Meshes", *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26(2), pp. 65-70, July 1992

[20] S.Tazawa, F.A. Leon, G.D. Anderson, T. Abe, K. Saito, A. Yoshii, D.L. Sharfetter, "3-D Topography Simulation of Via Holes using Generalized Solid Modeling," *International Electron Devices Meeting 1992. Technical Digest*, pp. 173-176, Dec. 1992.

[21] K.K.H. Toh, A.R. Neureuther, E.W. Scheckler, "Algorithms for Simulation of Three-Dimensional Etching", *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 5, pp. 616-624 May 1994.

[22] K.K.H. Toh, *Algorithms for Three-Dimensional Simulation of Photoresist Development*, Memo. No. UCB/ERL M90/123, Ph.D. Dissertation, University of California, Berkeley. Dec. 1990.

[23] G. Turk, "Re-Tiling Polygonal Surfaces", *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26(2), pp. 55-64, July 1992.

[24] R.H. Wang, *Centralizing Geometry Services for Three-Dimensional Integrated Circuits Topography Simulation*, To be submitted, Ph.D. Dissertation, University of California, Berkeley. Nov. 1995.

Appendix - User's Guide

The algorithms developed in this thesis have been packaged within NETCH release version 2.1. The following list contains an extension to the SIMULATOR COMMANDS section of the man page for NETCH.

SIMULATOR COMMANDS (additional):

unitization commands:

save_surface *surface_id sim_parameters node_direction filename*;

This command is used to save the geometry and some optional information obtained from running a NETCH simulation. This is useful for saving intermediate states of the surface so that it can be loaded back into NETCH's data structures at a later time for continued simulation or operations. The parameters include:

surface_id=-1 will process all surfaces, otherwise only the indicated is processed. NETCH tags each surface with an integer starting with 0, then 1, 2, etc.

sim_parameters=1 saves the necessary simulation parameters used during a facet etching or deposition process, otherwise use 0.

node_direction=1 saves the node advancement information for ray etching, otherwise use 0.

load_surface *filename*;

This command loads the geometry and possibly some simulation parameters of a surface that was saved using the *save_surface* command. This is useful for continuing a long simulation process from an intermediate state. If the file was generated by some other means, the contents of the file allows any and all simulator commands listed in the man page for NETCH as well as the following geometry commands:

Surface *surfID*;

Node *nodeID x y z*;

Segment *segmID nodeID nodeID*;

Triangle *trialD segmID segmID segmID*;

For the geometry commands to work properly, the Surface command must proceed all the elements belonging to the surface, and all elements ID's referenced in an argument list must have been previously defined.

create_surface_x *x*;

Within the simulation boundaries defined by *sim_region*, initialize a flat surface at *x* with a mesh density set by the *meshdensity* command.

create_surface_y *y*;

Within the simulation boundaries defined by *sim_region*, initialize a flat surface at *y* with a mesh density set by the *meshdensity* command.

create_surface_z *z*;

Within the simulation boundaries defined by *sim_region*, initialize a flat surface at *z* with a mesh density set by the *meshdensity* command. Note that this command is not required to begin a simulation. A similar operation is automatically executed when the program first executes.

surface triangulation commands:

calc_surface_area *[surface_id]*;

For the indicated surface, compute the area of all triangles within the surface. Setting *surface_id*=-1 will perform the operation on all defined surfaces.

calc_coplanarity *[surface_id]*;

For the indicated surface, calculate a normalized coplanarity vector at each node. As the magnitude of the coplanarity vectors approach 1.0, the surface is perfectly flat at the node. As the coplanarity vector approaches 0.0, the surface comes to a cusp at the node. See the *plot3Dmtv_vectors* command for viewing the result of these calculations. Setting *surface_id*=-1 will process all defined surfaces.

ridge_angle *degrees*;

Specifies the dihedral angle between adjacent triangles above which the adjoining segment is considered to be of topological significance. The *ridge_angle* parameter is used to make topological decisions in operations like *remove_thin_trias* and *delaunay_surface*. The default is 45°.

remove_thin_trias *[[surface_id] degrees]*;

Remove all the thin triangles throughout the identified surface without altering any significant topological features. Setting *surface_id*=-1 will perform the operation on all defined surfaces. The optional *degrees* parameter is used to set the *ridge_angle* which is used to identify a topologically significant segment whose topology must be preserved during the removal of its adjacent thin triangle.

delaunay_surface *[[surface_id] degrees]*;

For the identified surface, alter the triangulation of the mesh so it possesses the properties of a 2 $^1/_2$ D Delaunay triangulation. Setting *surface_id*=-1 will process all defined surfaces. The optional *degrees* parameter is used to set the *ridge_angle* which is used to identify a topologically significant segment that should not be flipped in the Delaunay triangularization process.

decimate_surface *[[[surface_id] incremental] calc_coplanarity] copl_threshold]*;

To decimate the indicated surface means to reduce the mesh granularity in regions where the surface curvature is low. The decision to remove triangles is based on the coplanarity calculations made by *calc_coplanarity*. The decimation arguments include:

surface_id=-1 will process all surfaces, otherwise only the indicated is processed. NETCH tags each surface with an integer starting with 0, then 1, 2, etc.

incremental=1 will decimate the surface using an incremental algorithm. When using the incremental algorithm, not all the nodes whose coplanarity vector has a magnitude greater than *copl_threshold* are removed. Afterwards, you may want to *remove_thin_trias* and do a *delaunay_surface* operation. Setting *incremental*=0 will use a fast greedy algorithm that will try to remove all the nodes whose coplanarity vector has a magnitude greater than *copl_threshold*. The default is *incremental*=0.

calc_coplanarity=1 will issue a call to the *calc_coplanarity* operation with *surface_id*. Setting *calc_coplanarity*=0 will not calculate the coplanarity. Be aware that the coplanarity must be calculated before you can successfully decimate the surface. The option not to calculate the coplanarity is useful when you are using the *incremental* algorithm to decimate triangles a little at a time. This allows successive decimations to be based on the initial topography.

copl_threshold is used to inform the algorithm the tolerable amount of normalized surface coplanarity beyond which the surface can be considered flat. See the command description for *calc_coplanarity*. The default is 0.98.

smooth_triangles *[surface_id]*;

Without repositioning any nodes, re-establish a mesh connectivity that reduces the crenulation of the surface. Mathematically, this command minimizes the dihedral angle between adjacent triangles. This command is useful for reestablishing a ridge line along surface areas of high curvature. After issuing this command, it may be desirable to issue a *delaunay_surface* and *remove_thin_trias* command. Setting *surface_id*=-1 will process all defined surfaces.

smooth_nodes *[surface_id]*;

Without changing the mesh connectivity, position nodes at the geometric center of their neighbor nodes. Particular attention is paid to nodes along a ridge line defined by the *ridge_angle* command so as to maintain the topographical features in the surface. After issuing this command, the surface may appear to possess the properties of a Delaunay triangulation. This is not necessarily true. Setting *surface_id*=-1 will process all defined surfaces.

orient_surface *[[surface_id] orient]*;

Orient the triangles of the indicated surface so their nodes are consistently ordered in the same direction. The optional orient parameter is used to set the triangles orientation flag; use 1 to orient them DOWN or -1 to orient them UP. The default is 1. Setting *surface_id*=-1 will process all defined surfaces.

solid extraction commands:

extract_solid_layer *surface_id z_lower z_upper*;

Extracts the solid formed by the volume bounded by the z coordinates *z_lower* and *z_upper* along with the identified *surface_id*. If mutiple groupings result from the solid extraction, each grouping is reorganized in the data structures to occupy its own surface. Note that further advancement of the surface should not be attempted after the solid layer has been extracted. Setting *surface_id*=-1 will process all defined surfaces; otherwise only the specified surface is processed.

create_fastcap_input *[[surface_id] filename]*;

Creates a generic input file readable by FASTCAP from M.I.T. for calculating capacitance matrices. The file created is called *filename*, or the default file "fastcap.inp" can be used. Setting *surface_id*=-1 will process all defined surfaces; otherwise only the specified surface is processed.

execution and plotting commands:

plot3Dmtv *[filename title subtitle xlabel ylabel zlabel node_ID segm_ID tria_ID thin_tria small_tria ridge_segm edge_segm intline_segm surface_id]*;

Create a PLOTMTV input data file for viewing of the surface mesh, and then execute the PLOTMTV program. All parameters are optional and assume a default value when not specified.

node_ID=1 includes the node's ID number in the plot; *node_ID*=0 does not.

segm_ID=1 includes the segment's ID number in the plot; *segm_ID*=0 does not.

tria_ID=1 includes the triangle's ID number in the plot; *tria_ID*=0 does not.

thin_tria=1 will distinguish thin triangles with a magenta color; *thin_tria*=0 does not.

small_tria=1 will distinguish small triangles with a cyan color; *small_tria*=0 does not.

ridge_segm=1 will distinguish segments positioned on a topologically significant ridge with a blueish-green color; *ridge_segm*=0 does not. See the command *ridge_angle*.

edge_segm=1 will distinguish segments positioned on the edge of the surface with a purple color; *edge_segm*=0 does not. An edge segment has only one adjacent triangle.

intline_segm=1 will distinguish segments created during a triangle intersection with a green color; *intline_segm*=0 does not.

surface_id=-1 will plot all the defined surfaces; otherwise plot a specific surface beginning with *surface_id*=0.

plot3Dmtv_vectors *[filename title subtitle xlabel ylabel zlabel node_ID segm_ID tria_ID tria_ndir node_dvect node_copl_vect surface_id]*;

Create a PLOTMTV input data file for viewing of various vector quantities, and then execute the PLOTMTV program. All parameters are optional and assume a default value when not specified.

node_ID=1 includes the node's ID number in the plot; *node_ID*=0 does not.

segm_ID=1 includes the segment's ID number in the plot; *segm_ID*=0 does not.

tria_ID=1 includes the triangle's ID number in the plot; *tria_ID*=0 does not.

tria_ndir=1 will show the triangles' normal vectors; *tria_ndir*=0 does not.

node_dvect=1 will show the nodes' direction vectors; *node_dvect*=0 does not.

node_copl_vect=1 will show the nodes' coplanarity vectors; *node_copl_vect*=0 does not.

surface_id=-1 will plot all the defined surfaces; otherwise plot a specific surface.