DECOMPOSITIONS OF POLYHEDRA IN THREE DIMENSIONS

Tamal Krishna Dey

CSD-TR-91-056
August 1991

DECOMPOSITIONS OF POLYHEDRA IN THREE DIMENSIONS

A Thesis
Submitted to the Faculty

of

Purdue University

by

Tamal Krishna Dey

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

August 1991

This thesis is dedicated to my parents without whose sacrifice this thesis would have been an impossibility.

ACKNOWLEDGMENTS

Many people helped in this thesis work directly and indirectly through their insights, patience, intuition and moral support.

My advisor Chanderjit Bajaj provided the necessary directions when I was only a beginner in the area of this thesis work. He took time to listen patiently to my ideas (not so well formed sometimes) and gave proper directions. At times when things went wrong, it was his moral support and encouragements which put me again on the run.

I must acknowledge the support I got from Kokichi Sugihara when he was visiting Purdue. Discussions with him were enlightening and he was always there to listen to my ideas even at odd hours.

I am thankful to Mikhail Atallah and John Rice for giving their many valuable comments to my questions and their unconditional support on many occasions.

On a personal level, many friends both in the office and at home made my stay at Purdue enjoyable. Special thanks go to Sanjiva Weerawarana, Vinod Anupam, Neelam Jasuja, Malcom Fields, Rajeev Chowdhary who provided their friendship and help on many occasions. I feel really lucky to have friends like Amitava, Sanjib, Subhajit, Saurabh, Soumya whose unconditional support and company made my life lot more easy and enjoyable. Finally, I thank the Department of Computer Science at Purdue University for providing an excellent working environment.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Figure Page

ABSTRACT

Dey, Tamal Krishna PhD., Purdue University, August 1991. Decompositions of Polyhedra in Three Dimensions. Major Professor: Chanderjit L. Bajaj.

This thesis deals with new theoretical and practical results on convex and CSG decompositions, and triangulations of polyhedra in three dimensions. Convex and CSG decompositions of polyhedra find applications in simpler algorithms in motion planning, computer graphics, and solid modeling. Triangulations of polyhedra are fundamental nontrivial steps in finite element simulations and CAD/CAM applications. To reduce ill conditioning as well as discretization error in finite element simulations, near regular shaped elements are desired. This motivates triangulation algorithms for polyhedra that produce well shaped tetrahedra.

We present efficient algorithms for convex and CSG decompositions of polyhedra with arbitrary genus. A modification of this decomposition method gives an efficient algorithm for triangulations of polyhedra. The efficiency of these algorithms is mainly derived from the use of "zone" theorem on hyperplane arrangements, studied in combinatorial geometry. A triangulation algorithm that triangulates a convex polyhedron and a three dimensional point set, in general, with guaranteed quality tetrahedra is also presented. In particular, this algorithm guarantees that four out of five possible bad tetrahedra are never generated.

Geometric algorithms, when implemented under finite precision arithmetic often crash or fail to produce valid output because of numerical errors. We have investigated this problem of output inconsistency under imprecise arithmetic computations

in order to provide topologically robust implementations of the decomposition algorithms. Implementations are carried out as part of SHILP, a solid modeling and display toolkit that runs on Unix workstations under the X Window System.

1. INTRODUCTION

The main purpose behind decomposing an object into simpler components is to simplify a problem for complex objects into a number of subproblems dealing with simpler objects. In particular, the problem of partitioning a polyhedron into simpler components arises in mesh generation for finite element methods, CAD/CAM applications, computer graphics, motion planning, and solid modeling. By a polyhedron we mean a 3-dimensional point set bounded by planar faces. Two dimensional counterparts of polyhedra are polygons. The problem of decomposing polyhedra comes with different flavors depending on the desired shape and size of the simpler components. Although several decomposition problems have been widely researched in two dimensions, very few results exist for their three dimensional counterparts. Two such decompositions, namely, convex decompositions, and triangulations of polyhedra are addressed in this thesis.

1.1 Convex Decompositions

Convex decompositions, in terms of a finite union of disjoint convex pieces are useful and are always possible for polyhedral models [Cha80, Ede87]. In 2D, there are efficient algorithms that decompose a polygon into convex pieces and optimize different metrics (number, length, area, angle) [Kei85, Cha80]. In 3D, however, we have some negative results that restrict our hope to obtain efficient solutions for certain decomposition problems. The problem of partitioning a non-convex polyhedron into a minimum number of convex parts is known to be NP-hard [Lin82, ORS83, DK91]. Further, it is not possible to decompose all polyhedra into convex pieces without introducing extra points, called Steiner points [OR87]. However, all polygons can be decomposed into convex pieces without Steiner points in 2D. Worse is the fact that

the problem of determining whether a polyhedron can be partitioned into tetrahedra (hence convex pieces) without *Steiner* points is NP-hard [RS89]. Due to these restrictions, we consider the problem of convex decompositions of polyhedra that allows Steiner points and achieves only a worst case optimality with respect to the number of convex pieces.

1.1.1 Applications

Convex decompositions lead to efficient algorithms, for example, in geometric point location and intersection detection; see [Ede87]. In motion planning, a disjoint convex decomposition of polyhedra allows for more efficient algorithms in collision detections. In computer graphics, rendering a convex object often comes as a part of graphics library routines supported by specialized hardware and software. To render a nonconvex polyhedron, convex decomposition of the input polyhedron can be used as a first step to generate only convex pieces. Moreover, convex decompositions can be used for efficient algorithms for ray tracing and hidden surface removal in computer graphics.

1.1.2 Polygon Nesting

As a subproblem of our convex decomposition algorithm we encounter the problem of polygon nesting. Given a set of simple polygons that do not intersect along their boundaries, polygon nesting problem asks for detecting the nesting structure of the input, i.e., for each polygon detecting the polygon that immediately contains it. This problem also arises in computer graphics for rendering polygons with multiple holes, and in feature classifications of pattern recognitions.

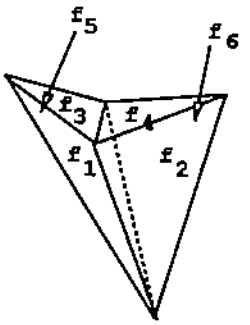
1.2 CSG decompositions

In solid modeling, a geometric object is often represented in terms of simpler components with regularized boolean operations (intersection, union, difference, complement) applied on them. This is called CSG (constructive solid geometry) representation of solids. A polyhedron can be represented as a union of convex pieces obtained through its convex decomposition. The simpler components along with boolean operations used for CSG representation of a polyhedron give equivalently a CSG decomposition of it. Computing a CSG decomposition that involves only union and intersection of the halfspaces corresponding to the supporting planes of the polyhedral facets often arises in graphics and solid modeling [DGHS88]. This type of decomposition was first considered by Peterson [Pet84].

Let $N(p_i)$ represent an ϵ-neighborhood of a point p_i inside the facet f_i of a polyhedron S. The literal f_i^+ represents the halfspace adjacent to the facet f_i that has nonempty intersection with $N(p_i) \cap S$. The literal f_i^- represents the other halfspace adjacent to f_i. Peterson considered the CSG decompositions that use only the halfspaces f_i^+'s. Although it is possible to find such decompositions for polygons in 2D, it is not possible to find such decompositions for polyhedra in 3D in general [DGHS88]. Hence, we allow both halfspaces f_i^+'s and f_i^-'s in the Peterson-style CSG decompositions of polyhedra. This type of CSG decompositions is useful in computer graphics for hidden surface removals [PY90]. A Peterson-style decomposition of a polyhedron S is shown in Figure 1.1. The disjoint convex decompositions of polyhedra can be easily extended to give efficient Peterson-style CSG decompositions.

1.3 Triangulations

In triangulations, we seek for the simplicial decompositions of the given polyhedra that produces a simplicial complex. In 3D, two tetrahedra in such a simplicial decomposition meet only at a full facet, or an edge, or a vertex. A triangulation of a polygonal domain in 2D is shown in Figure 1.2. In finite element mesh generation for



$$c=(f_1^+ \cap f_3^+ \cap f_5^+ \cap f_2^+ \cap f_6^+) \cup (f_4^+ \cap f_3^- \cap f_2^+ \cap f_6^+)$$

Figure 1.1 A Peterson-style CSG decomposition of a polyhedron.

polyhedral domains, triangulation is a nontrivial step. In CAD/CAM, different physical properties of an object are studied through finite element analysis. Triangular element mesh is used very frequently for this purpose.

In 3D, there are polyhedra that can not be triangulated without adding Steiner points. Moreover, as shown by Rupert and Seidel [RS89], the general problem of determining whether a polyhedron can be triangulated without Steiner points is NP-hard. Due to these constraints and as allowed by finite element methods, we consider the problem of triangulations with Steiner points for polyhedra in 3D. We show that the convex decomposition algorithm leads to an efficient algorithm for triangulations of polyhedra.

1.3.1 Good Triangulations

In finite element methods with triangular meshes, it is desired that the elements do not have bad angles [BA76, Fri72, TWM85]. This reduces ill-conditioning and discretization error. In this thesis we refer to such triangulations as good triangulations. Considerable amount of research has been done in 2D to generate triangulations that avoid bad angles. It is known that if Steiner points are not allowed, the Delaunay triangulations maximize the minimum angle among all possible triangulations of a point set in 2D [Sib78, LL86]. This property, however, does not extend through

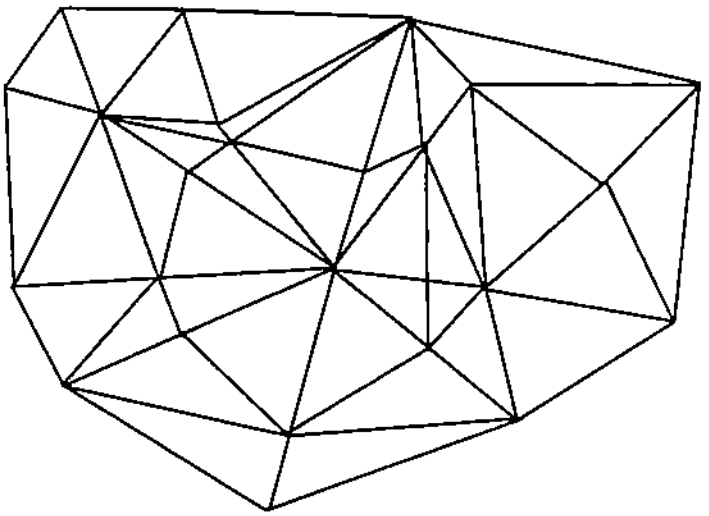


Figure 1.2 A triangulation of a polygon with Steiner points.

higher dimensions [Ede89]. In [ETW90], Edelsbrunner, Tan and Waupotitsch give an algorithm that triangulates a two dimensional point set which minimizes the maximum angle. Such optimum triangulations, however, can itself be bad with respect to the angles. We can hope to improve these triangulations only by adding Steiner points. The choice of Steiner points becomes a crucial factor in producing good triangulations. The algorithms of [TWM85, BGR88, Che89, BEG90, BE91] give different methods to choose these Steiner points.

In 3D, a number of algorithms exist to triangulate a point set or a polyhedron [AE86, EPW86, Joe89, CP90]. Few of them, however, address the problem of guaranteeing the shapes of the tetrahedra. We consider the problem of generating the good triangulations of the convex hull of a point set in 3D. Good triangulations of convex polyhedra are special cases of this problem. In particular, we show that a Delaunay triangulation based algorithm produces an output where four out of five possible bad tetrahedra are never generated.

1.4 Robustness under Finite Precision Arithmetic

Geometric algorithms, when implemented, often fail due to the degeneracies in input data and numerical errors introduced by finite precision arithmetic computations. In general, these algorithms deal with two types of data: numerical and topological. Topological inferences such as face adjacencies, vertex adjacencies are derived from the numerical data. Thus, inaccuracies in numerical computations may cause inconsistencies in topological data which in effect either produce invalid output or make the program fail. The ability of the geometric algorithms to deal with the degeneracies and the inaccuracies during various numerical computations is referred to as their robustness.

Several frameworks for achieving robustness have been proposed by different researchers. Edelsbrunner and Mucke [EM88], and Yap [Yap88] suggest using symbolic perturbation techniques to handle geometric degeneracies. Sugihara and Iri [SI89b], and Dobkin and Silver [DS88] describe an approach to achieve consistent computations in solid modeling by ensuring that computations are carried out with sufficiently higher precision than that used for representing the numerical data. There are drawbacks, however, as high precision routines are needed for all primitive numerical computations making algorithms highly machine dependent. Furthermore, the required precision for calculations is difficult to a priori estimate for complex problems.

Another approach is to live with the finite precision world and tune the arithmetic computations to satisfy certain topological and combinatorial constraints to achieve a consistent result. Certainly, the difficulty of achieving robustness in this approach depends on what we mean by "consistent result". Depending on this meaning of "consistent result", we classify robust algorithms into five categories, namely, type-1, type-2, type-3, type-4 and type-5 robust algorithms.

Definition 1.4.1 The algorithms that satisfy the following properties are called type-1 robust.

1. The programs corresponding to the algorithms never fail with finite precision arithmetics.
2. They produce exact outputs under infinite precision.

Definition 1.4.2 The algorithms that satisfy the following properties are called type-2 robust.

1. They are type-1 robust.
2. The output under finite precision satisfies certain (not necessarily all) essential topological properties of the exact output corresponding to a perturbed input.

Definition 1.4.3 The algorithms that satisfy the following properties are called type-3 robust.

1. They are type-2 robust.
2. The output under finite precision satisfies all topological properties (topologically exact) of the exact output corresponding to a perturbed input.

Definition 1.4.4 The algorithms that satisfy the following properties are called type-4 robust.

1. They are type-2 robust.
2. The output under finite precision satisfies certain (not necessarily all) essential topological properties of the exact output corresponding to a perturbed input, and the perturbation is small. By small perturbation, we mean the size of the perturbation is typically a polynomial function of the input size n, the input precision ϵ, and the maximum value of any coordinate B.

Definition 1.4.5 The algorithms that satisfy the following properties are called type-5 robust.

1. They are type-3 robust.

2. The perturbations required in the input to achieve the topological exactness as stated in type-3 robustness must be small.

One way to achieve the "non-failing" property in type-1 robust algorithms is to guarantee that the contradicting decisions about topological inferences are never taken during the computations. These algorithms have been termed as "parsimonious" by Fortune [For89]. Hoffmann, Hopcroft and Karasick [HHK87], and Karasick [Kar88], propose to use geometric reasoning to avoid contradicting topological inferences and apply it to the problem of polyhedral intersections in an attempt to devise a type-1 robust algorithm.

In type-2 robust algorithms, we focus on certain essential topological properties of the geometric structure of the problem. For example, while computing the Delaunay triangulation of a point set in 2D, we can require that the output be always a planar graph that has a straight line embedding in 2D which is a triangulation. In [SI89a], Sugihara and Iri give a type-2 robust algorithm for constructing the Voronoi diagram of a 2D point set. They ensure that the output produced by the algorithm is always planar, and given infinite precision, it converges to the true solution.

In type-2 and type-3 robust algorithms, we do not quantify the distance between the computed output and the true output geometrically. In type-4 and type-5 robust algorithms, we quantify the distance between the computed output and the true output both topologically and geometrically. In particular, we require that the computed output satisfies topological properties of an output corresponding to some perturbed input and the required perturbations be small. The algorithm proposed by Fortune and Milenkovic in [FM91] for line arrangements is type-4 robust. It produces an arrangement of pseudo lines that satisfy the certain essential properties of line arrangements and the required perturbations are proved to be small. To devise type-4 and type-5 robust algorithms, we must assume a bound on the relative error in the basic arithmetic computations: plus, minus, divide, multiply. Guibas, Salesin, and Stolfi [GSS89] propose a framework of computations, called ε-geometry, with which they give type-5 robust algorithms for some 2D problems. So does Fortune [For89]

who gives type-5 robust algorithms for the problem of computing the convex hull and the triangulation of a planar point set. The algorithm proposed by Li and Milenkovic in [LM90] for computing the convex hull are also type-5 robust.

In type-2 and type-3 robust algorithms, we may not require any bound on the relative error in basic arithmetic computations to achieve only topological exactness. Nevertheless, while designing type-2, type-3 robust algorithms, it is advisable to assume such bounds and perform arithmetic computations with thresholds as described in Section 3.4, and Section 5.4. It is our hope that, in many cases, type-2, type-3 robust algorithms become actually type-4, type-5 robust with such thresholded computations, though we cannot prove it.

The difficulty of designing robust algorithms depends upon the problem itself. For geometric operations (intersection, union, decomposition) on polyhedral models, it is very difficult to design even type-1 robust algorithms. The only success achieved so far in this respect is by Hopcroft and Kahn [HK89]. They have given a type-5 robust algorithm for computing the intersection of a halfspace with a convex polyhedron under certain minimum feature assumptions. On the other hand, for problems such as intersections of two lines [GSS89], convex hull of a 2D point set [For89, LM90], where topology is not very intricate, it is easier to design type-5 robust algorithms.

We give a type-5 robust algorithm for polygon nesting with a minimum feature assumption. It is type-5 robust since it computes the nesting structure correctly under finite precision computations and thus require zero perturbations of the input to satisfy the computed output.

In our convex decomposition algorithm, we use sophisticated heuristics based on geometric reasoning which make the algorithm more stable against numerical errors. Although we cannot prove that the algorithm with these heuristics is type-1 robust, our experimental results have been satisfactory.

In our effort to design a robust Delaunay triangulation algorithm in 3D, we first identify certain essential topological properties of the underlying graph of the triangulation of a 3D point set. This topological properties are used to achieve a type-2

robust algorithm for the Delaunay triangulations in 3D. This robust Delaunay triangulation algorithm is used in a robust implementation of our good triangulation algorithm in 3D.

1.5 Some Topological Aspects of Polyhedra

A surface is a 2-manifold if each point on it has an ϵ-neighborhood that is homeomorphic to an open 2D ball or half-ball [Arm79]. In this thesis, we will refer to 2-manifolds simply as manifolds. A manifold surface is called closed if it does not have a boundary, i.e., all points on it has an ϵ-neighborhood that is homeomorphic to an open 2D ball. For example, the surface of a sphere is a closed manifold whereas a rectangular patch on a plane is not. A manifold is called oriented if it has two distinct sides. the surface of a sphere is oriented since it has two sides, "inside" and "outside". The surfaces of Mobius strips and Klein bottles are not oriented [Arn62]. Polyhedra, having closed oriented manifold surfaces are called manifold polyhedra. Non-manifold polyhedra may have incidences as illustrated in Figure 3.1.

A polyhedron may have *through holes* which determine its *genus*. It may also have *internal voids* and thus have a disconnected boundary. Manifold polyhedra with holes are homeomorphic to *torii* with one or more *handles*. Manifold polyhedra with internal voids are homeomorphic to 3-dimensional *annuli*, i.e., spheres with internal voids. A polyhedron can be represented with its boundary which consists of three disjoint open point sets, called vertices (0-dimensional), edges(1-dimensional), and facets(2-dimensional). A systematic enumeration of vertices, edges, and faces with all relevant adjacency information gives a boundary representation (B-rep) of a polyhedron.

1.6 Results

For a given polyhedron S with n edges of which r are reflex [1], Chazelle [Cha80, Cha84] established a worst case lower bound of $O(r^2)$ on the number of convex polyhedra needed for complete convex decomposition of S. He gave an algorithm that produces a worst case, optimal number ($O(r^2)$) of convex polyhedra in $O(nr^3)$ time and $O(nr^2)$ space. Recently, Chazelle and Palios [CP90] have given an $O((n+r^2)\log r)$ time and $O(n+r^2)$ space algorithm to tetrahedralize a subclass of non-convex polyhedra. This algorithm, however, only allows polyhedra that are homeomorphic to a 2-sphere, i.e., have no holes(genus 0) and shells (internal voids). Our algorithm, based on the split and cut method of Chazelle [Cha80, Cha84], runs in $O(nr^2 + r^3 \log r)$ time and uses $O(nr + r^2\alpha(r))$ space. Here, α is the inverse Ackermann's function which grows extremely slowly. Thus, our algorithm improves upon the algorithm of Chazelle [Cha80, Cha84] w.r.t. time and space complexities and that of [CP90] w.r.t. the generality of the input. We also give an algorithm for convex decompositions that uses geometric based heuristics to overcome the inaccuracies involved with finite precision arithmetic computations. Although we cannot prove that these heuristics make the algorithm type-1 robust, the experimental results are very satisfying. These results appear in [BD91].

As a subproblem of our convex decomposition algorithm, we solve the problem of polygon nesting efficiently. Our algorithm for this problem runs in $O(n + (m + r)\log(m + r))$ time, where n is the total number of vertices in m polygons with r reflex vertices [2]. Note that m and r are much less than n in practice. We also give a type-5 robust algorithm for this problem with a minimum feature assumption on the "thickness" of the polygons. This algorithm runs in $O(n(\log n + m + r))$ time. These results appear in [BD90].

Simple extensions of our convex decomposition algorithm give efficient algorithms for Peterson-style CSG decomposition and triangulation of polyhedra. An $O(p^2\alpha(p))$

[1] edges where the internal dihedral angle is greater than 180°

[2] vertices where the internal angle is greater than 180°

size Peterson-style CSG decomposition can be computed in $O(p^3 \log p)$ time through our convex decomposition algorithm. This improves upon the algorithm of [PY90] w.r.t. the generality of the input. Their algorithm computes an $O(p^2)$ size Peterson-style CSG decomposition in $O(p^3)$ time for polyhedra with convex facets of $O(1)$ size. We also establish $O(p^2)$ lower bounds on certain types of Peterson-style CSG decomposition of polyhedra. The best known algorithm for triangulating polyhedra [CP90] runs in $O((n + r^2) \log r)$ time which produces $O(n + r^2)$ tetrahedra. This algorithm, however, has two drawbacks. Firstly, it allows only the simple polyhedra that have no holes and shells. Secondly, it produces a simplicial decomposition that is not a simplicial complex i.e., the generated tetrahedra do not meet at a full facet or an edge. A simple extension of our convex decomposition algorithm gives an $O(nr^2 + r^3 \log r)$ time algorithm for triangulating more general polyhedra (with holes and shells) which generates $O(nr + r^3)$ tetrahedra in a simplicial complex. Thus, our algorithm improves upon the algorithm of [CP90] w.r.t. the generality of the input and output. These results appear in [Dey91].

In 2D, there are number of algorithms for generating good triangulations. Chew in [Che89], gives an algorithm based on the constrained Delaunay triangulations which guarantees that all triangles have angles between 30° and 120°. In [Dey90], we improved this algorithm with minor modifications to guarantee better angle bounds for the boundary triangles. There is another approach, based on the *Grid Overlaying*, which was first used by Baker, Grosse, and Raferty in [BGR88] to produce a non-obtuse triangulation of a polygon. In [Dey90], we proposed a simpler method based on this grid approach to triangulate a polygon with good angles. Recently, in [BEG90], Bern, Eppstein, and Gilbert give algorithms for producing good triangulations which uses a special type of a grid that simulates the planar subdivision with the quadtree. Another approach, based on the *medial axis transformation*, is proposed by Srinivason, Nackman, and Tang to produce an adaptive triangular mesh that eliminates bad triangles [SNT90].

Although several good heuristics have been published, to date there is no known algorithm that triangulates the convex hull of a three dimensional point set with guaranteed quality tetrahedra. We present some results on the good triangulations of the convex hull of a point set in 3D. Good triangulations of convex polyhedra are a special case of this problem. Our main results in good triangulations are: (i) a 3D triangulation algorithm based on the Delaunay triangulations, as used by Chew [Che89] in 2D, to produce triangulations that do not have four out of five possible types of bad tetrahedra, (ii) a bound on the number of additional points used to achieve this guarantee, (iii) a type-2 robust algorithm for Delaunay triangulations in 3D that is used to produce a robust implementation of the good triangulation algorithm. These results appear in [DBS91].

1.7 Organization

We, first, describe the polygon nesting algorithm in Chapter 2 since it appears as a subproblem in our convex decomposition algorithm. It also makes the readers to be more familiar with the concepts of robustness. Chapter 3 describes the convex decomposition algorithm and the heuristics used for its robust implementation. CSG decompositions and triangulations of polyhedra with arbitrary genus and shells are described in Chapter 4. Good triangulations of the convex hull of a point set in 3D is presented in Chapter 5. It also describes a robust algorithm that is used in robust implementation of the good triangulation algorithm in 3D. Finally, we conclude this thesis in Chapter 6 which summarizes the contribution of this work and presents some related open problems.

2. POLYGON NESTING

2.1 Introduction

This chapter describes an efficient algorithm for polygon nesting that arises as a fundamental subproblem in our convex decomposition algorithm. Section 2.3 describes the algorithm under the assumption of exact arithmetic. Section 2.4 presents a type-5 robust algorithm that assumes a minimum feature for the input.

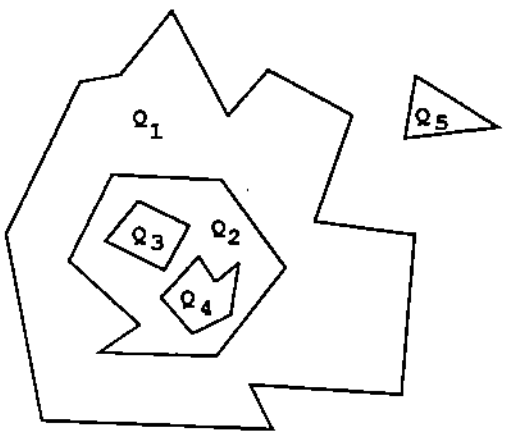


Figure 2.1 Polygon nesting.

Let $\wp$ be a set of m simple polygons $Q_i, i = 1..., m$ that do not intersect along their boundaries.

Definition 2.1.1 The *ancestor* of a polygon Q_i is defined as the set of polygons containing Q_i inside and is denoted as $ancestor(Q_i)$.

Definition 2.1.2 The polygon Q_k in $ancestor(Q_i)$ is called the *parent* of Q_i if $ancestor(Q_k)$=$ancestor(Q_i)$ - Q_k. Notice that there may not exist any such Q_k since $ancestor(Q_i)$ may be empty. In that case we say that the parent of Q_i is *null*.

Definition 2.1.3 The polygons whose parent is Q_k are called the *children* of Q_k and are denoted as $children(Q_k)$.

In Figure 2.1, $ancestor(Q_3) = \{Q_1, Q_2\}$, $parent(Q_3) = \{Q_2\}$, $children(Q_2) = \{Q_3, Q_4\}$, and $parent(Q_5) = children(Q_5) = null$. The nesting structure G of $\wp$ is an acyclic directed graph (a forest of trees) in which there is a node n_i, corresponding to each polygon Q_i in $\wp$, and there is a directed edge from a node n_i to n_j if and only if Q_j is the parent of Q_i. The polygon nesting problem is to compute the nesting structure of a set of simple polygons that do not intersect along their boundaries.

Given a set of simple nonintersecting polygons with n vertices, Chazelle, in [Cha84], gives an $O(n \log n)$ algorithm to detect the outermost polygons and their children. However, his algorithm does not compute the nesting structure of the given set of polygons.

In Section 2.3, we give an algorithm which computes the polygon nesting structure in $O(n+(m+r)\log(m+r))$ time where n is the total number of vertices in m polygons and r is the total number of reflex vertices. Since in practice m and r are much less than n, this algorithm runs faster than any $O(n \log n)$ algorithm in practice. In Section 2.4, we give a type-5 robust algorithm for the same problem restricted to a class of polygons called *fleshy* polygons. Our robust algorithm has a worst-case time bound of $O(n(\log n + m + r))$.

2.2 Preliminaries

Let Q be a simple polygon with vertices $v_1, v_2, ..., v_n$ in clockwise order. Between any two consecutive reflex vertices v_i, v_j in the clockwise order, the sequence of vertices $(v_i, v_{i+1}, ..., v_j)$ is called a *convex polygonal-line.*

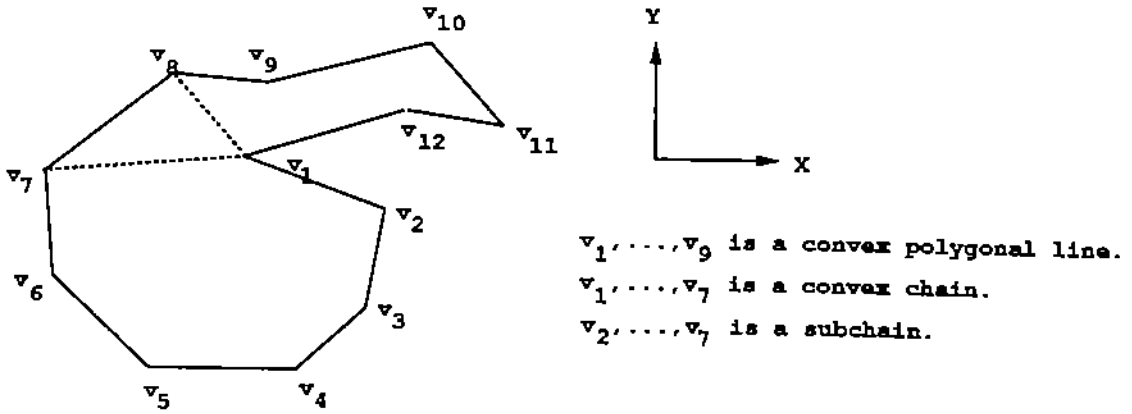


Figure 2.2 Convex chain and subchain.

Definition 2.2.1 A *convex chain* is a maximal piece of a convex polygonal-line with the property that its vertices form a convex polygon.

Definition 2.2.2 A *subchain* is a maximal piece of a convex chain with the property that the vertices of a subchain have x-coordinates in either strictly increasing or decreasing order.

Each convex polygonal-line can be partitioned into convex-chains and each convex chain can be partitioned into at most three subchains; see Figure 2.2.

A vertex or an edge is said to lie inside a polygon if it completely lies in the interior of the polygonal region. A vertex or an edge is said to be contained in a polygon if it lies on the boundary of the polygon.

Let L be a line drawn through a set of polygons. Let E be the set of edges that intersect L in the following two ways. An edge e in E either properly intersects L (i.e., two vertices of e lies on the opposite sides of L) or e intersects L at a vertex and the other vertex of e lies to the right of L. The third possible case, where one vertex of e lies on L and the other one to the left of L, is ignored as the information related to that edge would already be recorded in a plane sweep of our algorithm.

The remaining case of degenerate intersection (e is collinear with L) is described in section 2.3.

Definition 2.2.3 A vertex v_i is said to lie above v_j if the y coordinate of v_i is greater than that of v_j.

Definition 2.2.4 An edge e_1 is said to be *above* the edge e_2 in E if the point $L \cap e_1$ lies above the point $L \cap e_2$. If e_1 and e_2 have a common vertex through which L passes, e_1 is *above* e_2 if the other vertex of e_1 lies above the other vertex of e_2.

The line L induces a total order R on the edges in E with respect to the *above* relation. If L passes through a vertex v_i, *above*(v_i) denotes the set of edges whose point of intersection with L is above v_i. The lowest edge in *above*(v_i) is called the *neighbor* of v_i. Between v_i and its neighbor there is no other edge intersecting L. In Figure 2.4, e_4 is the neighbor of v since it is the lowest edge in *above*(v). Note that there may not exist any neighbor of v_i since *above*(v_i) may be empty. Order R naturally extends to another order O of subchains associated with the edges in R.

Definition 2.2.5 The subchain C_1 containing the edge e_1 is *above* the subchain C_2 containing the edge e_2 in O if and only if e_1 is *above* e_2 in R.

2.2.1 Useful Lemmas

In the following lemma, the line segments of a line that are interior to a polygon are called *chords*.

Lemma 2.2.1 Let Q be a polygon (possibly with holes) with r reflex vertices. No line can intersect Q in more than $r + 1$ chords.

Proof: The proof proceeds inductively. The case for $r = 0$ is trivial. In the general step, consider a polygon Q with $r = k \geq 1$ reflex vertices. Take an arbitrary reflex vertex, and resolve it by a cut through it. The cut may separate Q into two polygons Q_1 and Q_2 of r_1 and r_2 reflex vertices respectively, such that $r_1 + r_2 \leq k - 1$.

Furthermore, the number of chords of a line L in Q cannot exceed the sum of the number of chords in Q_1 and Q_2. Therefore, using the induction hypothesis, one can conclude that the line L intersects Q in no more than $r_1 + 1 + r_2 + 1 \leq k + 1$ chords. If, however, the cut does not split Q, one ends up with a polygon Q' of at most $k - 1$ reflex vertices. Since the line L may intersect the cut, just performed, the number of chords in Q is less than or equal to that in Q', which again implies that the former is less than or equal to $k - 1 + 1 \leq k + 1$. ♣

Lemma 2.2.2 Let Q be a simple polygon with r reflex vertices. The number of subchains c in Q is bounded as $c \leq 6(1 + r)$.

Proof: Follows from Theorem 3, page 22 of [Cha80]. ♣

Lemma 2.2.3 Let L be any line through a vertex v of a polygon Q_i. Let the edge e be the neighbor of v. Parent of Q_i is either the polygon Q_j containing e or Q_j's parent (possibly *null*).

Proof: If the neighbor edge e of v is an edge of Q_j which is the parent of Q_i, the lemma holds trivially. Suppose the neighbor edge e of v is an edge of Q_j which is not the parent of Q_i. We claim that v lies inside a polygon Q_ℓ if and only if e lies inside it. Suppose e lies inside Q_ℓ, and v does not. Then the region between v and e on L contains a part which is outside Q_ℓ. Hence, there must be an edge of Q_ℓ between e and v intersecting L. But this is impossible since e is the neighbor edge of v. Similarly, we can argue that if v lies inside a polygon Q_ℓ, so does e. Hence e lies inside the same set of polygons, within which v lies. Thus, Q_k is the parent of Q_i if and only if it is a parent of Q_j. ♣

Lemma 2.2.4 Let L be any line passing through v of Q_i. The vertex v is contained in the polygon $Q_{k,k\neq i}$ if and only if the number of edges of Q_k which are in $above(v)$ is odd.

Proof: Since any edge of a polygon Q demarks the regions "inside Q" and "outside Q" on L, the above lemma is obvious. ♣

Lemma 2.2.5 Let L be any line passing through v of Q_i. Let the edge e of the polygon Q_k be the neighbor of v on L. If the number of edges of Q_k in $above(v)$ is odd and $k \neq i$, then Q_k is the parent of Q_i. Otherwise, Q_k's parent(possibly *null*) is the parent of Q_i.

Proof: Combine Lemma 2.2.3 and Lemma 2.2.4. ♣

2.3 The Algorithm with Exact Arithmetic

Now, we describe the algorithm which is based on the plane sweep and uses exact arithmetic for all numerical computations. Each polygon Q_i consists of subchains $C_{i1}, C_{i2}, ..., C_{ik}$. We sweep a line L in the plane through all polygons, while maintaining the ordering O of the subchains induced by L. To maintain this ordering, we stop only at the endpoints of the subchains, while sweeping, say, from left to right. We break all the boundaries of the polygons into subchains in no more than $O(n)$ time and sort their endpoints on a line perpendicular to L. At each subchain endpoints we update the ordering O.

2.3.1 Update at a Vertex

If v is such a vertex that both subchains C_1 and C_2 connected to v have not yet been encountered by the sweep line L, we insert C_1 and C_2 in the ordering O on L by a simple binary search. For this search, we need to determine the position of v w.r.t. the edge intersected by L on a subchain C_i, already present in the ordering O.

This is done as follows. We keep the last visited edge associated with each subchain C_i in O. Let the last edge kept associated with C_i be e_1. We visit the sequence of edges $e_1, e_2..., e_k$ of C_i stopping at the first edge e_k which intersects L. We determine the position of v w.r.t. e_k and associate the edge e_k with C_i. Later, when we need to classify any other vertex w.r.t. C_i, we start from the edge e_k. This is reminiscent of the topological sweep of [EGS9]. In this sweep, the sweep line is actually a curved line, called pseudo-line. See Figure 2.3. Obviously, the edges like $e_2, ... e_{k-1}$ are

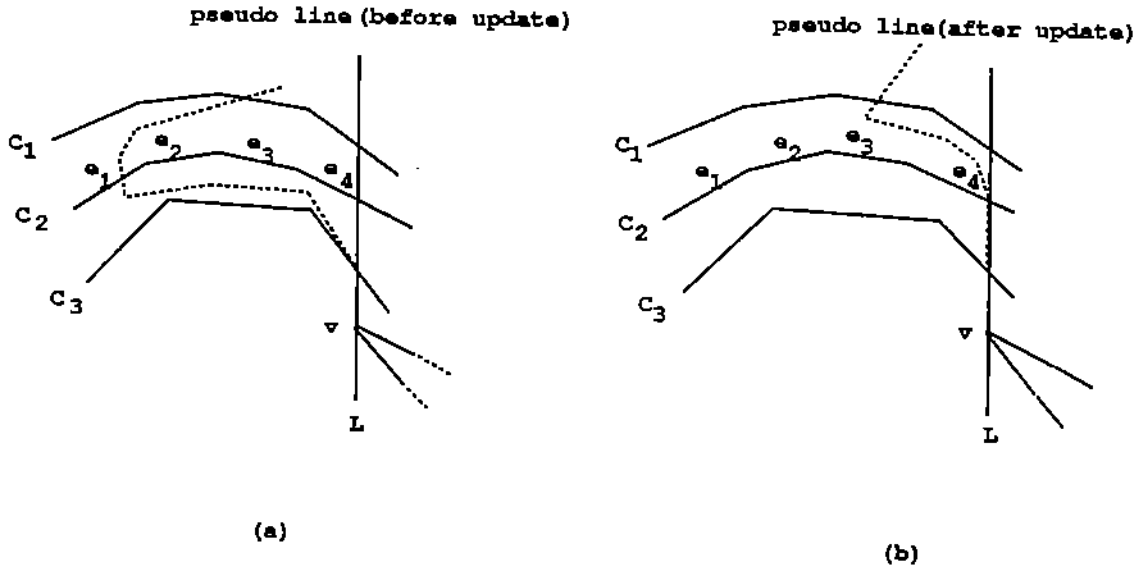


Figure 2.3 Sweeping status before and after the update at v.

visited only once, while edges like e_1 and e_k are visited more than once throughout the sweep. For each vertex-edge classification, there will be at most two edges similar to e_1 and e_k of a subchain which will be visited more than once throughout the sweep. Since in the binary search for determining the position of a vertex in the order O, we encounter only $O(\log c)$ subchains (c is the total number of subchains), there will be at most $O(\log c)$ edges, for each sweep line position, which will be visited more than once. Let t_i be the number of edges in subchain C_i which are visited only once throughout the sweep. As we observed, only $O(\log c)$ edges per update are visited that are encountered more than once throughout the sweep. If v is a vertex such that both subchains connected to v have been encountered, we delete both these subchains from the ordering O. This takes at most $O(\log c)$ time. Hence, the total time taken for all updates is $O\left(\sum_{i=1}^{c} t_i\right) + O(c \log c)$. Certainly, $\sum_{i=1}^{c} t_i = O(n)$ where n is the total number of vertices. Hence updates take $O(n) + O(c \log c)$ time.

2.3.2 Detecting the Parent of a Polygon

At the vertex v of Q_i, when we insert the subchains in the ordering O on L, we determine the parent of Q_i as follows. If the parent of Q_i has not already been determined, we find the neighbor edge e of v intersecting L (Actually, e is found while inserting the subchains connected to v). Let Q_j be the polygon containing e on the boundary. We determine k, the number of edges or equivalently the number of subchains of the polygon Q_j which are in *above*(v). Maintaining the ordering of subchains for each polygon separately, this number can be obtained in $O(\log c_i)$ time where c_i is the number of subchains in that polygon. If k is odd and $Q_j \neq Q_i$, we set Q_j as the parent of Q_i. Otherwise, we set the parent of Q_j to be the parent of Q_i (Lemma 2.2.5). Certainly, the parent determination at each update add up to at most $O(\log c)$ time.

2.3.3 Degenerate Cases

Degeneracy occurs when the sweep line L passes through more than one vertex, at any stop of L. In these cases, one or more than one edge may be collinear with L. Let $v_1, v_2, ..., v_k$ be the ordered sequence (w.r.t. the *above* relation) of vertices through which L passes at any stop.

We process each vertex v_i in the ordered sequence one after the other as follows. Let v_i be the vertex of polygon Q_i. For v_i, we insert or delete accordingly the subchain that does not correspond to the edge collinear with L from the ordering O. Since the edge collinear with L does not demark any region on L as "inside Q_i" or "outside Q_i", we should not insert that edge in the ordering O and in the ordering maintained separately for each polygon. Hence, a degenerate edge does not affect the number of edges of Q_i which would be in *above*(v_i) for any vertex v_i. See also Figure 2.4.

2.3.4 The Algorithm

Algorithm Polnest-Exact:

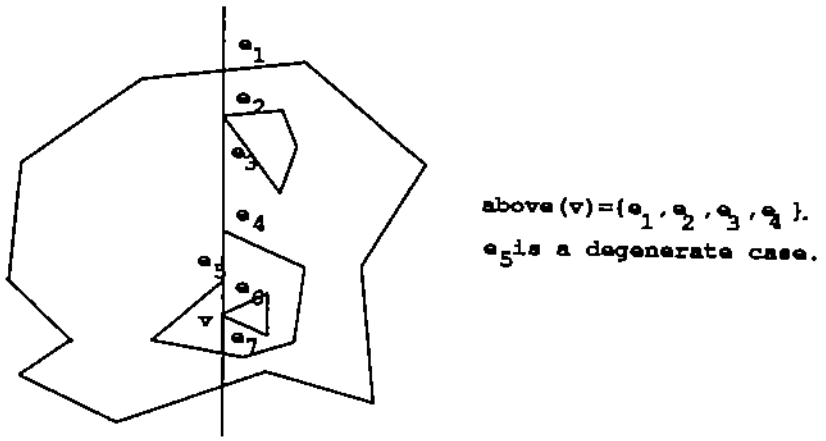


Figure 2.4 Degenerate cases.

Input: A set of m simple, nonintersecting polygons.

Output: A directed acyclic graph G, called the nesting structure, in which there is a directed edge from a node n_i corresponding to a polygon Q_i to the node n_j corresponding to the polygon Q_j if and only if Q_j is the parent of Q_i.

Step 1: Detect the endpoints of subchains in all polygons.

Step 2: Sort the x-coordinates of these endpoints. If two points have same x-coordinates, the one with higher y-coordinate is sorted before the other. Let this sorted sequence W be $v_1, v_2, ..., v_w$.

Step 3: Create a node for each polygon in G. Initialize O by inserting the two polygon edges as the representatives of the two subchains connected to the leftmost vertex in W.

Step 4: Sweep a pseudo-line from left to right taking steps at each vertex v_j of W as follows. Let v_j be on the boundary of the polygon Q_i. If both subchains connected to v_j have already been visited, delete them from the ordering O and skip steps from 4(a) to 4(d).

Step 4(a): Detect the position of v_j w.r.t. the subchains intersected by the sweep line. For this, carry out a binary search in the ordering O of these subchains. To

detect the position of v_j w.r.t. a subchain C_i during binary search, find the edge e_1 kept associated with C_i in O, and then follow the linked sequence of edges $e_1, e_2, ..., e_k$ until the edge e_k is found which intersects L.

Step 4(b): Let the edge e' of the polygon Q_j be the neighbor of v_j found in step 4(a). Determine the number of subchains k of Q_j that are in $above(v_j)$. This is done by a similar binary search, as in step 4(a), in the ordering of subchains maintained separately for each polygon.

Step 4(c): Insert two subchains connected to v_j in O and in the ordering of subchains maintained for polygon Q_i. In the degenerate case, insert or delete the subchain from O that does not correspond to the edge, collinear with the sweep line.

Step 4(d): If k is odd, create a directed edge in G from the node n_i corresponding to the polygon Q_i to the node n_j corresponding to the polygon Q_j. If k is even, create a directed edge from n_i to the node n_k(if any) to which n_j is connected through a directed edge.

Theorem 2.3.1 The problem of polygon nesting for m polygons can be solved in $O(n + (m + r)\log(m + r))$ time where n is the total number of vertices, and r is the total number of reflex vertices of all polygons.

Proof: Detecting the endpoints of the subchains takes $O(n)$ time. Sorting these endpoints requires $O(c \log c)$ time. Updating and determining the parent takes $O(n + c \log c)$ time. Hence, computing the nesting structure for all polygons takes $O(n + c \log c)$ time. By Lemma 2.2.2, c, the total number of subchains is bounded as $c \leq 6(m + r)$ where m is the total number of polygons, and r is the total number of reflex vertices. Hence, the total time spent is $O(n + (m + r)\log(m + r))$. ♣

2.4 Robustness under Finite Precision Arithmetic

In the algorithm given in the previous section, we assumed exact arithmetic in all our arithmetic computations. In this section, we give an algorithm for polygon nesting problem which is type-5 robust under finite precision arithmetic computations. This

algorithm is type-5 robust since it never fails and gives always the correct output under a minimum feature assumption.

2.4.1 Assumptions and Finite Precision Computations

We first assume that all coordinates have a maximum absolute value of B i.e., $-B < x < B$ and $-B < y < B$. We model the inexact arithmetic computations by ε-arithmetic [For89, GSS89] where the arithmetic operations $+, -, \div, \times$ are performed with relative error of at most ε.

Definition 2.4.1 A polygon Q is called *fleshy* if there is a point inside Q such that a square with the center(intersection of square's diagonals) at that point and with the sides of length $64\varepsilon B$ lies inside Q. Here, ε is the machine precision.

In our implementation, we set $B = 2^{10}$ units, $\varepsilon = 2^{-32}$ units. Hence the area of the square is 2^{-32}. The polygons that are not fleshy are thus too skinny to occur in most practical cases.

Definition 2.4.2 A binary predicate $CONT$ takes two polygons Q_1, Q_2 as arguments and returns true if and only if Q_1 contains Q_2. $NOT(CONT(Q_1, Q_2))$ denotes the negation of $CONT(Q_1, Q_2)$.

Definition 2.4.3 A point p_1 is said to be vertically visible from another point p_2 if the vertical line through p_2 also passes through p_1 and the vertical segment between p_1 and p_2 does not intersect any other edge. Similarly, we define an edge to be vertically visible from a point p_1 if the vertical line through p_1 intersects the edge and does not intersect any other edge in between.

The numerical computations in our algorithm are carried out at two places.

1. Sorting the vertices:

   Sorting can be carried out without any error as the comparison of two floating point numbers is exact upto the machine precision. A similar model of computations where comparisons of input data are free of error has also been assumed

by [Mil88, For89] (this fact is true on most of the machines available today). Here we assume that the given input data (coordinates of polygon vertices) is accurate, though our algorithm tolerates perturbations in the input that does not destroy the simplicity and nonintersecting properties of the polygons.

2. Computing the points of intersections of a vertical sweep line with the edges:

   In Lemma 2.4.1, we develop a bound on the maximum error that can occur during this computation.

Lemma 2.4.1 Given an edge e between two vertices $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$, and a vertical line passing through a vertex $v_0 = (x_0, y_0)$ intersecting the edge e at a point p, the absolute error e^p_{abs} in the computed position of p is bounded as $e^p_{abs} < 7\varepsilon B$. The absolute error e^d_{abs} in the computed distance of v_0 and p is bounded as $e^d_{abs} < 8\varepsilon B$. Here ε is the machine precision and B is the largest value of any coordinate.

Proof: Consider a vertical line $x = x_0$ through $v_0 = (x_0, y_0)$ that intersects e at p. Obviously, the x-coordinate of p is x_0. Let the actual and computed y-coordinate of p be y_3 and y_c. By simple geometry,

$$\begin{aligned} \frac{x_2 - x_1}{x_0 - x_1} &= \frac{y_2 - y_1}{y_3 - y_1} \\ y_3 &= \frac{(y_2 - y_1)(x_0 - x_1)}{x_2 - x_1} + y_1. \end{aligned}$$

With floating point arithmetics, the computed value y_c of y_3 is given by

$$y_c = \frac{(y_2 - y_1)(x_0 - x_1)(1 + \varepsilon^*)}{(x_2 - x_1)} + y_1(1 + \varepsilon_6)$$

where $(1 + \varepsilon^*) = \frac{(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_4)(1+\varepsilon_5)(1+\varepsilon_6)}{(1+\varepsilon_3)}$ and $|\varepsilon_i| \leq \varepsilon$. Let $t_0 = \frac{(y_2-y_1)(x_0-x_1)}{x_2-x_1}$. We can write

$$\begin{aligned} y_c &= t_0(1 + \varepsilon^*) + y_1(1 + \varepsilon_6) \\ y_c - y_3 &= t_0\varepsilon^* + y_1\varepsilon_6 \\ e^y_{abs} &\leq |t_0\varepsilon^*| + |y_1\varepsilon|. \end{aligned}$$

Neglecting higher order terms in ε_i, we get $|\varepsilon^*| \leq 6\varepsilon$. Since $\frac{|x_0-x_1|}{|x_2-x_1|} < 1$, we have

$$\begin{aligned} |t_0| &< |y_2 - y_1| \\ |t_0| &< B \\ e^p_{abs} &< 6\varepsilon B + \varepsilon B \\ e^p_{abs} &< 7\varepsilon B. \end{aligned}$$

The distance between v_0 and p is computed as $|y_c - y_0|$ which introduces additional error of at most $\varepsilon|y_c - y_0| \leq \varepsilon B$. Thus, the total error in the distance computation of p from v_0 is bounded as $e^d_{abs} < 7\varepsilon B + \varepsilon B = 8\varepsilon B$. ♣

2.4.2 Good Vertex

We define a vertex v of a simple polygon Q_i to be a "good vertex" as follows. Let L be a vertical line passing through v. The set of intersection points of the edges of any polygon Q_j with this vertical line can be partitioned into three sets $I^j_{above(v)}, I^j_{close(v)}, I^j_{below(v)}$ based on the proximity of the intersection points to v. $I^j_{above(v)}$ is the set of all intersection points above v whose computed distance from v is greater than or equal to $8\varepsilon B$. $I^j_{below(v)}$ is correspondingly defined for intersection points below v. The rest of the intersection points are in the set $I^j_{close(v)}$.

Definition 2.4.4 For the polygon Q_i containing v, if all points in $I^i_{below(v)}$ (respectively $I^i_{above(v)}$) are at a computed distance of at least $24\varepsilon B$ from v, and if $|I^i_{below(v)}|$(respectively $|I^i_{above(v)}|$) is odd then v is called a "good vertex" of Q_i from below (respectively above).

Since the absolute error in the distance computations of the intersection points from v is less than $8\varepsilon B$. the intersection points in $I^i_{close(v)}$ can lie at an actual distance of at most $16\varepsilon B$ either below or above v. On the other hand, the actual distance between v and the points in $I^i_{below(v)}$ (respectively $I^i_{above(v)}$) must be greater than $16\varepsilon B$. Hence, there must be a segment of L that lies between the points in $I^i_{close(v)}$ and $I^i_{below(v)}$ (respectively $I^i_{close(v)}$ and $I^i_{above(v)}$). This segment lies inside Q_i if $|I^i_{below(v)}|$ (respectively $|I^i_{above(v)}|$) is odd.

Lemma 2.4.2 Given two simple, nonintersecting polygons Q_1, Q_2, it can be correctly determined if one of the predicates $NOT(CONT(Q_1, Q_2))$, $NOT(CONT(Q_2, Q_1))$ is true by checking the leftmost vertices of Q_1 and Q_2.

Proof: Let $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$ be two leftmost vertices of Q_1 and Q_2 respectively. Certainly, $x_1 < x_2$ implies $NOT(CONT(Q_2, Q_1))$, and $x_1 > x_2$ implies $NOT(CONT(Q_1, Q_2))$. Furthermore, $x_1 = x_2$ implies $NOT(CONT(Q_1, Q_2))$ and $NOT(CONT(Q_2, Q_1))$ since Q_1 and Q_2 are simple nonintersecting polygons. ♣

2.4.3 Procedure ANSC

The procedure $ANSC$, when called with the argument v, a good vertex of P_i, reports some(may not be all) ancestors of Q_i as follows. W.l.o.g., assume v is a "good vertex" of the polygon Q_i from below. The procedure ANSC constructs $I^j_{close(v)}, I^j_{below(v)}, I^j_{above(v)}$ for all polygons $\{Q_j\}$. Let $\{Q_k\}$ be the set of polygons for which $|I^k_{below(v)}|$ is odd, and all points in $I^k_{below(v)}$ lie at a computed distance of at least $24\varepsilon B$ from v. The procedure ANSC reports those polygons in $\{Q_k\}$ as the ancestors of Q_i whose leftmost vertex has a smaller x-coordinate than that of Q_i.

Lemma 2.4.3 Given a set of simple, nonintersecting polygons in the plane with a "good vertex" v on the polygon Q_i, reported ancestors of the polygon Q_i by $ANSC(v)$ are true ancestors of it.

Proof: Let L be a vertical line passing through a "good vertex" v of Q_i. As stated earlier, a "good vertex" can easily be determined via distance computations of v from the intersection points of edges with L. W.l.o.g., assume v to be a "good vertex" of Q_i from below. Since v is a "good vertex" from below, there is a segment L' of L that lies between the points in $I^i_{close(v)}$ and $I^i_{below(v)}$. In $ANSC(v)$ we consider the set of polygons $\{Q_k\}$ that have odd number of points in $I^k_{below(v)}$ which lie at a computed distance of at least $24\varepsilon B$ from v. Certainly, a portion of the segment L' also lies inside these polygons. Hence, a polygon in the set $\{Q_k\}$ either contains Q_i or is contained in Q_i. We use Lemma 2.4.2 to eliminate one of these two possibilities and report those

polygons in $\{Q_k\}$ which contain Q_i. Hence, the reported polygons truly contain Q_i, though all polygons containing Q_i may not be reported. ♣

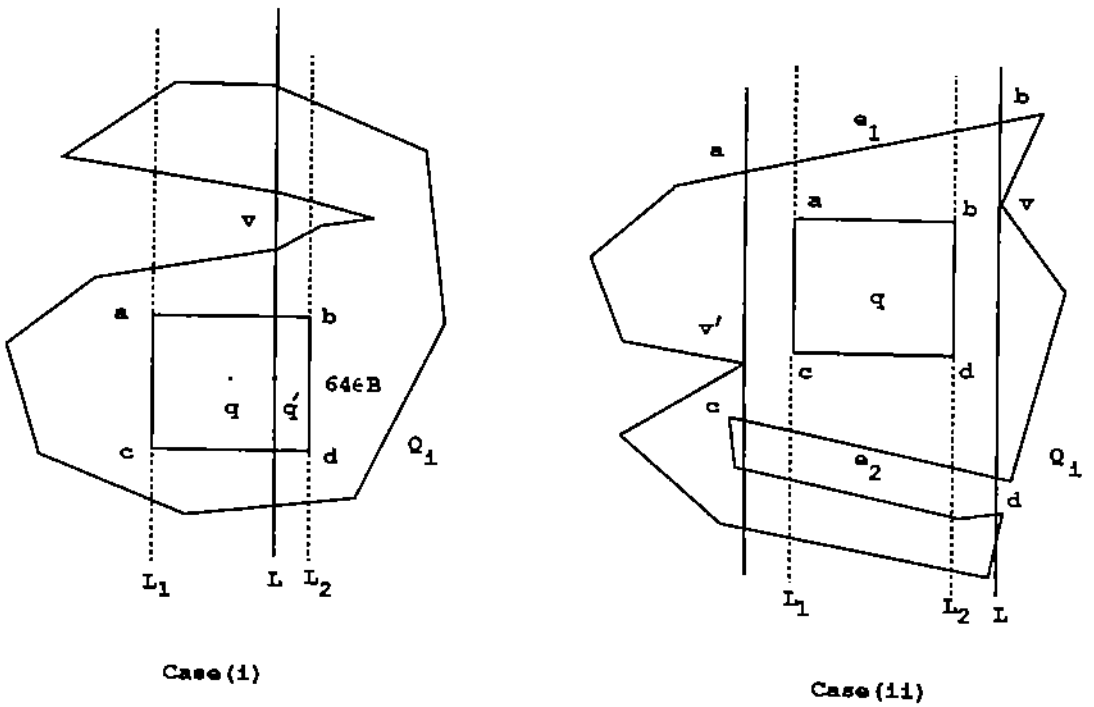


Figure 2.5 Cases of Lemma 2.4.4

Lemma 2.4.4 Given a set of simple, fleshy, nonintersecting polygons on a plane, there is a "good vertex" v of each polygon Q_i such that $ANSC(v)$ reports all true ancestors of Q_i.

Proof: Consider a simple, fleshy polygon Q_i. By definition, there is a point q inside Q_i such that a square box *abdc* with sides of length $64\epsilon B$ lies inside Q_i. Let q be the center of *abdc*. Consider two vertical lines L_1, L_2 coinciding with the two sides of the square as shown in Figure 2.5.

Case(i): There is a vertex v of Q_i within the two vertical lines. W.l.o.g., assume v to be above *ab*. Consider a vertical line L passing through v. Let q' be the orthogonal

projection of q on L. Let s be the point of intersection of L with the edge of Q_i which is vertically visible from q' and which is below cd. Any polygon containing Q_i cannot have an edge intersecting L in between v and s. Since the distance between v and s must be greater than or equal to $64\varepsilon B$, the computed distance between them must be at least $56\varepsilon B$. Hence, s cannot be in $I^i_{close(v)}$, and all the intersection points of Q_i that are in $I^i_{below(v)}$ must be at a computed distance of at least $56\varepsilon B$ from v. Certainly, $I^i_{below(v)}$ is odd. Hence, v is a "good vertex" of Q_i from below.

Case(ii): There is no vertex v which lies in between two vertical lines L_1 and L_2. In this case, only two edges of Q_i will be vertically visible from q. Let these two edges be e_1, e_2 as shown in Figure 2.5(b). Let v (respectively v') be the first vertex that is hit by a vertical line L while sweeping it from the position of L_2 (respectively, L_1) to right (respectively, left). Consider a vertical line through v that intersects e_1 and e_2 at b' and d' respectively. Similarly, consider the vertical line through v' that intersects e_1 and e_2 at a' and c' respectively. The quadrilateral $a'b'd'c'$ lies inside Q_i. Since $abdc$ lies inside $a'b'd'c'$, one of the edges $b'd'$ and $a'c'$ must be greater than or equal to $64\varepsilon B$. W.l.o.g., assume $b'd'$ is that edge. Certainly, v is at a distance of at least $32\varepsilon B$ either from b' or d'. W.l.o.g., assume the distance between v and d' is greater than equal to $32\varepsilon B$. This implies that the computed distance between v and d' is greater than $24\varepsilon B$. Following the same argument as in Case (i), we can say that v is a "good vertex" of Q_i from below.

Any polygon Q_j containing Q_i can not have an edge intersecting L in between v and s in Case(i) and in between v and d' in Case(ii). Hence, for such polygon Q_j, all intersection points in $I^j_{below(v)}$ must be at a computed distance of at least $24\varepsilon B$ from v and $|I^j_{below(v)}|$ must be odd. This ensures that $ANSC(v)$ reports all the true ancestors of Q_i. ♣

2.4.4 The Algorithm

Algorithm Polnest-Robust

Input : A set of simple, nonintersecting, fleshy polygons.

Output: An acyclic directed graph, called the nesting structure, in which each node n_i represents a polygon Q_i. There is a directed edge from n_i to n_j if and only if Q_j is the parent of Q_i.

Step 1: Sort the vertices of the polygons on the x axis.

Step 2: Sweep a vertical line from left to right taking the following steps at each vertex v.

Step 2(a): Let Q_i be the polygon having v on the boundary and E be the set of edges that were intersected by L when the sweep line stopped at the previous vertex. Compute the intersection points of L with edges in E. Construct the sets $I^j_{above(v)}, I^j_{close(v)}, I^j_{below(v)}$ for each polygon Q_j.

Step 2(b): Test whether v is a "good vertex" of Q_i or not. If it is, take step 2(c) otherwise skip 2(c).

Step 2(c): If v is a "good vertex" of Q_i from below (respectively above), for each polygon Q_j intersected by L, check whether $|I^j_{below(v)}|$ (respectively $|I^j_{above(v)}|$) is odd or not and whether all points in $I^j_{below(v)}$ (respectively $I^j_{above(v)}$) are at a distance of at least $24\epsilon B$ from v or not. If both conditions are satisfied, check the leftmost vertices of Q_j and Q_i to determine whether $NOT(CONT(Q_i, Q_j))$ is true or not. If it is true, create a directed edge from the node corresponding to Q_i to the node corresponding to Q_j in the nesting structure in case it is not already created. Note that this will create a directed edge from n_i to n_j if and only if Q_j is an ancestor (not merely parent) of Q_i. This nesting structure is refined in *Step 3*.

Step 2(d): If v is a vertex with both edges adjacent to it not in E, include them in E. If v is a vertex with both edges adjacent to it in E, delete them from E. If v is a vertex with one of the edges in E, delete that edge from E and include the other edge in E.

Step 3: The nesting structure G^* computed by *Step 2(c)* is the transitive closure of the actual nesting structure G of the set of polygons. G can be recovered from G^* in $O(e)$ time where e is the number of edges in G^*. Find all leaves in G^* i.e., the nodes that have an in-degree count equal to 0. These nodes are also leaves of G. Delete

all edges outgoing from these nodes. Find the new leaves in the modified G^*. These nodes are at a distance of one (w.r.t. the number of edges) from the leaves of G. Repeating this process, all nodes at a distance of one, two, three, ... from the leaves are found out and G is recovered. This algorithm can be carried out in $O(e)$ time where e is the number of edges in G^*.

Time Analysis: *Step 1* takes $O(n \log n)$ time. Since a vertical line intersects at most $O(m + r)$ edges (Lemma 2.2.1), *Step 2* takes $O(m + r)$ time for each stop while sweeping. Hence, the total time spent for *Step 2* is $O(n(m + r))$. *Step 3* takes $O(m^2)$ time since there are $O(m^2)$ edges in G^*. Thus, the time complexity of Polnest-Robust is $O(n \log n + n(m + r) + m^2) = O(n(\log n + m + r))$.

2.5 Conclusions

In this chapter, we have given an efficient algorithm for polygon nesting problem, where the polygons do not intersect along their boundaries. It is interesting to consider the case where polygons intersect along their boundaries. In that case, can we find the nesting structure in $O(n \log n)$ time or at least in $O(n \log n + s)$ time, where s is the total number of intersections between the polygons? This problem arises in pattern recognition during feature classifications.

We have devised a type-5 robust algorithm with a minimum feature assumption. It seems that some sort of minimum feature assumption is necessary to produce exact outputs under finite precision computations. There are applications, however, where an output "close" to the exact one is acceptable. In those cases, a type-5 robust algorithm without any minimum feature assumption is desirable.

3. CONVEX DECOMPOSITIONS

3.1 Introduction

This chapter deals with the convex decompositions of polyhedra. Convex decompositions, in terms of a finite union of disjoint convex pieces, are useful and are always possible for polyhedral models [Cha80, Ede87]. Convex decompositions lead to efficient algorithms, for example, in geometric point location and intersection detection; see [Ede87]. Specifically, a disjoint convex decomposition of simple polyhedra allows for more efficient algorithms in motion planning, in computer graphics, in solid modeling, and in the finite element solutions of partial differential equations.

The problem of partitioning a non-convex polyhedron S into a minimum number of convex parts is known to be NP-hard [Lin82, ORS83]. Rupert and Seidel [RS89] also show that the problem of determining whether a non-convex polyhedron can be partitioned into tetrahedra without introducing *Steiner* points is NP-hard. For a given polyhedron S with n edges of which r are reflex, Chazelle [Cha80, Cha84] established a worst case, $O(r^2)$ lower bound on the number of convex polyhedra needed for complete convex decomposition of S. He gave an algorithm that produces a worst case, optimal number $O(r^2)$ convex polyhedra in $O(nr^3)$ time and $O(nr^2)$ space. Recently, Chazelle and Palios [CP90] have given an $O((n+r^2)\log r)$ time and $O(n+r^2)$ space algorithm to tetrahedralize a subclass of non-convex polyhedra. The allowed polyhedra for their algorithm are all homeomorphic to a 2-sphere, i.e., have no holes(genus 0) and shells (internal voids).

In Section 3.3, we present an algorithm to compute a disjoint convex decomposition of a manifold polyhedron S which may have an arbitrary number of holes and shells. Given such a polyhedron S with n edges of which r are reflex, the algorithm produces a worst case optimal $O(r^2)$ number of convex polyhedra S_i, with $\bigcup_i S_i = S$

in $O(nr^2 + r^3 \log r)$ time and $O(nr + r^2\alpha(r))$ space. Here, α is the inverse Ackermann's function which grows extremely slowly. We extend this algorithm to work for non-manifold polyhedra which do not have abutting edges or facets but may have incidences as illustrated in Figure 3.1. The algorithm presented in this chapter is based on repeated cutting and splitting of polyhedra with planes that resolve reflex edges. Chazelle, in [Cha80], first used this method. We improve this method to obtain better time and space bounds using a refined complexity analysis and the efficient algorithms for certain subproblems.

In Section 3.4, we describe the geometric based heuristics that are used to overcome the inaccuracies involved with finite precision arithmetic computations. Although we cannot prove that these heuristics make the algorithm type-1 robust, the experimental results are very satisfying. This algorithm runs in approximately $O(nr^2 + nr \log n + r^4)$ time and $O(nr + r^2\alpha(r))$ space.

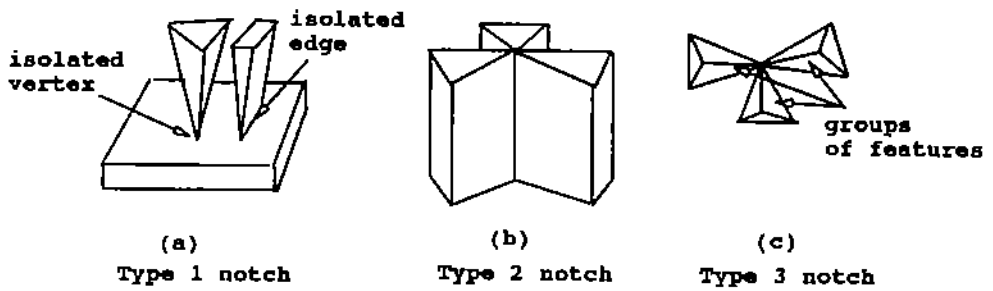


Figure 3.1 Non-manifold incidences or special notches.

3.2 Preliminaries

Manifold polyhedra can be nonconvex only due to the presence of reflex edges. Non-manifold polyhedra, however, can be nonconvex due to the features other than reflex edges. The features, causing nonconvexity in polyhedra are called *notches* in general.

3.2.1 Notches

Non-manifold polyhedra as considered in this thesis have the following four types of notches.

1. *Type 1 notches*: These notches are caused by isolated vertices and edges on a facet. An isolated vertex or an isolated edge on a facet is not adjacent to any other edge of the facet. See Figure 3.1(a).

2. *Type 2 notches*: These notches are caused by the edges along which more than two facets meet as illustrated in the Figure 3.1(b). If there are $2k$ ($k > 1$) facets incident on e_i, we assume that they form k notches.

3. *Type 3 notches*: These notches are caused by vertices where two or more groups of features (facets, edges) touch each other as illustrated in the Figure 3.1(c). The features within a group are reachable from one another while remaining only on the surface of S and not crossing the vertex. Actually, type 1 notches are a subclass of these notches. For convenience in the description, we exclude type 1 notches from type 3 notches. The number of groups attached to the vertex determines the number of type 3 notches associated with that vertex.

4. *Type 4 notches*: These notches are caused by reflex edges. A manifold polyhedron can have only this type of notches.

The notches of type 1, type 2, type 3 are called *special notches* as they are present only in non-manifold polyhedra. In our algorithm, we first remove all special notches from the input polyhedron S creating only manifold polyhedra. Subsequently, type 4 notches of the manifold polyhedra are removed by repeated cutting and splitting them with planes resolving the notches. Let an edge g with f_1, f_2 as its incident facets be a notch in a manifold polyhedron. A plane P_g that passes through g is called a *notch plane* if both angles (f_1, P_g) and (P_g, f_2), as measured from the inner side of f_1 and f_2 are not reflex. In other words, a notch plane resolves the reflex angle of a

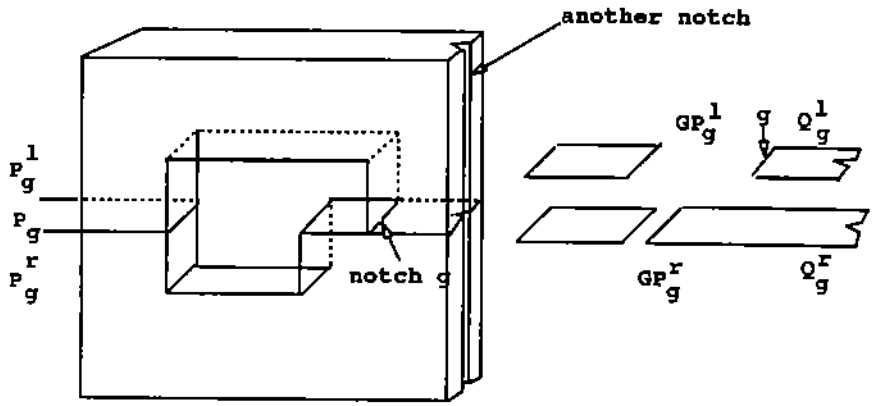


Figure 3.2 A notch and its notch plane, cross sectional map, cut.

notch. Clearly, for each notch g, there exist infinite choices for P_g. Note that P_g may intersect other notches, thereby producing *subnotches*; see Figure 3.2.

3.2.2 Data Structure

Let S be a polyhedron, possibly with holes and shells, and having s vertices : $\{v_1, v_2, ..., v_s\}$, n edges : $\{e_1, e_2, ..., e_n\}$, and q facets : $\{f_1, f_2, ..., f_q\}$. These lists of vertices, edges and facets of S are stored similar to the *star-edge* representation of polyhedra [Kar88].

Vertices: Each vertex is a record with two fields.

1. *vertex.coordinates*: contains the three dimensional coordinates of the vertex.

2. *vertex.adjacencies*: contains pointers to the edges incident on the vertex.

Edges: Each edge is a record with two fields.

1. *edge.vertices*: contains pointers to the incident vertices.

2. *edge.orientededges*: contains pointers to the record called *orientededges* which represent different orientations of an edge on each facet incident on it. The

orientation of an edge on a facet f is such that a traversal of the oriented edge has the facet f to its right.

Orientededges: each orientededge is a record with four fields.

1. *orientededge.edge*: contains pointer to the defining edge.

2. *orientededge.facet*: contains pointer to the facet on which the orientededge is incident.

3. *orientededge.orientation*: contains information about the orientation of the edge on the facet.

4. *orientededge.nextorientededge*: contains pointers (possibly more than one) to the next orientededges on the *oriented edge cycle* on a facet. See *facet cycles* below.

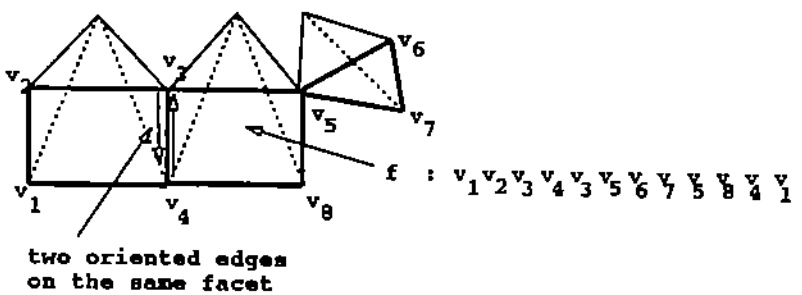


Figure 3.3 A non simple facet.

Facets: each facet is a record with two fields.

1. *facet.equation*: contains the equation of the plane supporting the facet.

2. *facet.cycles*: contains pointers to a collection of oriented edge cycles bounding the facet.Each oriented edge cycle is a linked list of orientededges. The traversal of each oriented edge on the cycle has the facet to its right. If there is an

isolated vertex on the facet, (Figure 3.1(a)) a pointer to the vertex is included in *facet.cycles* as a degenerate oriented edge cycle. An isolated edge is represented with the oriented edge cycle of two orientededges. For a non-manifold polyhedron, a facet may have configurations as shown in Figure 3.3, where a vertex or an edge is considered more than once in an oriented edge cycle, though an oriented edge is included only once.

3.2.3 Some Definitions

To deal with the non manifold polyhedra, we define the term polygon slightly differently in this chapter than the usual way. Let the *polygonal boundary* refer to an oriented edge cycle embedded on a plane with no edge intersecting another except at their end points. The traversal of a polygonal boundary may pass through an edge or a vertex more than once.

Definition 3.2.1 A polygon is a *connected* region on a plane that is bounded by one or more polygonal boundaries.

A polygon corresponding to the facet f is shown in Figure 3.3. Let Q be a polygon with vertices $v_1, v_2, ..., v_k$ in the clockwise order. The outer angle between two consecutive oriented edges d_{i-1} and d_i is measured in the anticlockwise direction from d_i to d_{i-1}.

Definition 3.2.2 A vertex is *reflex* in Q if the outer angle between the oriented edges $d_{i-1} = (v_{i-1}, v_i)$ and $d_i = (v_i, v_{i+1})$ is $\leq$ 180°. The vertices that are not reflex are called *normal vertices* of Q.

Notice that, with this definition, v_4, v_5 of the nonsimple facet in Figure 3.3 are reflex vertices, though v_3 is not.

Definition 3.2.3 A maximal piece of a polygonal boundary is called the *monotone chain* if its vertices have x-coordinates (or y-coordinates) in either strictly increasing or decreasing order, see Figure 3.4.

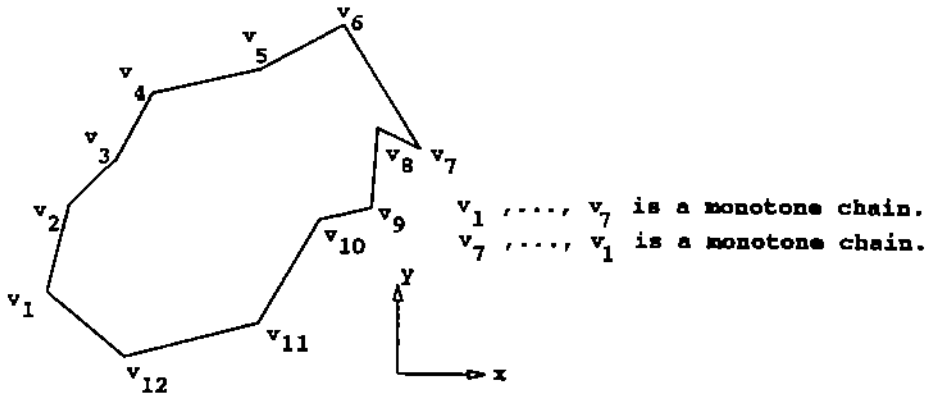


Figure 3.4 Monotone chains in a polygon.

3.2.4 Useful Lemmas

In the subsequent sections, we use the following lemmas.

Lemma 3.2.1 Let Q be a polygon with r reflex vertices. The number of monotone chains c in Q is bounded as $c \leq 6(1+r)$.

Proof: Follows from Lemma 2.2.2. ♣

Lemma 3.2.2 Let Q be a polygon with s normal vertices. There are at most $O(s)$ monotone chains in Q.

Proof: Let v be the vertex of Q with the minimum y-abscissa and let B be the boundary obtained by removing the vertex v and an *ϵ-ball* around v from the boundary of Q. Add six more edges to B as shown in Figure 3.5 to construct a new polygon Q'. The polygon Q' is oppositely oriented with respect to Q. Note that each reflex vertex of Q' corresponds to a normal vertex of Q. Thus, Q' has no more than s reflex vertices. According to Lemma 3.2.1, the boundary of Q' can be partitioned into $O(s)$ monotone chains. The polygon Q cannot have more monotone chains than Q' which implies that Q has $O(s)$ monotone chains.♣

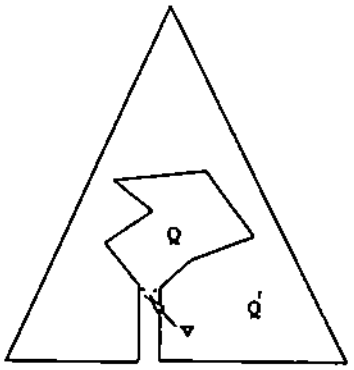


Figure 3.5 Constructing a polygon of opposite orientation.

Lemma 3.2.3 Let $\wp$ be a set of k polygons with r reflex vertices. No line can intersect $\wp$ in more than $r + k$ chords.

Proof: Follows immediately from Lemma 2.2.1.♣

Lemma 3.2.4 The problem of polygon nesting for a set of nonintersecting polygons can be solved in $O(s + t \log t)$ time assuming exact arithmetic computations where s is the total number of vertices, and t is the total number of monotone chains present in all input polygons.

Proof: Although the algorithm given in section 2.3 uses a slightly different type of monotone chains, called subchains, it also works for the monotone chains as defined in this chapter. Further, this algorithm can be straightforwardly adapted to the input set of polygons as defined in this chapter. ♣

3.3 The Algorithm with Exact Arithmetic

In this section, we develop and analyze a convex decomposition algorithm which assumes exact arithmetic computations. Given a polyhedron S, it is first split along vertices and edges of special notches to produce manifold polyhedra. Reflex edges

of a manifold polyhedron are removed by slicing it with notch planes. Notch planes may possibly intersect other notches to create subnotches. In general, the notch

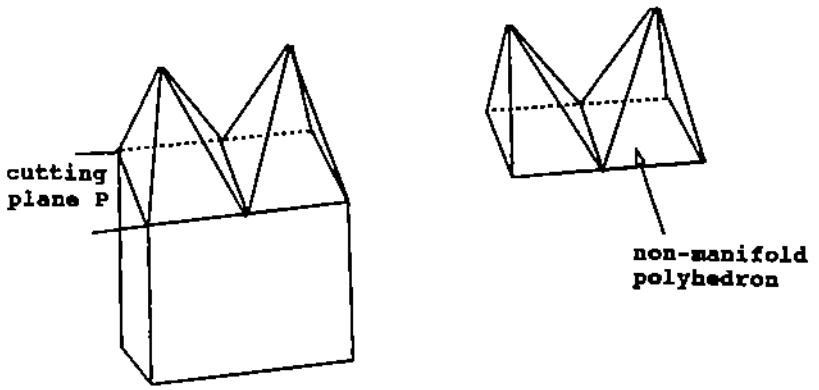


Figure 3.6 An example where manifold property is not preserved

elimination process produces a number of subpolyhedra. At a generic step of the algorithm, all subnotches of a notch, present in possibly different subpolyhedra, are eliminated with a single notch plane. Slicing a manifold polyhedron with a plane may produce non-manifold subpolyhedra with special notches. See Figure 3.6. As before, these non-manifold subpolyhedra are split along the special notches to produce only manifold polyhedra. If the notch plane, however, does not pass through a vertex of the polyhedron being cut, manifold property is preserved in the resulting subpolyhedra.

Algorithm ConvDecomp(S)

Step 1: Remove all special notches from S. This produces manifold polyhedra.

Step 2: Assign a notch plane for each notch in the manifold polyhedra produced in *Step* 1.

Step 3: repeat

Let $g_1, g_2, ..., g_k$ be the subnotches of a notch g
present in the polyhedra $S_1, S_2, ..., S_k$. Let P_g be

the notch plane assigned to g. Remove $g_1, g_2, ... g_k$
from $S_1, S_2, ..., S_k$ by the notch plane P_g.
Remove special notches produced by this slicing operation.
until all notches are eliminated.
end.

Step 1 of the algorithm is described in Section 3.3.2. Step 2 can be performed trivially in $O(r)$ time. The slicing step of the algorithm (Step 3) needs to be performed carefully and is detailed below in Section 3.3.1.

3.3.1 Intersecting a Manifold Polyhedron with a Notch Plane

Let S be a manifold polyhedron with r notches and p edges. By S, we denote here any polyhedron $S_1, S_2, ..., S_k$ that is encountered in step 3 of the above algorithm *ConvDecomp*. The notch plane P_g: $ax+by+cz+d = 0$ defines two closed half spaces $P_g^\ell : ax + by + cz + d \geq 0$ and $P_g^r : ax + by + cz + d \leq 0$. To cut a polyhedron S with the plane P_g, it is essential to compute

$$S^\ell = cl(int(P_g^\ell) \cap int(S))$$
$$S^r = cl(int(P_g^r) \cap int(S))$$

where $cl(O)$ and $int(O)$ denote the closure and interior of the geometric object O. Since polyhedra are represented with their boundaries, we need to compute the boundaries δS^ℓ, δS^r of S^ℓ and S^r respectively. To compute δS^ℓ and δS^r, it is essential to compute the features of δS^ℓ and δS^r lying on P_g.

Definition 3.3.1 The intersection of P_g with δS^ℓ and δS^r are called the *cross sectional maps* and are denoted as GP_g^ℓ and GP_g^r respectively.

Note that for a polyhedron S, and a plane P_g, the cross sectional maps GP_g^ℓ and GP_g^r may be different. See for example, Figure 3.2. In general, GP_g^ℓ and GP_g^r consist of a set of isolated points, segments and polygons, possibly with holes.

Definition 3.3.2 The unique polygons Q_g^ℓ, Q_g^r on GP_g^ℓ and GP_g^r respectively, containing the notch g on their boundary are called *cuts*.

Note that, to remove a notch g, it is sufficient to slice S along only the cut instead of the entire cross sectional map.

Instead of computing Q_g^ℓ, Q_g^r separately, we first compute the cut $Q_g = Q_g^\ell \cup Q_g^r$ and then refine it to obtain Q_g^ℓ and Q_g^r. This calls for computing the cross sectional map $GP_g = GP_g^\ell \cup GP_g^r$. The polygon corresponding to the cut Q_g may have a vertex or an edge appearing more than once while traversing its boundary. If an edge appears more than once in traversing the boundary of Q_g^ℓ or Q_g^r, the edge must make the corresponding subpolyhedron non-manifold; see Figure 3.6. It is interesting to observe that there can be at most four facets incident upon that edge since the original polyhedron being sliced was a manifold.

An additional fact is that a single slicing along the cut may not separate the polyhedron S into two different pieces. See Figure 3.2. In this case, two facets corresponding to Q_g^ℓ and Q_g^r are created that may overlap geometrically and considered distinct, so that the polyhedron is treated as manifold polyhedron.

The algorithm to cut a polyhedron S with a notch plane P_g consists of two basic steps.

- *Step I*: Computing the cut Q_g: This calls for computing inner (holes) and outer boundaries of the polygon Q_g.
- *Step II*: Splitting the polyhedron S.

Step I is detailed below in Section 3.3.1.1 and step II in Section 3.3.1.2.

3.3.1.1 Computation of the cut Q_g

Step A: First, all boundaries present in the cross sectional map GP_g are computed. To do that, all the facets of S are visited in turn. If the notch plane intersects a facet f, all intersection points are computed. Note that f must be a simple facet (no vertex

or edge is traversed twice along its boundaries) since S is a manifold polyhedron. Let $a_1, a_2, ..., a_k$ be the sorted sequence of intersection points along the line of intersection $P_g \cap f$. We call an intersection point a *new intersection vertex* if it does not coincide with any vertex of the facet f and call it an *old intersection vertex*, otherwise. It is essential to decide consistently whether there should be an edge between two consecutive intersection vertices a_i and a_{i+1} of this sorted sequence. This is done by scanning the vertices in sorted order and deciding whether we are "inside" or "outside" the facet as we leave a vertex to go to the next one. If a_i is a new intersection vertex, there can be an edge between a_i and a_{i+1} only if there is no edge between a_{i-1} and a_i and vice versa. On the other hand, if a_i is an old intersection vertex, there can be an edge between a_i and a_{i+1} irrespective of the presence of an edge between a_{i-1} and a_i.

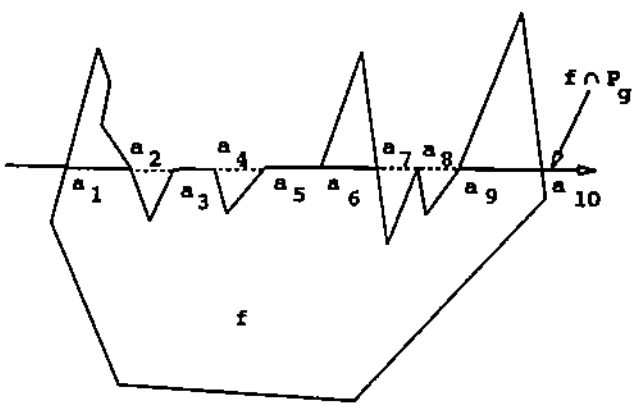


Figure 3.7 Generating new and old edges.

Switching between "inside" and "outside" of the facet is carried out properly, even with degeneracies, using a *multiplicity code* at each intersection vertex. During the scan of the sorted sequence of intersection vertices, a counter is maintained. The counter is initialized to zero and is incremented by the multiplicity code at each vertex. Our status toggles between "inside" and "outside" of the facet as the counter toggles between the "odd" and "even" count. A new intersection vertex is assigned a

multiplicity code of 1. An old intersection vertex has a multiplicity code of 1 if both of its incident oriented edges on the facet f do not lie in the same half-space of P_g and a multiplicity code of 2, otherwise. If there is an old edge (edge of f) between two vertices a_i and a_{i+1}, multiplicity codes are assigned to them as follows. If other two incident oriented edges on a_i, a_{i+1} on the facet f lie in the same open half-space of the notch plane, assign a multiplicity code of 1 to both of them. Otherwise, assign multiplicity codes of 1 and 2 to a_i and a_{i+1} in any order. In Figure 3.7, there is an old edge between a_3, a_4. The status ("outside") with which one enters the vertex a_3 is same as the one with which one leaves the vertex a_4. This is enforced by assigning a multiplicity code of 1 on the two vertices that increments the counter by an "even" amount and prevents it from toggling. In the same example, there is another old edge between a_5 and a_6. The status ("outside") with which one enters the vertex a_5 is different from the one with which one leaves the vertex a_6. This is enforced by assigning multiplicity codes of 1 and 2 on the two vertices in any order which increment the counter by an "odd" amount and make it toggle. A new edge from vertex a_i to a_{i+1} is created if the count is "odd" on leaving vertex a_i. In case, there is an old edge between a_i and a_{i+1}, no new edge is created between them. This process is repeated for all facets intersected by P_g resulting eventually in creating the *1-skeleton* or the underlying graph of GP_g. This underlying graph becomes a directed graph if the oriented edges associated with the edges in GP_g are considered. Orientation of each such edge is determined in constant time since the orientations of the facets intersecting the notch plane are known. A traversal in a depth-first manner in this directed graph traces the boundaries of GP_g.

Timing Analysis: According to Lemma 2.2.1, the notch plane P_g intersects a facet f of S in at most $2r_i + 2$ points where r_i is the number of reflex vertices in f. Thus, sorting of the intersection points on a facet takes at most $O(u_i \log r_i)$ time where u_i is the number of intersection points on the facet. Considering all such facets, we obtain the sorted sequence of intersection vertices on the facets computed in $O(p + u \log r)$

time where u is the number of vertices in GP_g. Generating the edges between these intersection vertices takes no more than $O(p)$ time altogether. The time taken for tracing the boundaries of GP_g is linear in the number of edges in GP_g. Overall, the computation of GP_g takes $O(p + u \log r)$ time.

Step B: Next, the inner and outer boundaries of Q_g are determined from GP_g. It is trivial to determine the boundary B_g containing the notch g. One can determine whether B_g is an inner or outer boundary of Q_g by checking the orientations of the edges on the boundary.

Case(i): B_g is an outer boundary of Q_g: Let I_i be the polygon corresponding to an inner boundary (hole) of Q_g. The polygon I_i has at least one vertex which is normal. Since the boundary of I_i constitute an inner boundary of Q_g, the normal vertices of I_i are reflex vertices of Q_g. Definitely, reflex vertices of Q_g lie on notches of S. This implies that all inner boundaries of Q_g will have a vertex where P_g intersects a notch of S. The set W of boundaries having at least one such vertex is determined. The boundaries in the set $W \cup B_g$ are called *interesting boundaries*. The polygon nesting algorithm applied on the polygons constituted by the interesting boundaries detects the children of B_g. The boundaries of these children constitute the inner boundaries of Q_g.

Timing Analysis: The set W can be created in $O(u)$ time where u is the number of vertices present in the cross sectional map. Certainly, the number of interesting boundaries is $O(t)$ where t is the number of notches intersected by the notch plane P_g. The interesting boundaries. that are outer boundaries of some polygon in the cross sectional map, have $O(t)$ reflex vertices since these vertices are generated by the intersection of a notch of S with the notch plane. On the other hand, the interesting boundaries that are inner boundaries of some polygon in the cross sectional map have $O(t)$ normal vertices. Thus, according to Lemma 3.2.1 and 3.2.2, there are at most $O(t)$ monotone chains in the interesting boundaries. If there are u' vertices in the

interesting boundaries, the children of B_g can be determined in $O(u' + t \log t)$ time using the polygon nesting algorithm (Lemma 3.2.4). Thus, in this case, the inner and outer boundaries of Q_g can be detected in $O(u + u' + t \log t) = O(p + t \log t)$ time, since $u = O(u') = O(p)$.

Case(ii): B_g is an inner boundary of Q_g: The boundaries that completely contain the boundary B_g inside are determined. This can be done by checking the containment of any point on B_g with respect to all boundaries in the cross sectional map. These boundaries, together with B_g, are the interesting boundaries. The polygon nesting algorithm, applied on these interesting boundaries, detects the boundaries of the parent polygon of B_g. This boundary is the outer boundary of Q_g. Note that Q_g may have other inner boundaries different from B_g. Once the outer boundary of Q_g is computed, all of its inner boundaries can be obtained applying the technique used in case(i).

Timing Analysis: Detection of all boundaries containing B_g takes $O(u)$ time. The set of interesting boundaries can be partitioned into two classes according to whether they are inner or outer boundaries of some polygon. It is not hard to see that there can be at most one more outer boundary than inner boundaries in this set. Hence, the number of interesting boundaries is of the order of inner boundaries present in the cross sectional map. As discussed in case(i), the number of inner boundaries must be bounded above by the number of notches intersected by the notch plane. Thus, there are $O(t)$ interesting boundaries. Further, as explained before, the number of monotone chains present in these interesting boundaries can be at most $O(t)$. Hence, the outer boundary of Q_g can be determined in $O(p + t \log t)$ time. Detection of other inner boundaries that are different from B_g takes another $O(p + t \log t)$ time. Thus, in this case also all outer and inner boundaries of Q_g can be detected in $O(p + t \log t)$ time.

Combining all these costs together, we see that the "cut computation" takes $O(p + t \log t + u \log r)$ time.

3.3.1.2 Splitting S

Separation of S along the cut Q_g is carried out by splitting facets that are intersected by Q_g. Suppose f is such a facet which is to be split at $a_1, a_2, \ldots a_k$. The splitting of f consists of splitting the old intersection vertices and the edges on which a new intersection vertex lies. For this splitting operation, the intersection vertices on each facet f are visited, and for each such intersection vertex, constant time is spent for setting the relevant pointers. The facet f may be split into several subfacets. The inner boundaries of f that are not intersected by P_g remains as inner boundaries of some of these subfacets. The polygon nesting algorithm determines the inclusions of these inner boundaries into proper subfacets. The cut Q_g is refined to yield Q_g^ℓ and Q_g^r. It is observed that the differences between Q_g^ℓ and Q_g^r are caused by the edges of S that lie completely on P_g. Hence, to refine Q_g, one needs to determine which of the edges of S are to be transferred to Q_g^ℓ (Q_g^r respectively). This can be done using the following simple rule. An old edge e must be transferred to Q_g^ℓ (Q_g^r respectively.) if any facet (or a part of it) that is adjacent to e and not coplanar with P_g lies in P_g^ℓ (P_g^r respectively). A copy of Q_g is created, and one of the two Q_g's is designated for Q_g^ℓ and another for Q_g^r. From a copy, all those edges that are not to be transferred to it are deleted. Note that the transfer of edges lying on Q_g takes care of the facets lying on Q_g. Two oppositely oriented facets at the same geometric location corresponding to the cuts Q_g^ℓ and Q_g^r are created. All modified incidences are adjusted properly. A depth first traversal in the modified vertex list either completes the separation of S by collecting all the pertinent features of each piece or reveals the fact that S is not separated into two different pieces by the cut. In the latter case, either the number of holes or the number of shells in S is reduced by one.

Timing Analysis: Adjustment of all incidences in the internal structure of S cannot take more than $O(p)$ time since each edge is visited only $O(1)$ times. The polygon nesting takes $O(p + r \log r)$ time since there can be at most $O(r)$ holes in the facets of S containing $O(r)$ monotone chains. Further, creation of Q_g^ℓ and Q_g^r from Q_g and the depth first traversal in the modified vertex list cannot exceed $O(p)$ time. Hence, the "splitting operation" takes $O(p + r \log r)$ time.

3.3.2 Elimination of Special Notches and its Analysis

For a non-manifold polyhedron S, nonconvexity results from four types of notches as discussed in Section 3.2.1. Let S have n edges and r notches. The counting of special notches is described in Section 3.2.1. A preprocessing is carried out as follows to remove the notches of the first three types, called *special notches*.

Removal of type 1 notches: As can be observed from Figure 3.1(a), the vertex or the edge causing the nonconvexity is detached from the facet on which it is incident as an isolated vertex or an isolated edge. Identifying these vertices and edges and detaching them from the corresponding facets take at most $O(n)$ time.

Removal of type 2 notches: Here, more than two facets are incident on an edge e_i. Let these facets be $f_1, f_2, ..., f_{r_i}$. Let C be a cross section obtained as the intersection of the facets incident on e_i with the plane P that is normal to the edge e_i. C consists of edges $e_j = (f_j \cap P)$. The facets around e_i are sorted circularly by a simple circular sort of the edges e_j's around $e_i \cap P$. The adjacent facets that enclose a volume of S are paired. Let this pairing be $(f_1, f_2), (f_3, f_4), ..., (f_{r_i-1}, f_{r_i})$. An edge between each pair of facets is created, and the edge e_i is deleted. All these edges are at the same geometric location of e_i. All incidences are adjusted properly. Sorting of the facets around the edge e_i takes $O(r_i \log r_i)$ time. Further, for all type 2 notches, the adjustment time of all incidences in the internal representation of S cannot exceed $O(n)$. Thus, removal of all type 2 notches takes at most $(n + r \log r)$ time.

Removal of type 3 notches: Let v be a vertex that corresponds to a type 3 notch. In this case, we group together all features (edges and facets) that are incident on v, and are reachable from one another while remaining always on the surface of S and never crossing v. This gives a partition of the features incident on v into smaller groups. For each such group, a vertex at the same geometric location of v is created and all incidences are adjusted properly. This, in effect, removes the nonconvexity caused by v. All such vertices causing type 3 notches in S can be identified in $O(n)$ time by edge-facet-edge traversal in the internal data structure of S. Removal of all such notches takes at most $O(n)$ time. This is due to the fact that each edge can be adjacent to at most two type 3 notches and thus is visited only $O(1)$ times. Thus, all type 3 notches can be removed in $O(n)$ time.

Finally, a mixture of cases may occur where an isolated vertex is also a type 3 notch or an isolated edge is also a type 2 notch. All these cases are handled by first eliminating all type 1 notches and then eliminating type 3 notches followed by type 2 notches.

Removal of all the above notches generates at most $O(n)$ new edges and produces at most k manifold polyhedra where k is the number of special notches in S.

3.3.3 Worst Case Complexity Analysis

Combining the costs of the "cut computation" of Section 3.3.1.1 and the "splitting operation" of Section 3.3.1.2 yields the following lemma.

Lemma 3.3.1 A manifold polyhedron S having p edges can be partitioned with a notch plane P_g of a notch g in $O(p + t \log t + (u + r) \log r)$ time and in $O(p)$ space where t is the number of notches intersected by P_g, and u is the number of vertices in GP_g.

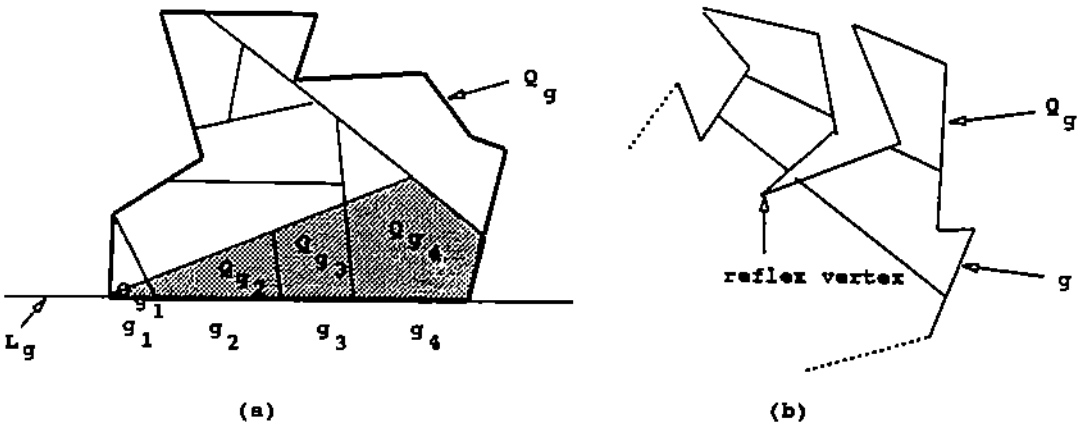


Figure 3.8 Superimposing a cut on the arrangement of notch line segments.

Let $S_1, S_2, ..., S_k$ be the polyhedra in the current decomposition where each S_i contains a subnotch g_i of a notch g of a manifold polyhedron S with n edges and r notches. Let m_i and u_i be the number of edges and vertices in Q_{g_i} respectively.

Lemma 3.3.2 The total number of edges and vertices in all cuts supported by the subnotches of a notch g are given as $m = \sum_{i=1}^{k} m_i = O(n + r\alpha(r))$ and $u = \sum_{i=1}^{k} u_i = O(n + r\alpha(r))$.

Proof: Consider the cut Q_g produced by the intersection of S with P_g. The region in Q_g is divided into smaller facets by *notch line segments* produced by the intersection of other notch planes with P_g. We focus on the facets $Q_{g_1}, Q_{g_2}, ..., Q_{g_k}$ adjacent to the subnotches $g_1, g_2, ..., g_k$ of the notch g.

Consider the set of notch line segments that divides Q_g. These lines and the line L_g corresponding to the notch g produce an arrangement of line segments on the notch plane P_g. The facets adjacent to the line L_g in this arrangement form the *zone* Z_g of L_g. Let the set of vertices and edges of Z_g be denoted as V_g and E_g respectively. It is known that $|V_g| = O(l\alpha(l))$ and $|E_g| = O(l\alpha(l))$ if there are l line segments in the arrangement; see [EGP+88]. Overlaying Q_g on Z_g produces $Q_{g_1}, Q_{g_2}, ..., Q_{g_k}$;

see Figure 3.8(a). Let V'_g and E'_g denote the sets of vertices and edges respectively in $Q_{g_1}, Q_{g_2}, ..., Q_{g_k}$. The vertices in V'_g can be partitioned into three disjoint sets, namely, T_1, T_2, T_3. The set T_1 consists of vertices formed by the intersections of two notch line segments; T_2 consists of vertices of Q_g, and T_3 consists of vertices formed by the intersections of the notch line segments with the edges of Q_g. Certainly, $|T_1| \leq |V_g| = O(l\alpha(l))$ since overlaying Q_g on Z_g cannot introduce more vertices in T_1. If Q_g has u' vertices, $|T_2| \leq u'$.

To count the number of vertices in T_3, we first assume that Q_g does not have any hole. Consider an edge e in E_g that contributes one or more edge segments to E'_g as a result of intersections with Q_g. There must be at least one reflex vertex of Q_g present between two successive edge segments of e. Charge a unit cost to the reflex vertex that lies to the left (or, right) of each segment and charge a unit cost to e itself for the leftmost (or, rightmost) segment. We claim that each reflex vertex of Q_g is charged at most once by this method. Suppose, on the contrary, a reflex vertex is charged twice by this procedure. That reflex vertex must appear between two segments of two edges in E_g as shown in Figure 3.8(b). As can be easily observed, all four edge segments cannot be adjacent to the regions incident on the edge g of Q_g. This contradicts the assumption that all these four edge segments are present in E'_g. Hence, the total charge incurred upon the reflex vertices of Q_g and the edges of E_g can be at most $O(r_g + l\alpha(l))$ where r_g is the number of reflex vertices present in Q_g. This implies that as a result of intersections with Q_g, at most $O(r_g + l\alpha(l))$ segments of edges in E_g are contributed to E'_g. Hence, $|T_3| = O(r_g + l\alpha(l))$.

Consider next the case where Q_g has holes. We refer to the polygon corresponding to a hole in Q_g as *hole-polygon*. From Q_g create a polygon Q'_g that does not have any hole merging all polygons into a single polygon as follows. Let H_1 and H_2 be two hole-polygons that have at least two visible vertices v_1, v_2 i.e., the line segment joining v_1, v_2 does not intersect any other edge. Split v_1 and v_2 and join them with the line segments as shown in Figure 3.9 to merge H_1, H_2. Repeat this process successively for all hole polygons until they are merged into a single polygon. Finally, connect the

boundary of this new polygon to the outer boundary of Q_g to create Q'_g. Consider superimposing Q'_g on Z_g. Let T'_3 denote the set of vertices formed by the intersection of edges of E_g and those of Q'_g. The distance between split vertices of Q'_g can be kept arbitrarily small to preserve all intersections between the edges of Q_g and those of Z_g. This ensures that $|T_3| \leq |T'_3|$. The polygon Q'_g has at most $O(u')$ vertices since the original polygon Q_g had u' vertices, and at most $O(u')$ extra vertices are added to form Q'_g from Q_g. Furthermore, the polygon Q'_g can have at most $O(u')$ reflex vertices. Applying the previous argument on the superimposition of Q'_g on Z_g we get $|T_3| \leq |T'_3| = O(u' + l\alpha(l))$.

Putting all these together, we have $|V'_g| = |T_1| + |T_2| + |T_3| = O(r_g + l\alpha(l) + u')$. Since there can be at most r notch planes, $l \leq r$. Certainly, $r_g \leq r$ and $u' \leq n$. This gives $u = |V'_g| = O(n + r\alpha(r))$. Since $Q_{g_1}, Q_{g_2}, ..., Q_{g_k}$ form a planar graph, we have $m = |E'_g| = O(|V'_g|) = O(n + r\alpha(r))$.♣

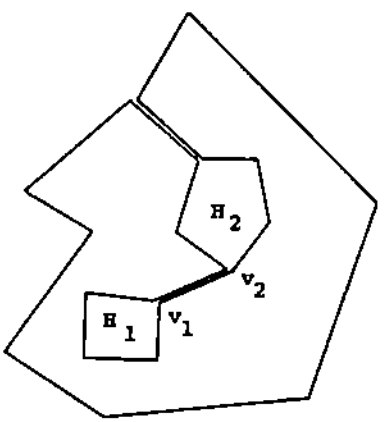


Figure 3.9 Merging polygons to create Q'_g from Q_g

Lemma 3.3.3 The total number of edges in the final decomposition of a polyhedron S with r notches and n edges is $O(nr + r^2\alpha(r))$.

Proof: Edges in the final decomposition consist of newly generated edges by the cuts, and the edges of S that are not intersected by any notch plane. By Lemma 3.3.2, the total number of edges present in all cuts corresponding to the subnotches of a notch is $O(n + r\alpha(r))$. This implies that each notch plane generates $O(n + r\alpha(r))$ new edges. Thus, r notch planes generate $O(nr + r^2\alpha(r))$ new edges. Hence, the total number of edges in the final decomposition is $O(n + nr + r^2\alpha(r)) = O(nr + r^2\alpha(r))$.♣

Lemma 3.3.4 Let $S_1, S_2, ..., S_k$ be the polyhedra in the current decomposition where each S_i contains a subnotch g_i of a notch g. Let u_i be the total number of vertices in the cross sectional map in S_i. Then we have $u = \sum_{i=1}^{k} u_i = O(n + r^2)$, where u is the total number of vertices in the cross sectional maps in $S_1, S_2, ..., S_k$.

Proof: Consider the cross sectional map GP_g. The lines of intersection between P_g and other notch planes, called the notch lines, divide this map into smaller facets. These facets are present in the cross sectional maps in $S_1, S_2, ..., S_k$, i.e., in $\cup_{i=1}^{k} GP_{g_i}$. The vertices in $\cup_{i=1}^{k} GP_{g_i}$ can be partitioned into three sets, viz., T_1, T_2 and T_3. The set T_1 consists of vertices that are created by the intersections two notch lines. The set T_2 consists of vertices of GP_g and the set T_3 consists of vertices that are created by the intersections of edges of GP_g and notch lines . Since there are at most r notch lines, $|T_1| = O(r^2)$. Certainly, $|T_2| = O(n)$. By Lemma 3.2.3, each notch line can intersect GP_g in at most $O(r)$ chords since GP_g can have at most r polygons containing no more than r reflex vertices all together. This gives $|T_3| = O(r^2)$. Thus,

$$\begin{aligned} u = \sum_{i=1}^{k} u_i &= |T_1| + |T_2| + |T_3| \\ &= O(n + r^2).\clubsuit \end{aligned}$$

As discussed in [Cha84], one can always produce a worst case optimal number ($O(r^2)$) of convex polyhedra by carefully choosing the notch planes.

Lemma 3.3.5 A manifold polyhedron S with r notches can be decomposed into $\frac{r^2}{2} + \frac{r}{2} + 1$ convex pieces if all subnotches of a notch are eliminated by a single notch plane.

Further, this convex decomposition is worst-case optimal since there exists a class of polyhedra that cannot be decomposed into fewer than $O(r^2)$ convex pieces.

Proof: See [Cha84].♣

Theorem 3.3.1 A manifold polyhedron S, possibly with holes and shells and having r notches and n edges, can be decomposed into $O(r^2)$ convex polyhedra in $O(nr^2 + r^3 \log r)$ time and $O(nr + r^2\alpha(r))$ space.

Proof: Decomposition of a polyhedron consists of a sequence of cuts through the notches of S as illustrated in the algorithm *ConvDecomp*. Step 1 assigns a notch plane for each notch in S in $O(r)$ time. According to Lemma 3.3.5, *ConvDecomp* produces worst case optimal $O(r^2)$ convex pieces at the end since all subnotches of a notch are removed by a single notch plane. Note that all holes and shells are removed automatically by the notch elimination process.

At a generic instance of the algorithm, let $S_1, S_2, ..., S_k$ be k distinct (nonconvex) polyhedra in the current decomposition where each S_i contains a subnotch g_i of a notch g that is going to be removed. Let S_i have m_i edges of which r_i are notches. Let t_i be the number of notches intersected by P_g in S_i and $t = \sum_{i=1}^{k} t_i$, u_i be the number of vertices in GP_{g_i} of S_i and $u = \sum_{i=1}^{k} u_i$.

Applying Lemma 3.3.1, removal of a notch g can be carried out in $O(\sum_{i=1}^{k}(m_i + t_i \log t_i + (u_i + r_i) \log r_i))$ time. Since $m = \sum_{i=1}^{k} m_i = O(nr + r^2\alpha(r))$, $\sum_{i=1}^{k} r_i = O(r^2)$, $u = O(n + r^2)$, and a notch plane can intersect at most $r - 1$ notches giving $t = O(r)$, we have $O(\sum_{i=1}^{k}(m_i + t_i \log t_i + (u_i + r_i) \log r_i)) = O(nr + r^2\alpha(r) + r^2 \log r)$.

As described before, elimination of a notch may produce non-manifold polyhedra having special notches. To remove them, the same method is used for eliminating special notches as used for the original polyhedron. Note that the type 2 notches in these non-manifold polyhedra can be adjacent to at most four facets. Hence, no logarithmic factor appears in the time complexity of removing such notches. This implies that the elimination of special notches from the non-manifold polyhedra produced as

a result of cutting manifold polyhedra with notch planes can be carried out in totally $O(m) = O(nr + r^2\alpha(r))$ time.

Thus, each notch elimination step takes $O(nr + r^2 \log r)$ time and Step 3 of *ConvDecomp* which eliminates r notches takes $O(nr^2 + r^3 \log r)$ time. Combining the complexities of Step 2 and Step 3, we obtain an $O(nr^2 + r^3 \log r)$ time complexity for the convex decomposition of a manifold polyhedron. The space complexity of $O(nr + r^2\alpha(r))$ follows from Lemma 3.3.3. ♣

Theorem 3.3.2 A non-manifold polyhedron S, possibly with holes and shells and having r notches and n edges, can be decomposed into $O(r^2)$ convex polyhedra in $O(nr^2 + r^3 \log r)$ time and $O(nr + r^2\alpha(r))$ space.

Proof: Removal of all special notches from S is carried out in $O(n + r \log r)$ time and in $O(n)$ space as discussed before. Let $S_1, S_2, ..., S_l$ be the manifold polyhedra created by this process. Let S_i have n_i edges of which r_i are reflex. Using Theorem 3.3.1 on each of them, we conclude that S can be decomposed into $O(r^2)$ convex polyhedra in $O(\sum_{i=1}^{l} n_i r_i^2 + r_i^3 \log r_i) = O(nr^2 + r^3 \log r)$ time and in $O(\sum_{i=1}^{l} n_i r_i + r_i^2\alpha(r_i)) = O(nr + r^2\alpha(r))$ space.♣

3.4 Robustness under Finite Precision Arithmetic

In this section, we describe the heuristics used in attempt to make the convex decomposition algorithm type-1 robust. It is clear from the discussion of our algorithm in Section 3.3 that numerical computations arise in various intersections and incidence tests. Under the ϵ-arithmetic model, the absolute error in the distance computations of one polyhedral feature from another is bounded by a certain quantity $\delta = k\epsilon B$ where B is the maximum value of any coordinate, and k is a constant; see for example [Mil88]. When making decisions about the incidences of these polyhedral features (vertices, edges, facets) on the basis of the computed distances(with signs), one can rely on the sign of the computations only if the distances are greater than δ. On the

other hand, if the computed distances are less than δ, one also needs to consider the topological constraints of the geometric configuration to decide on a reliable choice. In particular, in regions of uncertainity, i.e., within the δ-ball, the choices are all equally likely that the computed quantity is negative, zero, or positive. Such decision points of uncertainity where several choices exist, are either "independent" or "dependent". At independent decision points, any choice may be made from the finite set of local topological possibilities, while the choice at dependent decision points should ensure that it does not contradict any previous topological decisions. We follow this paradigm to make our convex decomposition algorithm to be type-1 robust. Unfortunately, we cannot guarantee that it is possible to follow this paradigm throughout the algorithm.

3.4.1 Intersection & Incidence Tests

In what follows, we assume the input polyhedra to be manifold. Non-manifold polyhedra can be handled as discussed in the earlier sections. We assume minimum feature criteria for the input polyhedra wherein the distance between two distinct vertices or between a vertex and an edge is at least δ. To decide whether an edge is intersected by a plane, one must decide the classification of its terminal vertices with respect to the same plane. The same classification of a vertex is used to decide the classification of all the features incident on that vertex. This, in effect, avoids conflicting decisions about the polyhedral features. The decisions about different types of intersections and incidence tests are carried out using three basic tools, namely, (i) vertex-plane classifications, (ii) facet-plane classifications, and (iii) edge-plane classifications. The order of classifications is (i) followed by (ii) followed by (iii). In what follows, we assume that the equation of any plane $P_i : a_i x + b_i y + c_i z + d_i$ is normalized, i.e., $a_i^2 + b_i^2 + c_i^2 = 1$.

3.4.1.1 Vertex-Plane Classification

To classify the incidence of a vertex $v_i = (x_i, y_i, z_i)$ w.r.t. the plane $P : ax + by + cz + d = 0$, the normalized algebraic distance of v_i from P is computed which

is given by $ax_i + by_i + cz_i + d$. The *sign* of this computation, viz., zero, negative, or positive, classifies v_i as "on" P (zero), "below" P (negative) or "above" P (positive), where "above" is the open half space containing the plane normal (a, b, c). The sign of the computation is accepted as correct if the above distance of v_i from P is larger than δ. Otherwise, geometric reasoning is applied, as detailed below, to classify the vertex v_i w.r.t. the plane P. In the following algorithmic version of the vertex-plane classification, the intersection between an edge e_j incident on v_i and the plane P is computed as follows. Let e_j be incident on planes P_1, P_2, where $P_i : a_i x + b_i y + c_i z + d_i = 0$. The intersection point r of e_j and the plane P is determined by solving the linear system, $Ar = d$ where $A = \begin{bmatrix} a & b & c \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix}$ $d = [-d, -d_1, -d_2,]^T$. The linear system is solved using Gaussian elimination with scaled partial pivoting and iterative refinement to reduce the numerical errors.

Vertex-Plane-Classif (v_i,P)

begin

Let $v_i = (x_i, y_i, z_i)$ be a vertex incident on edges

$e_1 = (v_i, w_1), e_2 = (v_i, w_2), ..., e_k = (v_i, w_k)$.

Let $P : ax + by + cz + d = 0$.

Compute $l = ax_i + by_i + cz_i + d$.

if $|l| > \delta$ *then* (**Comment*: unambiguously decide via the sign of the distance computation*)

if $l > 0$ *then*

classify v_i as "above"

else

classify v_i as "below"

endif

else

loop

(**Comment*: if the distance computation does not yield an unambiguous classification for the vertex with respect to the plane, ensure that the "above", "below" classification is consistent with all edges incident on that vertex. If such consistency cannot be ensured then the vertex is classified as "maybeon" and left for the future $facet - plane$ classifications to decide its classification consistently.*)

Search for an edge e_j incident on v_i such that $r = e_j \cap P$ is at a distance greater than δ from v_i and $w_j = (x_j, y_j, z_j)$.
Get the classification of w_j if it is already computed.
Otherwise, compute $l' = ax_j + by_j + cz_j$.
if $|l'| > \delta$ then classify w_j accordingly.
if the classification of w_j is "below" or "above" *then*
if r is in between v_i and w_j *then*
classify v_i oppositely to that of w_j
else
classify v_i same as that of w_j
endif
endif
endif
endloop
if no such edge e_j is found *then*
classify v_i as "maybeon"
(**Comment:* To be classified later in the facet-plane classifications*)
endif
endif
end.

3.4.1.2 Facet-Plane Classification

If a facet f_i does not lie on a plane P, the points of intersection between them should necessarily be (i) collinear with the line of intersection $f_i \cap P$, and (ii) all vertices of f_i on one side of the intersection line should have the same classification w.r.t. the plane P. Vertices that have been temporarily classified as "maybeon" are classified in such a way that they satisfy the above two properties (i) and (ii) as closely as possible. Note that this heuristic forces the classification of "maybeon" vertices to be more consistent than the one obtained by classifying them arbitrarily. An algorithmic version of the facet-plane classification is given below.

Facet-Plane-Classif (f_i, P)
begin
case
(i) All vertices of f_i have been classified as "maybeon":
Classify f_i as "on" the plane and change the classification
of all incident vertices to "on".

(ii) At least one vertex v_u of f_i has been classified as "above", or "below",
but no edge of f_i has its two vertices
classified with opposite signs("below" and "above"):
if there is only one "maybeon" vertex v_i *then*
classify v_i as "on" and consider v_i as $f_i \cap P$
else
take two "maybeon" vertices v_i,v_j and
classify v_i and v_j as "on".
Let L be the line joining v_i, v_j.
Consider L as $f_i \cap P$.
loop

for each "maybeon" vertex v_k on f_i do

if v_k is at a distance greater than δ from L *then*

if v_k and v_u lie on opposite sides of L *then*

classify v_k with the opposite classification of v_u.

else

classify v_k with the classification of v_u.

endif

endif

endloop

endif

The vertices which are still not classified

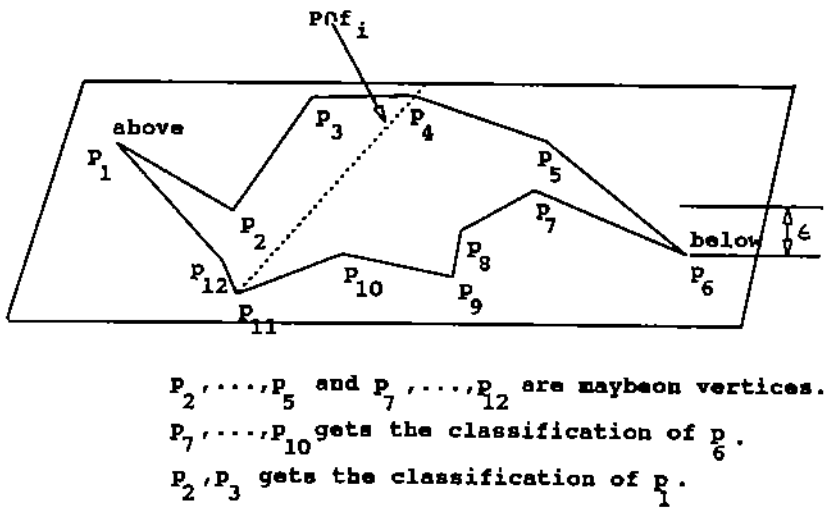


Figure 3.10 Case(ii) of facet-plane classification.

classify them as "on"

(**Comment:* these vertices are within a distance of δ from L and hence will be collinear with L by a perturbation of at most δ. See Figure 3.10.*)

(iii) There is an edge e whose two vertices have opposite sign classifications:

if there is no other such edge *then*

let L be the line joining the intersection point of e and

P to any "maybeon" vertex v_i.

classify v_i as "on".

consider L as $f_i \cap P$.

apply methods of case (ii) to classify other "maybeon" vertices.

else

let L be the line which fits in least square sense to all the points

of intersections and apply the methods of case (ii) to classify the

remaining "maybeon" vertices.

endif

endcase

end.

3.4.1.3 Edge-Plane Classification

An edge can receive any of the three classifications which are "not-intersected", "intersected", and "on". The classifications of the vertices incident on an edge e_i are used to classify it. An algorithmic version of the edge-plane classification is given below.

Edge-Plane-Classif (e_i, P)

begin

Let $e_i = (v_i, v_j)$.

case

(i) v_i and v_j are both classified as "on":

classify e_i as "on".

(ii) Only one of v_i, v_j, say v_i is classified as "on":
classify e_i as "intersected" and consider v_i as $e_i \cap P$.

(iii) v_i and v_j are classified with one as "above" and another as "below":
classify e_i as "intersected".
compute $r = e_i \cap P$ if it has not been computed yet.
if r does not lie within e *then*
choose a point at a distance of at least δ from the vertex
which is nearest to the computed point and consider it
as the intersection point of e_i and P.
endif

(iv) v_i and v_j are of same classifications and they are not "on":
classify e_i as "not-intersected".
endcase
end.

3.4.2 Nesting of Polygons with Finite Precision Arithmetic

Lemma 3.4.1 The problem of polygon nesting for k fleshy polygons with s vertices and t monotone chains can be solved in $O(k^2 + s(t + \log s))$ time under finite precision arithmetic.

Proof: Since any vertical line (orthogonal to the x direction) can intersect at most t edges of a set of polygons having t monotone chains, the above time bound is obvious from the time analysis of the algorithm under finite precision arithmetic as given in Section 2.4.♣

3.4.3 The Algorithm with Heuristics

The same paradigm of cutting and splitting polyhedra along the cuts is followed to produce the convex decomposition of a nonconvex, manifold polyhedron. One of the two planes supporting the facets incident on a notch is chosen as a notch plane. This ensures that no new plane other than facet-planes is introduced by the algorithm. As we have seen earlier, computations of intersection vertices involve plane equations incident on those vertices. Thus, using the original plane equations for such computations reduces the error propagation. Furthermore, this also guarantees that all input assumptions about the supporting planes of the facets remain valid throughout the iterative process of cutting and splitting the polyhedron. We apply heuristics at each numerical computation through geometric reasoning to make our algorithm as parsimonious as possible.

In the construction of GP_g, first all boundaries are computed. For this, one needs to compute the intersection vertices on the facets of S. This is carried out by the vertex-plane, edge-plane and facet-plane classifications as described before. Note that these classifications use heuristics that make the numerical computations more reliable. After computing all intersection vertices lying on a facet f, we sort them along the line of intersection $f \cap P_g$. Since the computed coordinates of these vertices are not exact, sorting them on the basis of their coordinates is prone to error. We use the minimum feature criteria and the orientations of the edges on a facet to obtain a topologically correct sort.

Two intersection vertices can be closer than δ if they lie on the edges which meet at a vertex. Other possibilities do not occur because of the minimum feature assumptions. Using the orientations of these two edges on the facet f containing them, the exact ordering of the two new intersection vertices on $f \cap P_g$ can be determined. Generation of edges between intersection vertices can be carried out exactly since it does not involve any numerical computation.

The cut Q_g is selected from GP_g using the method of Section 3.3.1.1. The polygon nesting algorithm, used for this purpose, is adapted to cope with the inexact numerical

computations as stated in Lemma 3.4.1. The polygon nesting algorithm with inexact arithmetic computations requires all input polygons to be fleshy. Although in most of the cases this is true, we do not know how to guarantee this property throughout the decomposition process. Refinement of Q_g needs proper transferring of the edges of S that are decided to be coplanar with P_g. This is done using the following simple heuristic. For an edge e computed to be "on" the plane P_g, we check all its oriented edges incident on facets computed to be "off" the notch plane P_g. Suppose, f is such a facet. Classify any vertex v of f w.r.t. the oriented edge of e on f. If it is on the same side of e in which f lies, e is transferred to GP_g^ℓ (GP_g^r respectively) if v has been classified to lie in P_g^ℓ (P_g^r respectively). It is trivial to decide the side of e in which f lies.

Splitting S about the cuts Q_g^ℓ and Q_g^r completes the cutting of S with the notch plane P_g. This step again does not involve any numerical computations.

Note that we assume the minimum feature property to be valid throughout the iterative process of cutting and splitting of polyhedra. Although for the original polyhedron it is valid, it may not be preserved throughout the entire cutting process. The method described in [SS85] can be used to eliminate this problem.

3.4.3.1 Complexity Analysis

We use Lemma 2.2.1 and Lemma 3.3.4 in our analysis which are valid only under the exact arithmetic model. Nonetheless, the analysis presented here gives a good estimate of the complexity of the algorithm.

Consistent vertex-plane, edge-plane and facet-plane classification take overall $O(p)$ time where p is the total number edges of the polyhedron S. The above bound follows from the fact that each edge of S is visited only $O(1)$ times to determine the intersection points of S with the notch plane P_g. The sorting of intersection vertices on the facets adds $O(u \log r)$ time where u is the total number of vertices in GP_g. Once the map GP_g is constructed, it is trivial to recognize the boundary B_g containing the notch g. The methods as described in Section 3.3.1.1 can be used to determine

the interesting boundaries. As discussed earlier, there are $O(t)$ interesting boundaries containing $O(t)$ monotone chains where t is the number of notches intersected by P_g. Let u' be the number of vertices on the interesting boundaries. According to Lemma 3.4.1, the children and parent of B_g can be determined exactly in $O(t^2+u'(t+\log u'))$ time if the polygons corresponding to the interesting boundaries are fleshy. Detection of children and parent of the polygon containing the notch g, in effect, determines the inner and outer boundaries of Q_g. Obviously $u' = O(u)$. Combining the complexities of computing GP_g and detecting the inner and outer boundaries of Q_g, we conclude that Q_g can be computed in $O(p + t^2 + u(t + \log u) + u \log r)$ time.

At a generic instance of the algorithm, let S_1, S_2, ..., S_k be the k distinct (nonconvex) polyhedra in the current decomposition that contain the subnotches of a notch g which is to be removed. Let p_i be the number of edges in S_i of which r_i are reflex, u_i be the number of vertices in the cross sectional map in S_i and t_i be the number of notches intersected by the notch plane in S_i. Let $p = \sum_{i=1}^{k} p_i$, $u = \sum_{i=1}^{k} u_i$ and $t = \sum_{i=1}^{k} t_i$. Certainly, $k = O(r)$ and $t = O(r)$ since a notch can have at most $r-1$ subnotches and a notch plane can intersect at most $r-1$ notches. The time $\Im$ to remove the notch g is given by

$$\begin{aligned} \Im &= O(\sum_{i=1}^{k}(p_i + t_i^2 + u_i(t_i + \log u_i) + u_i \log r_i)) \\ &= O(p + r^3 + ur + u \log u + u \log r). \end{aligned}$$

By Lemma 3.3.4, $u = O(n + r^2)$. This gives,

$$\begin{aligned} \Im &= O(p + r^3 + (n + r^2)r + (n + r^2) \log n) \\ &= O(nr + n \log n + r^2 \log n + r^3) \\ &= O(nr + n \log n + r^3) \end{aligned}$$

To eliminate r notches, we need $O(nr^2 + nr \log n + r^4)$ time. Obviously, the space complexity is $O(p) = O(nr + r^2\alpha(r))$. If S is a non-manifold polyhedron, all special

notches are removed from S to produce manifold polyhedra each of which is decomposed into convex pieces by the method as discussed before. The complexity remains the same for this case. ♣

3.4.4 Experimental Results

We have implemented our polyhedral decomposition algorithm under floating point arithmetic in Common Lisp on UNIX workstations. The numerical computations are all in C, callable from Lisp using interprocess communications. We used $\delta = 2^{-17}$ in the 32 bit machine with precision 2^{-24}. Simple examples are shown in Figure 3.11 and in Figure 3.12. The experimental results have been very satisfying. Test polyhedra are created and results are displayed in the X-11 window based. SHILP solid modeling and display system.

3.5 Conclusions

In this chapter we have given an $O(nr^2 + r^3 \log r)$ time and $O(nr + r^2\alpha(r))$ space algorithm for convex decompositions of polyhedra with arbitrary genus and shells. Although a better algorithm for polyhedra with zero genus and no shell exists, this is the best known algorithm for polyhedra with arbitrary genus and shells. The analysis of the algorithm which uses a marvelous theorem ("zone theorem") from combinatorial geometry shows that the method of successive cutting and splitting polyhedra with planes is not as costly as they were thought to be in [Cha84]. It is an open question whether we can further reduce the complexities.

Minimum convex partition is known to be NP-hard for polyhedra with holes. It remains an open question whether minimum convex partition is still NP-hard for polyhedra without any hole.

Designing a robust algorithm of any type (preferably type-4 and type-5) for convex decompositions is a crucial open problem. To have any success in this respect, we have to understand the deep interactions between the underlying topology of polyhedra and perturbations in their features.

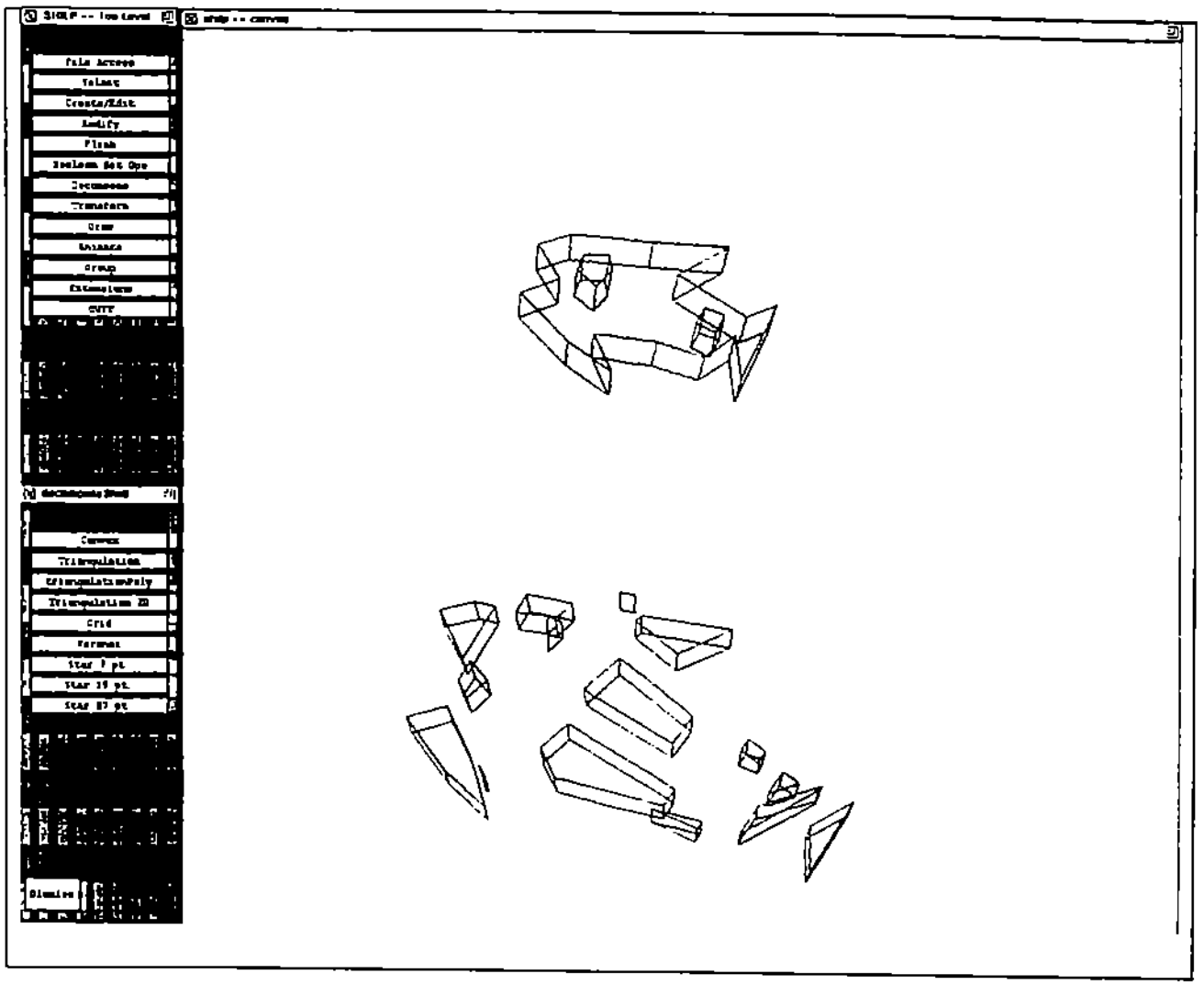


Figure 3.11 Convex decomposition.

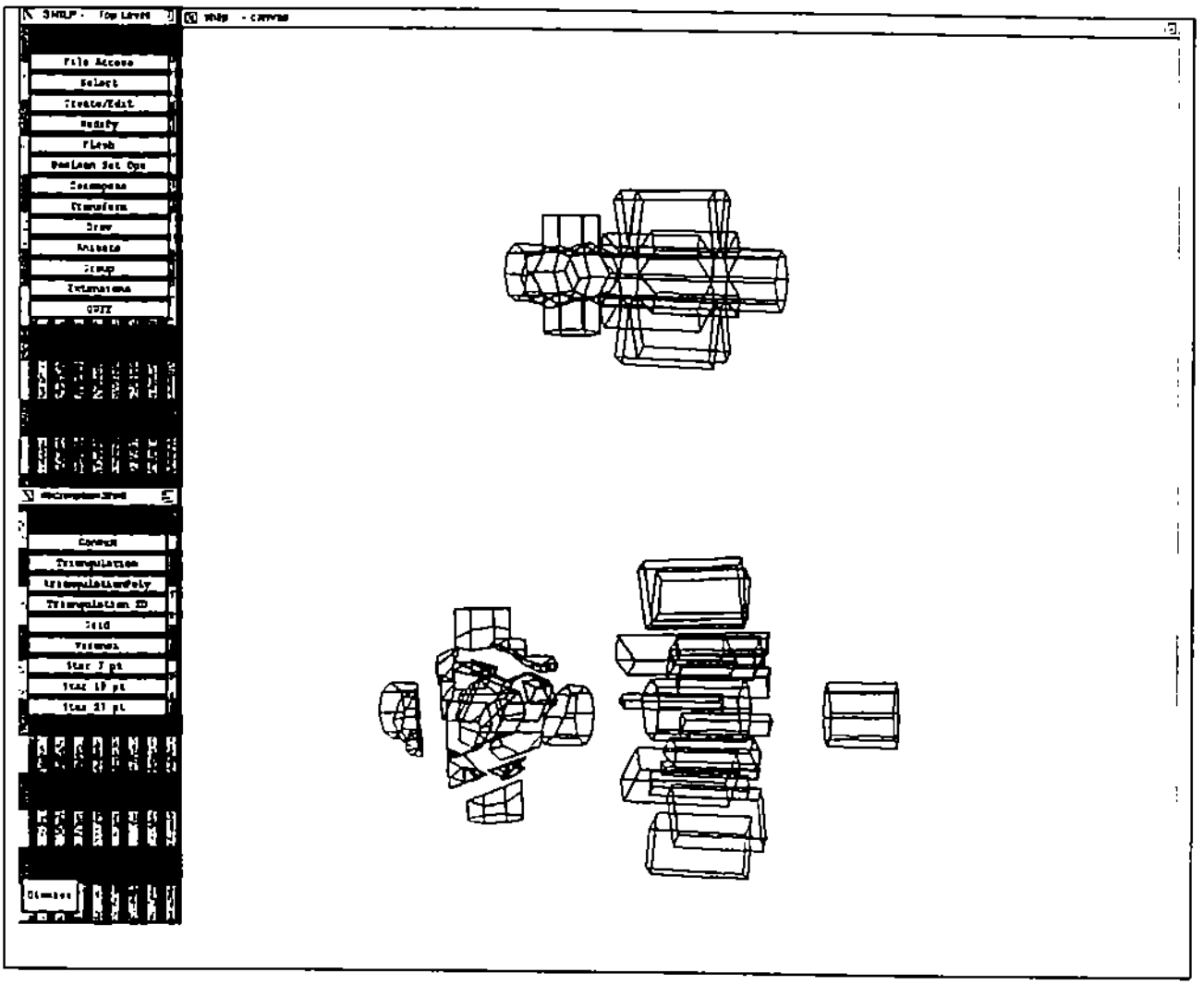


Figure 3.12 Convex decomposition.

4. CSG DECOMPOSITIONS AND TRIANGULATIONS

4.1 Introduction

In this chapter, we will see how the convex decompositions as discussed in the previous chapter lead to a special type of CSG decompositions of polyhedra as well as their triangulations.

A CSG decomposition, in terms of regularized boolean operations such as intersection, union, difference, complement on simpler components, is used for CSG tree representation of polyhedra. The expression involving these boolean operations together with the literals corresponding to the simpler components is referred to as CSG formula. In [Pet84], Peterson considered CSG formulae that allow only intersection and union of the halfspaces supporting the facets of polyhedra. We call such formulae as Peterson-style CSG formulae. The problem of computing the Peterson-style CSG formulae for polyhedra from their boundary representations often arises in solid modeling and computer graphics [DGHS88]. In 2D, Dobkin, Guibas, Hershberger, and Snoeyink [DGHS88] give an $O(n \log n)$ time algorithm to compute the Peterson style CSG formulae of size $O(n)$ for polygons with n vertices. They posed the question of computing short Peterson-style CSG formulae for polyhedra in 3D. As they pointed out, $O(p^3)$ size Peterson-style CSG formulae for polyhedra with p facets is trivial to compute. In [PY90], Paterson and Yao give an $O(p^3)$ time algorithm to compute the Peterson-style CSG formulae of size $O(p^2)$ for a restricted class of polyhedra. These polyhedra, however, have only convex facets with $O(1)$ edges. We consider more general polyhedra that may have nonconvex facets with arbitrary number of edges.

Establishing a nontrivial lower bound on the size of Peterson-style formulae for general polyhedra is an open question. We prove an $O(p^2)$ lower bound for the following two types of Peterson-style formulae. Let $(\alpha_{11}o_{11}\alpha_{12}...)r_1(\alpha_{21}o_{21}\alpha_{22}...)r_2...(\alpha_{k1}o_{k1}...)$

be a Peterson-style formula for a polyhedron where o_{ij}'s and r_i's denote the operators intersections ($\cap$) or unions ($\cup$), and α_{ij}'s denote the literals corresponding to the simpler components. In case where $o_{ij} = \cap$ and $r_i = \cup$ for all i, j, we say that the given Peterson-style formula is in disjunctive normal form (DNF). On the other hand, if $o_{ij} = \cup$ and $r_i = \cap$ for all i, j, we say that the given formula is in conjunctive normal form (CNF). We refer to such formulae as CNF Peterson-style and DNF Peterson-style formulae respectively.

In triangulations we seek for simplicial decompositions of the given polyhedron that produce simplicial complex. This is a non trivial step in finite element mesh generation for polyhedral domains. In three dimensions, there are polyhedra that are not triangulable without additional Steiner points. Moreover, as shown by Rupert and Seidel [RS89], the general problem of determining whether a polyhedron is triangulable without Steiner points or not is NP-hard. Due to these constraints, we consider the problem of triangulation with Steiner points. Chazelle's worst-case lower bound on convex decomposition suggests an $O(r^2)$ worst case lower bound on the output size of triangulations of polyhedra. Recently, in [CP90], Chazelle and Palios have given an $O((n + r^2)\log r)$ time algorithm that tetrahedralize simple polyhedra and produces $O(n + r^2)$ tetrahedra. The allowed polyhedra are homeomorphic to spheres, i.e., they cannot have holes (genus 0) and shells (internal voids) and are manifold. This algorithm, however, does not produce a simplicial complex, i.e., the generated tetrahedra may not meet at a full facet or an edge. Its analysis relies upon the fact that the input polyhedra are homeomorphic to spheres. It is not clear how one can generalize this algorithm for polyhedra with arbitrary genus and shells in acceptable time and space bounds. In this chapter, we give an algorithm for triangulating (producing a simplicial complex) manifold polyhedra with arbitrary genus and shells. To handle non-manifold polyhedra that have special notches, a preprocessing as described in Section 3.3.2 is carried out.

The basis of our algorithms for triangulation and computation of the Peterson-style CSG representation of polyhedra is the convex decomposition algorithm as discussed in Chapter 3.

In Section 4.2, we show that we can obtain a Peterson-style formula of size $O(p^2\alpha(p))$ for any manifold polyhedron through our convex decomposition algorithm in $O(p^3 \log p)$ time. Here p is the number of facets of the polyhedron. We establish an $O(p^2)$ lower bound on CNF and DNF Peterson-style formulae for polyhedra.

In Section 4.3, using our convex decomposition algorithm, we give an $O(nr^2 + r^3 \log r)$ time and $O(nr + r^3)$ space algorithm to triangulate a manifold polyhedron with arbitrary genus and shells.

4.2 CSG Decomposition

The convex decomposition algorithm as described in the previous chapter gives the Peterson-style CSG formulae for polyhedra when the notch planes are carefully chosen. For each notch g in S, if the plane supporting one of the facets adjacent to g is chosen as the notch plane for g, all facets of the convex pieces in the final decomposition lie only on the supporting planes of the facets of S. Further, each convex piece can be expressed as the intersection of half-spaces corresponding to the supporting planes of its facets. Finally, S can be represented as the union of the expressions obtained for each convex piece. This gives a Peterson-style CSG formula for S. The number of literals in this formula is equal to the number of facets present in the convex pieces.

4.2.1 Upper Bound

Theorem 4.2.1 For any manifold polyhedron, a Peterson-style CSG formula of size $O(pl + l^2\alpha(l))$ can be computed in $O(pl^2 + l^3 \log l)$ time where p is the number of facets in S of which l are adjacent to notches.

Proof: By Lemma 3.3.3, the total number of edges in the final decomposition is $O(nr + r^2\alpha(r))$. Certainly, $r = O(l)$ and since S is a manifold polyhedron $n = O(p)$. Thus, the total number of facets in the convex pieces of final decomposition is $O(pl + l^2\alpha(l))$. This determines the size of the Peterson style CSG formula of S. The time complexity for this CSG computation is same as that of computing the convex decomposition of S. Expressed in terms of p and l, this complexity is $O(pl^2 + l^3 \log l)$.♣

An upper bound of $O(p^2\alpha(p))$ on the size of Peterson-style CSG formulae that can be computed in $O(p^3 \log p)$ time follows from the fact that $l = O(p)$.

4.2.2 Lower Bound

Lemma 4.2.1 There exists a class of polyhedra for which any CNF Peterson-style CSG formula has a size of $O(p^2)$ where p is the number of facets of S.

Proof: Consider the polyhedron S as constructed by Chazelle in [Cha84] to prove a lower bound on the number of convex pieces needed to decompose a non-convex polyhedron. The notches of this polyhedron form two sets of line segments, each lying on the surface of a hyperbolic paraboloid which have a small distance of ϵ between them; see Figure 4.1.

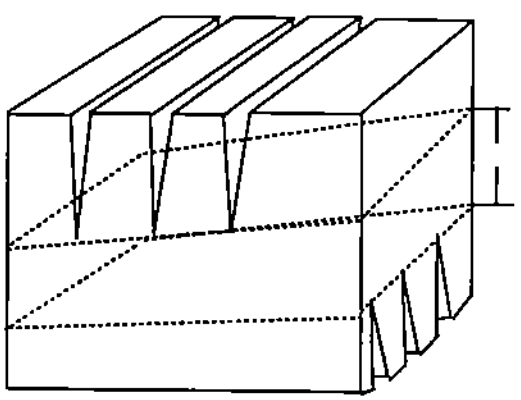


Figure 4.1 Chazelle's solid with two sets of notches.

Let Σ denote the region between these two hyperbolic paraboloid surfaces each containing r notches. Assuming unit distances between consecutive notches, the volume of Σ is $O(\epsilon r^2)$. Chazelle showed that a single convex polyhedron whose volume lies inside S can occupy only $O(\epsilon)$ volume in Σ, thus requiring $O(r^2)$ convex pieces to cover Σ. Let $C = C_1 \cup C_2 \cup, ..., \cup C_k$ be the CNF Peterson-style CSG formula for S where each C_i represents the maximal collection of literals along with only intersection operators in between them. Each C_i represents a closed convex polyhedron S_i. The polyhedron S_i is convex since it is constructed by the intersection of finite number of halfspaces and it is closed since its union with S is closed. The convex polyhedra corresponding to $C_i, i = 1, ..., k$ cover the polyhedron S and hence Σ. Thus k must be $O(r^2)$ giving an $O(r^2)$ lower bound on the size of C. The worst-case lower bound of $O(p^2)$ follows immediately from the fact that S can be made to have $r = O(p)$.♣

Lemma 4.2.2 There exists a class of polyhedra for which any DNF Peterson-style CSG formula has a size of $O(p^2)$ where p is the number of facets of S.

Proof: Consider a polyhedron S_0 constructed as follows. Let S_1 be the unbounded polyhedron obtained by taking the closure of the complement of the Chazelle's solid. The unbounded polyhedron S_1 has an internal void whose boundary is exactly similar to that of Chazelle's solid. Let S_2 be a cube, large enough to contain the internal void of S_1 inside. Let $S_0 = cl(S_1 \cap S_2)$. The polyhedron S_0 is a closed polyhedron. Its outer boundary consists of six facets of the cube S_1, and its inner boundary consists of the boundary of Chazelle's solid. Let $C = C_1 \cap C_2 \cap C_3 ... \cap C_k$ be a DNF Peterson-style CSG formula for S_0 where each C_i represents the maximal collection of literals along with only union operators. Let $\overline{C_i}$ represent the complement of C_i where the complement of a closed halfspace H_i is replaced by $cl(\overline{H_i})$, another closed halfspace. The formula $\overline{C} = \overline{C_1} \cup \overline{C_2} \cup ... \cup \overline{C_k}$ is a CNF Peterson-style formula that represents two disjoint polyhedra, the Chazelle's solid and the unbounded polyhedron $cl(\overline{S_1})$ corresponding to the complement of S_1. Each $\overline{C_i}$ represents a convex polyhedron that lies completely either inside the Chazelle's solid or inside the unbounded polyhedron $cl(\overline{S_1})$. Since

the portion denoted by Σ in the Chazelle's solid is covered by convex polyhedra that lie inside it, k must be $O(r^2)$. Making $r = O(p)$, we have $k = O(p^2)$.

Theorem 4.2.2 There exists a class of polyhedra for which any DNF or CNF Peterson-style CSG formula has a size of $O(p^2)$ where p is the number of facets of S.

Proof: Consider a solid that is formed by gluing Chazelle's solid to the solid S_0 as described in Lemma 4.2.2. From the proof of Lemma 4.2.1 and Lemma 4.2.2, it is clear that any CNF or DNF Peterson style formula for this solid has $O(p^2)$ size.

4.3 Triangulation

We observe that the triangulation of each convex piece produced by *ConvDecomp* of Section 3.3 does not yield a triangulation of the original polyhedron S since two facets created corresponding to the cut Q_g may be decomposed differently later by other notch planes. Thus, the triangulation of the portions where these facets touch each other may not match giving an invalid triangulation of S; see Figure 4.2.

4.3.1 Complete Cuts

We can overcome the problem of mismatch of facet triangulations if we cut through the entire polyhedron S each time with a notch plane. In other words, all *sub polyhedra* through which a notch plane passes are partitioned with that notch plane. We call such slicings as *complete cuts*. With such slicings, all edges on a facet will be present in other touching facet. For such decompositions, we cannot use Lemma 3.3.2 since the new edges created by complete cuts are not restricted to the regions adjacent to the notch g. In fact, in this case, we have to consider all edges inside and on GP_g in the arrangement of notch line segments with GP_g superimposed on it. The natural expectation is that the complete cuts are costly. In Lemma 4.3.1 and 4.3.2, we show that the time and space complexities do not change much due to the complete cuts.

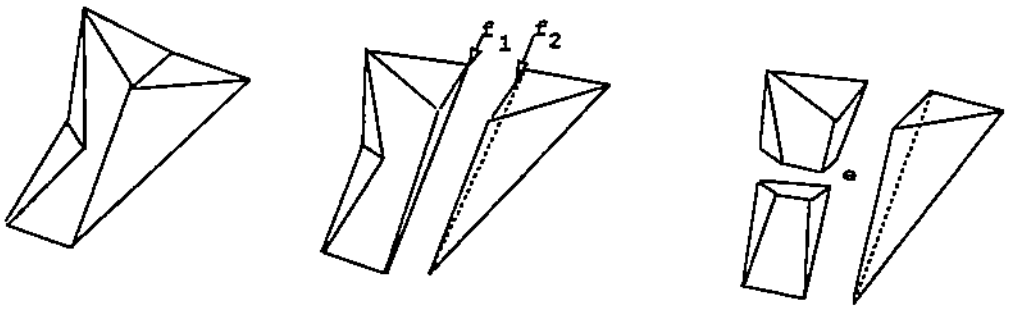


Figure 4.2 Edge e causes mismatch on f_1 and f_2.

4.3.2 Analysis of Complete Cuts

Lemma 4.3.1 If a polyhedron S is decomposed by complete cuts, the number of edges in the final decomposition is $O(nr + r^3)$.

Proof: By Lemma 3.3.4, the number of edges on and inside GP_g for each notch g is only $O(n + r^2)$. This implies that one complete cut generate $O(n + r^2)$ new edges. Thus, r complete cuts produce $O(nr + r^3)$ new edges.♣

If we use the size of the final decomposition (Lemma 4.3.1) to estimate the number of edges in the subpolyhedra through which a complete cut passes, we get $O(\sum_{i=1}^{k} m_i) = O(nr+r^3)$ in Theorem 3.3.1. This gives a straightforward $O(nr^2+r^4)$ time complexity for decompositions with complete cuts. However, the following lemma helps us to show that the true complexity is lower than this.

Lemma 4.3.2 If a polyhedron S is decomposed by complete cuts, the total number of edges in subpolyhedra through which a complete cut passes is only $O(nr)$.

Proof: Consider the complete cut corresponding to the plane P_g. let R be the set of planes used before P_g for other complete cuts. The planes in $R \cup P_g$ form an arrangement A of planes in three dimensions. The cells adjacent to the plane P_g in A constitute the zone Z_g of P_g. By well known zone theorem [Edes], the number

of edges in Z_g is $O(q^2)$ if there are q planes in the arrangement. Let A' be the new arrangement obtained by superimposing the boundary facets of S on Z_g. Consider the cells adjacent to P_g that constitute the zone Z'_g in A'. Subpolyhedra through which P_g passes consist of cells that are members of Z'_g. Thus, the number of edges in Z'_g gives an upper bound on the number of edges of subpolyhedra through which P_g passes. To count the number of edges in Z'_g, we carefully analyze the effect of superimposing p boundary facets of S on Z_g.

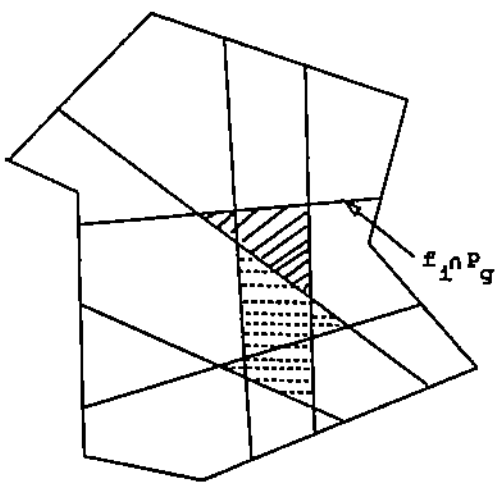


Figure 4.3 The facets in F_i are hatched with dotted lines; facets in F'_i are hatched with solid lines; facets in B'_i are not hatched.

Let f_i be a facet of S that contributes to the boundaries of some cells in Z'_g. Consider the lines of intersections between f_i and the other facets of Z'_g. These lines together with the line segments supporting the edges of f_i form an arrangement of line segments on the plane supporting f_i. Let B_i denote the facets in this arrangement that are inside f_i. Further, let B'_i denote the set of facets in B_i that are adjacent to line segments supporting the edges of f_i; B''_i denote the rest of the facets in B_i. Let F_i denote the set of facets in B''_i that do not have any edge formed by the intersection

of P_g with f_i; F_i' denote the rest of the facets in B_i''; see Figure 4.3. In the following, by $E(F)$ we denote the number of edges in a set F of facets.

The facets in F_i are created by slicing the cells in Z_g completely by f_i such that f_i does not intersect P_g inside those cells. The portions of these cells that remain in Z_g' after this slicing are not intersected by any other facet of S. Thus, a facet of Z_g contributes at most one edge in $\cup_{i=1}^{p} F_i$. This implies that $\sum_{i=1}^{p} E(F_i)$ is bounded above by the number of facets in Z_g giving $\sum_{i=1}^{p} E(F_i) = O(q^2)$. All other facets in B_i'' (if any) are adjacent to the line of intersection of f_i with P_g. Thus, the facets in F_i' are members of the zone of this line in an arrangement of $O(q)$ lines. Since there can be at most p lines of intersection between the planes supporting the facets of S and P_g, we get $\sum_{i=1}^{p} E(F_i') = O(pq)$ by applying the zone theorem of line arrangement. This gives $\sum_{i=1}^{p} E(B_i'') = \sum_{i=1}^{p} E(F_i) + E(F_i') = O(pq + q^2)$.

To estimate the number of edges in the facets of B_i', consider the arrangement of $O(q)$ lines that represent the intersections between the supporting plane of f_i and the planes in R. The number of edges in the facets of B_i' adjacent to an edge e of f_i can be estimated by the number edges in the facets through which the line supporting e passes in this arrangement. This number is $O(q)$. Since S has n edges, we have $\sum_{i=1}^{p} E(B_i') = O(nq)$. Combining all these, we get that the number of new edges contributed to Z_g' as a result of superimposing p facets of S on Z_g is only $O(pq + nq + q^2) = O(nr)$ since $q = O(r)$, $p = O(n)$. This immediately implies that Z_g' have at most $O(r^2 + nr) = O(nr)$ edges. Thus, the total number of edges in subpolyhedra through which the plane P_g passes is at most $O(nr)$. ♣

Theorem 4.3.1 A manifold polyhedron S with arbitrary genus and having n edges of which r are reflex can be triangulated in $O(nr^2 + r^3 \log r)$ time and $O(nr + r^3)$ space.

Proof: We get $\sum_{i=1}^{k} m_i = O(nr)$ in Theorem 3.3.1 using Lemma 4.3.2. This gives an $O(nr^2 + r^3 \log r)$ time bound for convex decompositions through complete cuts. Lemma 4.3.1 gives $O(nr + r^3)$ space complexity. Each convex piece can be triangulated in a straightforward way by triangulating its facets and joining all triangles thus

produced to a point inside the convex piece. However, we need to ensure that all pairs of facets that overlap completely on one another have same triangulation. Since the facets in each such pair have same topological structure and have the same geometric location, any deterministic algorithm that triangulates a facet can be made to produce same triangulations for both facets. This triangulation phase does not increase the time and space complexity.♣

4.4 Conclusions

This chapter shows that how a simple algorithm for convex decomposition can lead to efficient algorithms for triangulations and Peterson-style CSG decompositions of polyhedra. The complexity analysis of the complete cuts exhibits again the power of "zone theorem". Lemma 4.3.2 has implications beyond its use in complete cuts. Given an arrangement A of r planes in three dimensions, Lemma 4.3.2 shows that the zone complexity of each plane is $O(nr + r^2)$ if n planar facets intersecting only at the boundaries are superimposed on A. We believe that this combinatorial fact would be useful in analyzing other algorithms.

We have proved an $O(p^2)$ lower bound for CNF and DNF Peterson-style formulae in case of polyhedra. Proving a non trivial lower bound for general Peterson-style formulae for polyhedra remains open. We suspect that this bound is also $O(p^2)$.

5. GOOD TRIANGULATIONS

5.1 Introduction

In the previous chapter, we described an algorithm that triangulates polyhedra. This triangulation method, however, does not guarantee anything about the shapes of the tetrahedra. As a result, it is possible that very thin and flat tetrahedra are generated. To reduce ill-conditioning as well as discretization error, finite element methods require triangular meshes where the elements are well-shaped, i.e., they do not have very small and very large angles [BA76, Fri72]. These type of triangulations where shapes of the triangular elements are guaranteed to be good are called guaranteed quality triangulations or good triangulations.

In 2D, there are basically three approaches known so far to produce guaranteed quality triangulations. The first approach based on the *Constrained Delaunay Triangulations* was first suggested by Chew [Che89]. He guarantees that all triangles produced in the final triangulation have angles between 30° and 120°. In [Dey90], we improved this algorithm with minor modifications to guarantee the boundary triangles to have better angle bounds. There is another approach based on *Grid Overlaying* which was first used by Baker, Grosse, and Raferty in [BGR88] to produce a nonobtuse triangulation of a polygon. In [Dey90], we proposed a simpler method based on this grid approach to triangulate a polygon with good angles. Bern, Eppstein, in a current paper [BE91], give an improved method (w.r.t. the number of extra points added by the algorithm) for nonobtuse triangulation of a polygon. In [BEG90], Bern, Eppstein, and Gilbert give algorithms for producing good triangulations which uses a special type of a grid that simulates the planar subdivision with the quadtree. Another approach proposed by [SNT90] is based on the *medial axis transformation* that produces an adaptive triangular mesh and eliminates bad triangles.

Although a number of algorithms exist for triangulating a point set or a polyhedron in 3D [AE86, CP90, EPW86, Joe89], few of them address the problem of guaranteeing the shape of the triangular elements. This chapter presents an algorithm that triangulates the convex hull of a point set in 3D with guaranteed quality tetrahedra. The problem allows one to introduce new points to generate good tetrahedra with the restriction that all points are added only inside or on the boundary of the convex hull. Good triangulations of convex polyhedra are a special case of this problem.

In Section 5.3, we present the 3D triangulation algorithm based on the Delaunay triangulations as used by Chew [Che89] in 2D. We characterize the bad tetrahedra in 3D and show that the algorithm does not produce four out of five possible types of bad tetrahedra. We also give a bound on the number of additional points used to achieve this guarantee. In Section 5.4, we present a type-2 robust algorithm for 3D Delaunay triangulations. This algorithm is used in the robust implementation of our good triangulation algorithm.

5.2 Preliminaries

5.2.1 Characterizing Bad Tetrahedra

In 3D, a tetrahedron can be degenerate or bad in three possible ways as described in [Bak89]. The following two parameters ω, κ characterize bad tetrahedra as follows. Let $\omega = \frac{R}{L}$ and $\kappa = \frac{L}{l}$ where R is the radius of the circumscribing sphere of a tetrahedron, L and l are the lengths of its longest and shortest edges respectively. Bad tetrahedra can be classified into three categories.

Category(i): $\omega = O(1), \kappa \gg 1$.

Category(ii): $\omega \gg 1$.

Category(iii): $\omega = O(1)$, $\kappa = O(1)$.

Definition 5.2.1 A sliver is a tetrahedron that is formed by four almost coplanar points and whose solid angles are very close to zero.

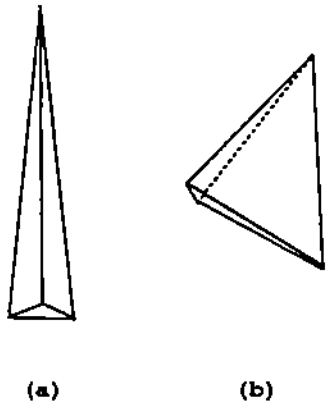


Figure 5.1 Category(i) tetrahedra.

Category(i) corresponds to tetrahedra that have a very short edge relative to the other edges and have circumscribing spheres that do not have an arbitrarily large radius compared to the length of the longest edge. Specifically, category(i) consists of type(i) and type(ii) tetrahedra. Type(i) tetrahedra are needle-like tetrahedra in which one of the solid angles is highly acute and the face opposite to it has a negligible area (Figure 5.1(a)). Type(ii) tetrahedra are slivers with a very short edge (Figure 5.1(b)).

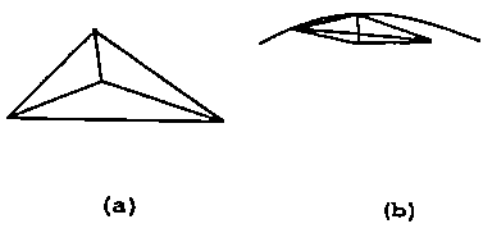


Figure 5.2 Category(ii) tetrahedra.

Category(ii) corresponds to tetrahedra that have a circumscribing sphere with arbitrarily large radius compared to the longest edge. Specifically, category(ii) consists

of type(iii) and type(iv) tetrahedra. Type(iii) tetrahedra are flat tetrahedra which have one of the solid angles highly obtuse (Figure 5.2(a)). Type(iv) tetrahedra are slivers which lie very close to the surface of their large circumscribing spheres (Figure 5.2(b)). Category(iii) consists of type(v) tetrahedra. Type(v) tetrahedra are slivers whose edges have lengths within a constant factor of each other and which do not have a close incidence with the surface of the circumscribing sphere (Figure 5.3). We

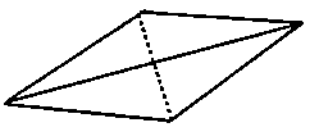


Figure 5.3 Category(iii) tetrahedra.

present an algorithm that triangulates the convex hull of a three dimensional point set with the guarantee that type(i) through type(iv) tetrahedra are not generated.

5.2.2 2D Algorithm

The core of the algorithm presented in this paper consists of the Delaunay triangulation which is the straight line dual of the Voronoi diagram. In 2D, the circumscribing circle of a triangle in the Delaunay triangulation of a point set does not contain any other point inside it. Similarly, in 3D, the circumscribing sphere of a tetrahedron in the Delaunay triangulation does not contain any other point inside it. This property of the Delaunay triangulation is utilized by Chew in 2D to produce good triangulations. He introduces the centers of those circumscribing circles that maintain a certain minimum distance from the three vertices of the corresponding triangle. Of course, the edges of the boundary have to satisfy certain length criteria. In his algorithm, Chew uses edge lengths in between d and $\sqrt{3}d$ where any pair of input points is at least d units away from each other. In the modified algorithm of [Dey90], we require edge lengths in between d and $1.5d$. This gives two distinct advantages.

1. It is easier to divide edges between d and $1.5d$ in practice.
2. The triangles that have circumcenters outside the boundary have better bounds on their angles.

We present below this modified algorithm for good triangulations in 2D.

Algorithm 2D-Tri:

Input: Finite number of points in the plane within a polygonal boundary. The vertices of the polygonal boundary are included in the input point set.

Input Conditions: There exists a quantity d, such that no two given points are closer than d and no boundary edge is greater than $1.5d$ and less than d.

begin

Construct the Delaunay triangulation
of the given point set.

Repeat

Add the circumcenter v of a
triangle $g = \triangle p_i p_j p_k$ satisfying
the following property:
v_l is at a distance of at least d from all
three points p_i, p_j, p_k.
Update the current triangulation by constructing
the Delaunay triangulation
of the augmented point set.

Until there is no such triangle.

end

Original edges of the polygon are divided to satisfy the input conditions of *2D-Tri*. However, caution should be taken to ensure that the new points, thus generated on the edges, are at least d units away from each other. For a simple polygonal boundary with a certain lower bound ($30°$) on the minimum internal angles at the vertices, it

is always possible to divide the original edges so that the new points also satisfy the input conditions. Algorithm *2D-Tri* produces a planar triangulation T that has the following properties.

Property 1: All edges in T have lengths in between d and $2d$, and, in particular, all boundary edges have lengths in between d and $1.5d$.

Property 2: The circumscribing circle of all triangles in T has radius less than d.

5.2.3 Geometric Lemmas

We use the following geometric lemmas in the next section.

Lemma 5.2.1 Let T be the Delaunay triangulation of a point set in 2D. Let R be the maximum radius of all circumscribing circles of the Delaunay triangles in T. The radius of any empty circle whose center lies inside T is less than or equal to R.

Proof: See Theorem 6.15 of [PS86]. ♣

Definition 5.2.2 Let c be a circle drawn on the surface of a sphere s. Let $\overline{p_1 p_2}$ be the axis which is perpendicular to the supporting plane of c and which passes through the center of c. This axis intersects s at p_1 and p_2. The points p_1, p_2 are called the *poles* corresponding to the circle c.

Lemma 5.2.2 Let c be a circle with the radius less than r drawn on the surface of a sphere s. Let the distance between c and its nearest pole be greater than d. The radius R of s must satisfy the condition $R < \frac{r^2+d^2}{2d}$.

Proof: Consider the circle c as shown in Figure 5.4 with the nearest pole p_1. Let a, b be the centers of s and c respectively. Obviously, $|\overline{ab}| < (R - d)$. Consider the right angled triangle $\triangle abt$ where t is a point on the circle c. Since the radius of c is less than r, we have $|\overline{bt}| < r$. Hence, $|\overline{at}^2| = R^2 = |\overline{ab}|^2 + |\overline{bt}|^2 < (R - d)^2 + r^2$ giving $R < \frac{r^2+d^2}{2d}$. ♣

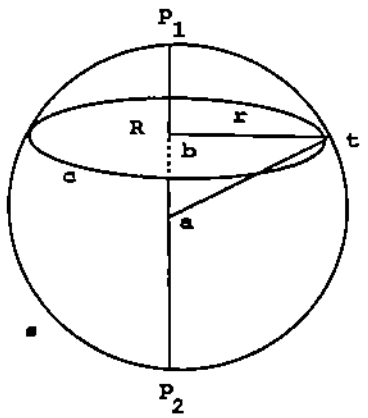


Figure 5.4 Poles and circles on a sphere.

5.3 3D Algorithm

In this section, we describe the good triangulation algorithm for a three dimensional point set. In what follows, by the convex hull of a point set, we mean its interior along with its boundary. We refer to the boundary of the convex hull as the *boundary*. A point is called an *internal point* if it is not on the boundary and is called a *boundary point* otherwise. The facets of the boundary are referred to as *boundary facets* and the edges on the boundary facets are called *boundary edges*.

Algorithm 3D-Tri:

Input: Finite number of points in three dimensional space.

begin

Let d_1 be the minimum of the distances
between two points.
Let d_2 be the minimum distance from an
internal point to a boundary facet.
Let d_3 be the minimum distance between
two nonadjacent boundary facets.

Let $r = \frac{1}{6}min\{d_1, d_2, d_3\}$.

Triangulate each facet of the boundary using

algorithm 2D-Tri in such a way that

every edge has length in between r *and* $2r$ *and*

every boundary edge

has length in between r *and* $1.5r$.

Let P *be the current point set.*

Construct a 3D Delaunay triangulation $T(P)$

of the point set P.

repeat

Add the center v *of the circumscribing sphere*

of a tetrahedron t_i *in* $T(P)$

satisfying the following properties:

(i) all four vertices of t_i *are at a distance of*

at least $2r$ *from* v,

(ii) the center v *lies inside the boundary.*

Set $P = P \cup v$.

Update the Delaunay triangulation $T(P)$.

until there is no such tetrahedron.

end

With the above choice of r and with the assumption that all the face-angles of the facets on the boundary satisfy the minimum angle criterion, it is possible to triangulate them by *2D-Tri* maintaining the edge lengths as stated. In the following Lemma, we prove that the above procedure terminates.

Lemma 5.3.1 Algorithm *3D-Tri* terminates.

Proof: Algorithm *2D-Tri* terminates since the points added by it are always at a certain distance from all other points. There can be only finitely many such points inside the given polygonal boundary. Extending this argument to Algorithm *3D-Tri*,

we can observe that all circumcenters of tetrahedra that are added as new points are at a distance of at least $2r$ from all other points. There can be only finitely many such points inside the convex hull of the input points assuring the termination of the Algorithm *3D-Tri.* ♣

5.3.1 Lower Bounds on Distances

Lemma 5.3.2 Any point on a boundary facet that does not lie on a boundary edge must be at a distance of at least $\frac{\sqrt{7}}{4}r$ from all edges of that facet.

Proof: Consider a point p on a facet f. Let e be any edge of f. Note that the edge e is divided into smaller edges $e_1, e_2, ..., e_n$ through the triangulation of the boundary facets adjacent to e. Drop a perpendicular from p on the line supporting e. If the perpendicular intersects the edge e, let e_l be the edge of the triangulation on e which is intersected by it. According to property 1, all boundary edges of the triangulation of f must have lengths in between r and $1.5r$. Further, the point p is at least r units away from the end points of e_l. Thus, the minimum distance between p and e_l is at least $\frac{\sqrt{7}}{4}r$. In case the perpendicular dropped from p does not intersect e, it must intersect some other edge e' of f. In that case, the distance between p and e must be greater than the distance between p and e'. We can estimate the minimum distance between p and e by estimating the same between p and e'. While estimating the distance between p and e', if it occurs that the perpendicular dropped from p does not intersect e', we will have another edge to estimate the minimum distance between p and e'. Since there are finite number of edges, and since each time we go to a next edge, its distance from p gets smaller than the previous one, there must be an edge of f which is intersected by the perpendicular dropped from p. Let e'' be the first such edge encountered in the above process. As argued above, the distance between p and e'' is at least $\frac{\sqrt{7}}{4}r$. Hence, the distance between p and e is at least $\frac{\sqrt{7}}{4}r$. Thus, any point on a boundary facet that does not lie on a boundary edge must be at a distance of at least $\frac{\sqrt{7}}{4}r$ from all edges of that facet.♣

Lemma 5.3.3 All edges in the triangulation produced by the algorithm *3D-Tri* have lengths greater than $l_{\min}$ where $l_{\min} = \min(r, \frac{\sqrt{7}}{2} r \sin\frac{\theta_m}{2})$. Here θ_m is the minimum dihedral angle between two adjacent boundary facets.

Proof: Initially, all internal points are at a distance of at least $6r$ units from every other point. Two boundary points, lying on non adjacent facets, are at least $6r$ units away from each other. These conditions are ensured by the particular choice of r. A boundary point is at a distance of at least r from every other point on the same facet which is ensured by the algorithm *2D-Tri*. The points added by the algorithm *3D-Tri* are always at a distance of at least $2r$ from every other point. Thus, all points except the points on the adjacent facets are at a distance of at least r from each other. To estimate the minimum distance between any two points on the adjacent boundary facets, consider two points p_i, p_j lying on the adjacent facets f_i, f_j respectively. Let e be the edge shared by f_i and f_j. Drop a perpendicular from p_i on e. Let it meet e at p_m. Consider the triangle $\triangle p_i p_j p_m$. Let the minimum dihedral angle between any two adjacent facets be θ_m. It is easy to prove that the angle between $\overline{p_i p_m}$ and $\overline{p_j p_m}$ in the triangle $\triangle p_i p_j p_m$ must be at least θ_m. From the above discussion, it follows that $|\overline{p_i p_m}| > \frac{\sqrt{7}}{4} r$ and $|\overline{p_j p_m}| > \frac{\sqrt{7}}{4} r$. Thus, the distance between p_i and p_j is at least $\frac{\sqrt{7}}{2} r \sin\frac{\theta_m}{2}$. Hence, all edges in the final triangulation produced by the algorithm *3D-Tri* have lengths greater than $l_{\min} = \min(r, \frac{\sqrt{7}}{2} r \sin\frac{\theta_m}{2})$. ♣

Lemma 5.3.4 Any point p present as a vertex in the triangulation produced by the algorithm *3D-Tri* is at a distance of at least $\frac{\sqrt{7}}{4} r \sin\theta_m$ from any boundary facet on which p does not lie. Here θ_m is the measure of an angle such that all dihedral angles of the input boundary are within θ_m and $180° - \theta_m$.

Proof: If p is an inner point, we already know p is at least r units away from every boundary facet. By the choice of r, any point on a boundary facet is at least r units away from any other nonadjacent facet. We prove that if p lies on a boundary facet but not on a boundary edge, it is at a distance of at least $\frac{\sqrt{7}}{4} r \sin\theta_m$ from all adjacent facets. Let p lie on f_i and let f_j be any facet adjacent to f_i. In Lemma

5.3.2, we proved that the distance of p from any line supporting an edge of the facet f_i is at least $\frac{\sqrt{7}}{4}r$. Let l be the distance of p from the line where f_i and f_j meet. The distance d of p from f_j is given by $d = l\sin\theta$ where θ is the dihedral angle between f_i and f_j. Putting the minimum value of l and θ gives the lower bound on d. Thus, the distance of a point from any facet that does not contain it is at least $d_{\min} = \min(r, \frac{\sqrt{7}}{4}r\sin\theta_m) = \frac{\sqrt{7}}{4}r\sin\theta_m$. ♣

5.3.2 Qualities of Tetrahedra

Definition 5.3.1 A tetrahedron in the final triangulation is said to have a *good circumcenter* if the center of its circumscribing sphere lies inside or on the boundary (convex hull boundary). Conversely, a tetrahedron is said to have a *bad circumcenter* if the center of its circumscribing sphere lies outside the boundary.

We classify the tetrahedra with bad circumcenters into two classes, namely class A and class B.

Definition 5.3.2 A tetrahedron t with a bad circumcenter is called a class A tetrahedron if it satisfies the following property. There exists a facet f intersected by the circumscribing sphere s of t in such a way that the foot of the perpendicular dropped from the center of s on the supporting plane of f lies inside f. Any other tetrahedron with a bad circumcenter is called a class B tetrahedron. See figure 5.5 and figure 5.6.

Assuming lower and upper bounds on the dihedral angles between adjacent boundary facets, we can prove that all tetrahedra produced by *3D-Tri* cannot be in category(i) or category(ii). Although we cannot avoid category(iii) tetrahedra, occurrences of them in practice are rare [Bak89]. Finally, in most of the cases, these tetrahedra can often be avoided by introducing a suitable point inside the circumscribing sphere; see [Bak89]. In what follows, we assume that all dihedral angles between adjacent boundary facets are greater than θ_m and less than $180° - \theta_m$.

Lemma 5.3.5 No tetrahedron with good circumcenter can be in category(i) or category(ii).

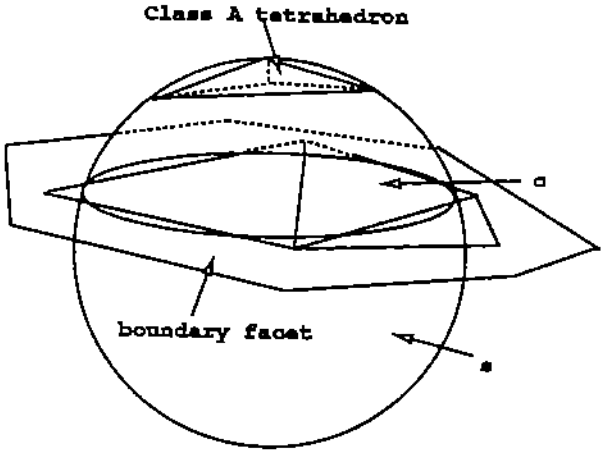


Figure 5.5 Class A tetrahedron.

Proof: All tetrahedra in the final triangulation having good circumcenters must have circumscribing spheres with radii less than $2r$, because otherwise these circumcenters would have been introduced as new points. Hence, all these tetrahedra have edges of length less than $4r$. By Lemma 5.3.3, all edges have lengths greater than $\min(r, \frac{\sqrt{7}}{2} r \sin\frac{\theta_m}{2})$. Thus, κ for these tetrahedra can be at most $\max(4, \frac{8}{\sqrt{7}\sin\frac{\theta_m}{2}})$. Assuming a lower bound on the dihedral angles of the input boundary, we get κ for these tetrahedra to be of $O(1)$ which violates the condition for category(i) tetrahedra. Further, ω for these tetrahedra can be at most $\max(2, \frac{4}{\sqrt{7}\sin\frac{\theta_m}{2}}) = O(1)$ which prohibits them to be in category(ii). ♣

Lemma 5.3.6 No class A tetrahedron can be in category(i) or category(ii).

Proof: Let t be a class A tetrahedron with the circumscribing sphere s. By the definition of class A tetrahedron, there exists a boundary facet f such that the foot of the perpendicular dropped from the center of s on the supporting plane of f lies inside f. Let c be the circle of intersection of S with the supporting plane of f. By Lemma 5.3.4. a vertex p of t that does not lie on f must be at a distance of at least $\frac{\sqrt{7}}{4} r \sin\theta_m$ from f where θ_m is defined as before. The center of the circle c lies inside f. Thus, the center must lie inside the triangulation T of f produced by

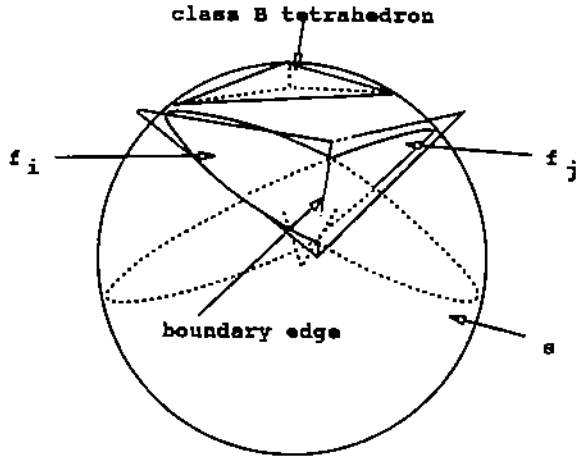


Figure 5.6 Class B tetrahedron.

the algorithm *2D-Tri*. Further, c must be an empty circle since s does not include any point of f inside it. See figure 5.5. By property 2, all triangles of T have circumscribing circles of radii less than r. Hence, according to Lemma 5.2.1, c must have a radius less than or equal to r. The vertex p lying on s must be at a distance of at least $\frac{\sqrt{\tau}}{4} r \sin\theta_m$ from c. Further, the vertex p and the center of s lie on the opposite sides of c. This implies that c is at a distance of at least $\frac{\sqrt{\tau}}{4} r \sin\theta_m$ from its nearest pole. Thus, according to Lemma 5.2.2, s must have a radius less than or equal to $k_1 r$ where $k_1 = (\frac{\sqrt{\tau}\sin\theta_m}{8} + \frac{2}{\sqrt{\tau}\sin\theta_m})$. This puts an upper bound of $2k_1 r$ on the lengths of the edges of t_i. By Lemma 5.3.2, all edges of t_i are greater than $k_2 r$ where $k_2 = \min(1, \frac{\sqrt{\tau}}{2}\sin\frac{\theta_m}{2})$. Hence, ω, κ for t_i are $O(1)$ assuming a lower bound on θ_m (A lower bound on θ_m puts lower and upper bounds on the dihedral angles between adjacent boundary facets). This prohibits it to be in category(i) or category(ii). ♣

Lemma 5.3.7 Let t be a class B tetrahedron with the circumscribing sphere s. There must exist two boundary facets f_i, f_j intersected by s with the following criterion: Let c be any circle drawn on s which is normal to the line where f_i, f_j meet. The feet of the perpendiculars dropped from the center of c on the supporting planes P_i and P_j of f_i and f_j lie outside the line segments $c \cap f_i$, $c \cap f_j$.

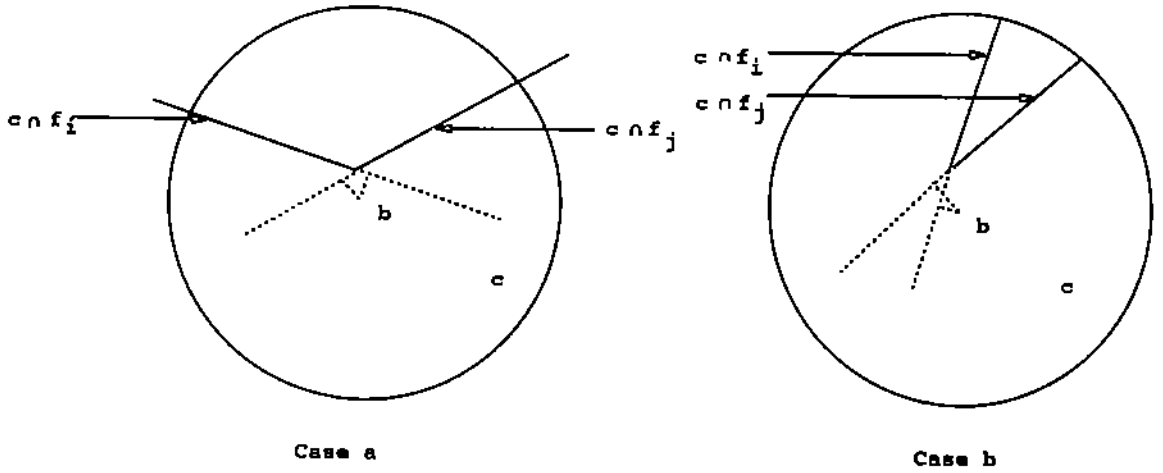


Figure 5.7 Cases of Lemma 5.3.7.

Proof: Consider a boundary facet f_i that has the convex hull and the center of s on opposite sides. Since t has a bad circumcenter, such a facet always exists. Consider any other facet f_j sharing an edge with f_i that has been intersected by s. Drop perpendiculars from the center of s on the supporting planes of f_i and f_j. The feet of these perpendiculars lie outside f_i, f_j since t is a class B tetrahedron. Consider the great circle c' of s whose supporting plane is normal to the edge shared by f_i and f_j. The feet of the perpendiculars dropped from the center of s on the supporting planes P_i and P_j of f_i and f_j cannot lie on the line segments $c' \cap f_i$ and $c' \cap f_j$. Two different cases are shown in figure 5.7. This immediately implies that the condition stated in Lemma 5.3.7 is true for any circle c on s that has a supporting plane parallel to that of c'.♣

Lemma 5.3.8 No class B tetrahedron can be in category(i) or category(ii).

Proof: Let t be a class B tetrahedron. Let the circumscribing sphere s of t intersect the boundary edge e shared by the facets f_i and f_j which satisfy the criterion as stated in Lemma 5.3.7. The endpoints of the edge segment e_n on e which is intersected by

s cannot be inside s. Let w, y be the points where s intersects e_n. Further, let a and R denote the center and radius of s respectively.

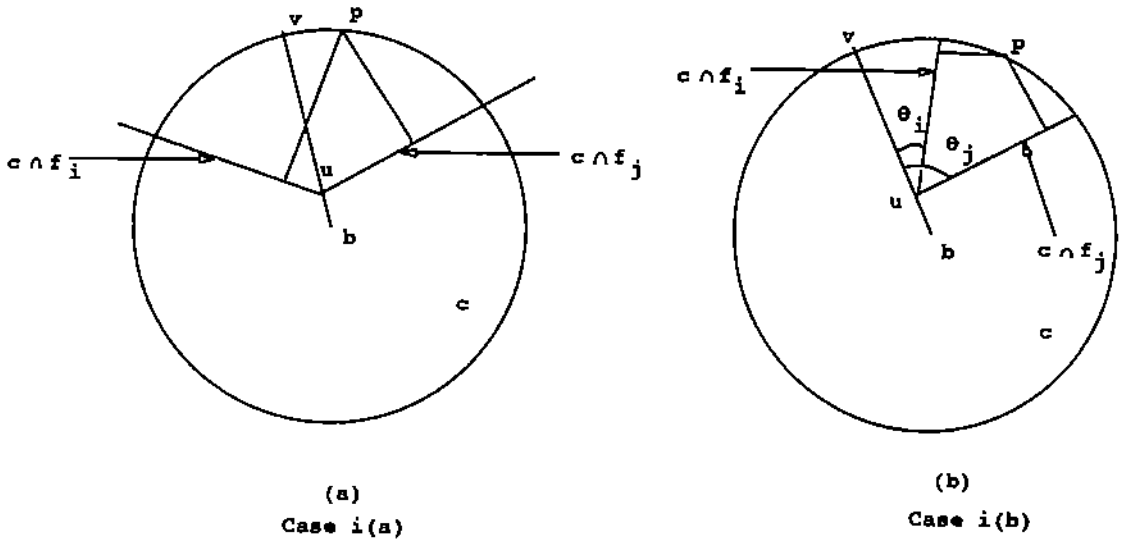


Figure 5.8 Case(i) of Lemma 5.3.8.

Case(i): The tetrahedron t has a vertex p which lies neither on the facet f_i nor on the facet f_j. Consider the circle c on s whose supporting plane is perpendicular to e_n and which passes through p. Let R' be the radius of c. Join the center b of c with the point u where c meets e_n. Extend the line $\overline{bu}$ beyond u until it intersects the boundary of c at v as shown in figure 5.8. Let $|\overline{bu}| = x$. Certainly, $|\overline{uv}| = R' - x$. Let d denote the minimum distance of p from the two facets f_i and f_j. There are two subcases as shown in figure 5.8. In subcase $i(a)$, the center of c lies in the sides of the planes containing f_i, f_j which are opposite to those containing the convex hull. It is not difficult to see that, in this subcase, $d \leq |\overline{uv}| = R' - x$. Since, $R \geq R'$, we have $d \leq R - x$. To estimate a lower bound on x, drop a perpendicular $\overline{az}$ from the center a of s on e_n. This perpendicular has the same length as $\overline{bu}$. Consider the triangle $\triangle awy$. We observe that $|\overline{az}| = \sqrt{R^2 - \frac{|\overline{wy}|^2}{4}}$. Since e_n can have a length of at most

$1.5r$, we have $x = |\overline{az}| \geq \sqrt{R^2 - \frac{9r^2}{16}}$. Thus, $d \leq R - \sqrt{R^2 - \frac{9r^2}{16}}$. We already know $d \geq \frac{\sqrt{7}}{4} r \sin\theta_m$ (Lemma 5.3.4). Hence,

$$\begin{aligned} \frac{\sqrt{7}r}{4} \sin\theta_m &\leq R - \sqrt{R^2 - \frac{9r^2}{16}}, \\ R &\leq \frac{7\sin^2\theta_m + 9}{8\sqrt{7}\sin\theta_m} r. \end{aligned}$$

Now, consider the subcase $i(b)$. In this subcase, one of the supporting planes of f_i and f_j has the center of c and the convex hull on its opposite sides and the other one has them on same side. Without loss of generality, assume that the supporting plane of f_i has them on same side as shown in figure 5.8(b). The line segments $c \cap f_i$ and $c \cap f_j$ make angles less than equal to 90° with $\overline{uv}$. Otherwise, f_i, f_j do not satisfy the criterion as stated in Lemma 5.3.7. In this subcase, we have $d \leq R - x$ since the distance of v from the supporting plane of f_j is greater than that of p from the same plane. Thus, in both subcases $i(a)$ and $i(b)$, we have,

$$R \leq \frac{7\sin^2\theta_m + 9}{8\sqrt{7}\sin\theta_m} r.$$

Case(ii): All vertices of the tetrahedron t lie either on f_i or on f_j. This immediately implies that one of the vertices of t lies on f_i but not on f_j and another on f_j but not on f_i. Consider the vertex p_i lying on f_i but not on f_j. Let c be the circle passing through p_i with the supporting plane being perpendicular to e_n. As in the previous case, let b be the center of c, u be the foot of the perpendicular dropped from b to e_n, and v be the point of intersection of the line $\overline{bu}$ and the circle c such that u is in between b and v. Again, we have two subcases as shown in figure 5.9. Consider the subcase $ii(a)$. We have $|\overline{p_iu}| \leq \frac{|\overline{uv}|}{\cos\theta_i}$ where θ_i is the angle between $\overline{p_iu}$ and $\overline{uv}$. We proved in Lemma 5.3.2 that the distance of any point on a boundary facet that does not lie on any of its edges is at least $\frac{\sqrt{7}}{4}r$ away from any of its edges. Thus, $|\overline{p_iu}| \geq \frac{\sqrt{7}}{4}r$. Hence, $\frac{\sqrt{7}}{4}r \leq \frac{R'-x}{\cos\theta_i} \leq \frac{R-x}{\cos\theta_i}$ where $x = |\overline{bu}|$. Similarly, considering the vertex p_j of t lying on f_j but not on f_i we can prove that $\frac{\sqrt{7}}{4}r \leq \frac{R-x}{\cos\theta_j}$ where θ_j is the angle between $f_j \cap c$ and $\overline{uv}$. The angle $\theta = \theta_i + \theta_j$ is the dihedral angle between f_i

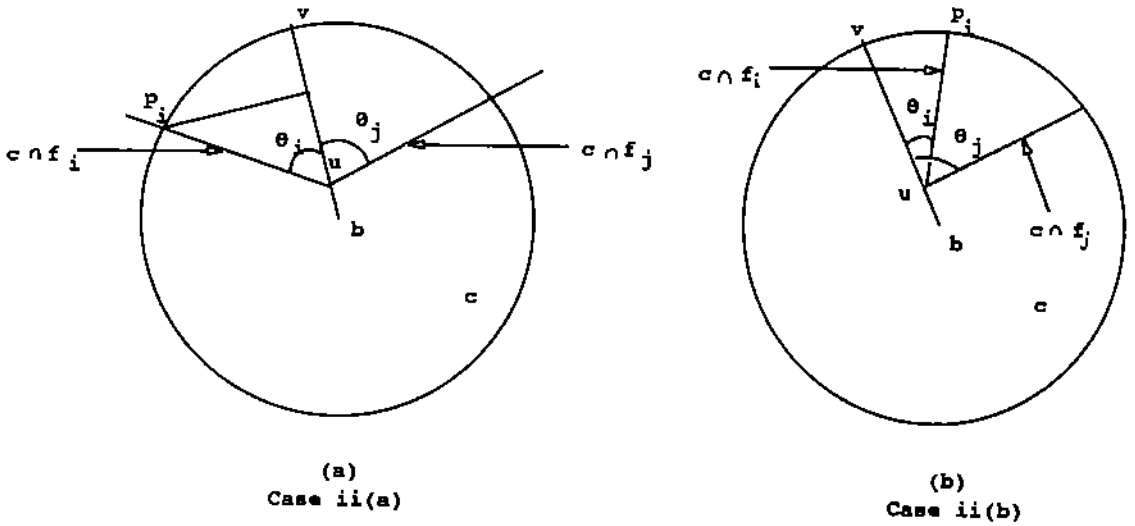


Figure 5.9 Case(ii) of Lemma 5.3.8.

and f_j. Since one of θ_i, θ_j is less than or equal to 90° and the cos function decreases monotonically from 0° to 90°, we have $\frac{\sqrt{7}}{4}r \leq \frac{R-x}{\cos\frac{\theta}{2}}$. By the same argument as in case(i), we get $x \geq \sqrt{R^2 - \frac{9}{16}r^2}$. Hence,

$$\begin{aligned} \frac{\sqrt{7}}{4}r &\leq \frac{R - \sqrt{R^2 - \frac{9}{16}r^2}}{\cos\frac{\theta}{2}} \\ R &\leq \frac{\frac{9}{16}r^2 + \frac{7}{16}r^2\cos^2\frac{\theta}{2}}{\frac{\sqrt{7}}{2}r\cos\frac{\theta}{2}}. \end{aligned}$$

Assuming an upper bound on $\theta \leq (180° - \theta_m)$ we have

$$R \leq \frac{7\sin^2\frac{\theta_m}{2} + 9}{8\sqrt{7}\sin\frac{\theta_m}{2}}r.$$

Now, consider the subcase $ii(b)$. The angles between $\overline{uv}$ and the line segments $c \cap f_i$ and $c \cap f_j$ are less than 90° since otherwise f_i, f_j violate the condition of Lemma 5.3.7. Without loss of generality assume that $\theta_i < \theta_j$. The distance between v and $c \cap f_j$ is greater than that between p_i and $c \cap f_j$. This implies $d \leq R - x$ giving the same upper bound on R as we derived in case(i).

Thus, all class B tetrahedra have a circumscribing sphere of radius $k_1 r$ where $k_1 = O(1)$ assuming lower and upper bounds on the dihedral angles between adjacent boundary facets. This with the fact that edges of all tetrahedra have lengths greater than $k_2 r$ where $k_2 = O(1)$ (recall Lemma 5.3.3), makes ω and κ of these tetrahedra to be of $O(1)$ and thus prohibits them to be in category(i) or category(ii). ♣
The following Theorem is immediated from Lemmas 5.3.5, 5.3.6, and 5.3.8.

Theorem 5.3.1 Algorithm *3D-Tri* triangulates the convex hull of a three dimensional point set with the guarantee that the tetrahedra of type(i) through type(iv) are never generated assuming lower and upper bounds on the dihedral angles between adjacent boundary facets of the convex hull.

5.3.3 Complexity

Algorithm *3D-Tri* produces tetrahedra whose edges are greater than l_{min} as defined in Lemma 5.3.3. The circumscribing sphere of each such tetrahedron must have a volume of $\Omega(l_{min}^3)$. Let V be the volume of the convex hull of the given point set. Let n and n_o be the number of points present in the input and output respectively. Certainly, $n_o = O(\frac{V}{l_{min}^3})$. Consider a triangulation T of the input point set where $|T| = O(n)$. Such a triangulation always exists; see [EPW86]. Let L be the largest edge length in T. All tetrahedra in T have a volume less than L^3. Thus, $V = O(nL^3)$. This gives an upper bound of $O(n\frac{L^3}{l_{min}^3})$ on n_o. Putting $A = \frac{L}{l_{min}}$, we have $n_o = O(nA^3)$. The quantity A captures the notion of how badly distributed the input point set is.

The basis of *3D-Tri* is the incremental Delaunay triangulation algorithm. We use Watson's algorithm [Wat81] for this purpose. In this algorithm, all tetrahedra whose circumscribing spheres contain the inserted point inside are removed. To produce new tetrahedra, the new point is connected to the triangles present on the boundary of the union of all removed tetrahedra. In *3D-Tri*, we introduce the circumcenters of tetrahedra that satisfy specific properties as new points. We maintain a queue of all such tetrahedra throughout the algorithm. This queue supports deletion and addition of an element in logarithmic time. Thus, we can pick a tetrahedron t_i whose

circumcenter is to be added in $O(\log n_o)$ time. We can determine all tetrahedra to be removed and to be added in $O(n_o)$ time once we have chosen t_i. This is due to the fact that there are at most $O(n_o)$ tetrahedra to be removed and added for each insertion and they form a connected component together. Updating the queue for these removed and added tetrahedra takes $O(n_o \log n_o)$ time which dominates the time complexity for a single insertion. Thus, inserting all valid circumcenters takes $O(n_o^2 \log n_o)$ time. Algorithm *2D-Tri* cannot take more than $O(n_o^2)$ time [Dey90]. Hence, *3D-Tri* takes $O(n_o^2 \log n_o) = O(n^2 A^6 \log n \log A)$ time and $O(n_o) = O(nA^3)$ space.

5.3.4 Implementation Issues

We consider the problem of numerical errors under finite precision arithmetic while implementing the algorithm *3D-Tri*. For robust implementation of *3D-Tri*, we need a robust algorithm for computing the Delaunay triangulations in 3D. In the next section, we present a type-2 robust algorithm for this problem. To triangulate the facets robustly, we use the type-2 robust algorithm of [SI89a] for 2D voronoi diagram and use its dual.

With numerical errors, the computed points on the boundary facets may not be exactly coplanar, and without proper care they may form very thin tetrahedra. While constructing the triangulation of the point set obtained by triangulating all boundary facets, we take into account the topological constraint that the points generated on a boundary facet are coplanar. We have implemented our good triangulation algorithm on SUN workstations in AKCL. An example where a convex polyhedron is triangulated is shown in Figure 5.15. For clarity, we show only the triangulations on the facets.

5.4 Robust Delaunay Triangulations

We give a type-2 robust algorithm for three dimensional Delaunay triangulations. Recall that a type-2 robust algorithm must have the following properties. It must

not fail–the "non-failing" property; it should give true output under infinite precision and the output should satisfy certain essential topological properties under finite precision–the "convergence" property. Although in type-2 robust algorithms, it is not essential to use thresholds in numeric computations as long as consistent topological inferences can be drawn without them, we use such thresholds in attempt to produce an output close to the true one. Actually, it is our hope that with these thresholds, these algorithms can become type-4 robust, though we cannot prove it.

In this approach a typical segment of a robust program looks as follows.

value=Numeric-computation.
If absolute(value) $\geq \delta$ then A else B

The quantity δ acts as a threshold for safe computations and is proportional to the precision as we have seen in Section 3.4. It becomes zero with infinite precision. Thus, under infinite precision the action B is never taken and the output is guided by the action A. Let A' be the action that should be taken by the algorithm under infinite precision. The action A is designed in such a way that it becomes equivalent to A' w.r.t. the input-output relation under infinite precision. Further, the actions A and B are designed in such a way that they guarantee the desired topological properties of the computed data and never contradicts the previous decisions. This, in turn, guarantees the "non-failing" property of the program.

Design of A and B is dependent on the desired topological properties of the output. For triangulation of a point set, we use the conditions of the *topological triangulations* as basis of our topological validity tests.

Let $G = (V, E)$ be a connected graph with the vertex set V and the edge set E. A face consists of a cycle of alternating vertices and edges. A 3-cell consists of a collection of faces where each edge is incident on two faces.

Definition 5.4.1 A *combinatorial augmentation* of G is a tuple $C_G = (V, E, F, T)$ where F is a set of faces and T is a set of 3-cells. Each vertex and edge is incident on at least one face. Each face is incident on at least one 3-cell. A *simplicial combinatorial*

augmentation is a combinatorial augmentation in which each face consists of a cycle of three vertices and edges and each 3-cell consists of four faces.

An embedding of a combinatorial augmentation is a mapping $h : V - S$ where S is the point set in three dimensional space bounded by a closed oriented manifold. The mapping extends to edges, faces, and 3-cells. If v_1, v_2 are endpoints of an edge e, then $h(e)$ is an open curve segment joining $h(v_1)$ and $h(v_2)$. If the image of a face cycle of face f is a simple closed curve, then $h(f)$ is an open surface bounded by the closed curve. If a 3-cell t consists of faces $f_1, f_2, .., f_k$, then $h(t)$ is the open three dimensional region bounded by $h(f_1), h(f_2), ..., h(f_k)$. An embedding is planar in 3D if h is pairwise disjoint for vertices, edges, faces and 3-cells.

5.4.1 Topological Triangulations

Definition 5.4.2 A 3D *topological triangulation* is a connected graph G that has a planar embedding in a 3D space $S \subseteq R^3$ where S is bounded by a closed oriented manifold and the embedding gives a simplicial decomposition (with simplicial complex) of S. If the surface of S is homeomorphic to that of a sphere, G is called to be a 3D genus zero topological triangulation. The tetrahedra produced by the simplicial decomposition may have curved edges and curved faces.

In the rest of this chapter, we refer to the 3D topological triangulations simply as the topological triangulations and the simplicial decompositions with simplicial complexes as the simplicial decompositions. From the definition of the topological triangulations, it is clear that the underlying graph of any triangulation of a point set in 3D is a genus zero topological triangulation. This essential topological property of 3D triangulations is used to design a type-2 robust algorithm for 3D Delaunay triangulations. The underlying graph of the output computed by the algorithm satisfies certain essential properties of a genus zero topological triangulation.

5.4.2 Orientations

The orientation of a face $f = (v_1, v_2, v_3)$ can be specified by the cyclic order on its vertices. There are only two such unique orders. An oriented face is a part of a oriented manifold and thus has positive and negative sides. One particular order on its vertices fixes its positive and negative sides. The side from which the order is viewed as clockwise is designated as the negative side.

Definition 5.4.3 Two oriented faces match if the shared edges (if any) are directed in opposite directions in them.

A tetrahedron has its faces oriented in such a way that they match to each other. In Figure 5.10, the faces f_1 and f_2 of a tetrahedron are oriented to match each other. A particular orientation of a face fixes the orientations of all other faces

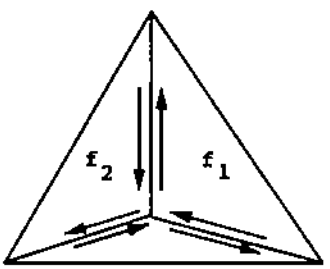


Figure 5.10 A tetrahedron with oriented faces.

of a tetrahedron. Thus, a tetrahedron has two unique orientations. To orient a tetrahedron unambiguously, we orient a face in such a way that the fourth vertex lies on its positive side.

Definition 5.4.4 Let v be a vertex of a graph G. Let V' be the set of vertices adjacent to v. Consider v not to be included in V'. The subgraph $G' = (V', E')$ where E' is the set of edges whose both endpoints are in V' is called the *star* of v and is denoted as star(v).

Definition 5.4.5 A graph is called *planar triangular* if and only if it has a planar embedding with triangular faces except possibly the outer face. All faces including the outerface must be simple.

In the rest of this chapter, all stars are meant to be planar triangular. The star

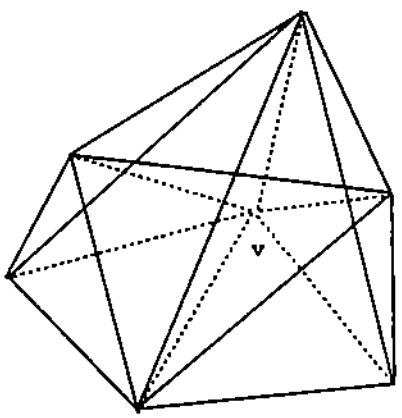


Figure 5.11 The star of a vertex.

of a vertex v is shown in Figure 5.11. Edges drawn with the solid lines are the edges of star(v). There are only two unique planar embeddings of such graphs w.r.t. the orientations of the faces. One is the mirror image of the other. A particular orientation of a face fixes the orientations of all other faces. Thus, we can specify the orientation of a planar triangular star by the orientation of any of its faces.

Definition 5.4.6 A star with an orientation *matches* with the other if and only if the shared faces have opposite orientations on them. With this definition, two oriented stars match vacuously if they do not share any face or edge.

In Figure 5.12, stars of v_1 and v_2 (consisting of f_1, f_2) match each other since the shared faces have opposite orientations.

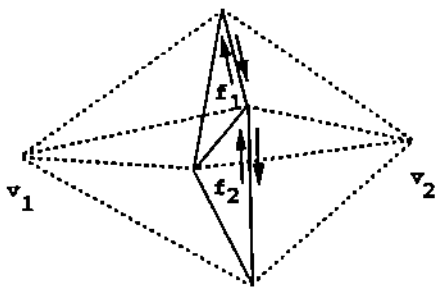


Figure 5.12 Matching of two stars.

5.4.3 Properties of Topological Triangulations

Lemma 5.4.1 A connected graph G is a topological triangulation only if the following conditions are satisfied.

1. C1: For each vertex $v \epsilon V$, star(v) is planar triangular.
2. C2: Any triangular face appears in the stars of at most two vertices.
3. C3: It is possible to orient the stars of all vertices simultaneously so that each one matches with the other.

Proof: We show that the underlying graph of a simplicial decomposition of a 3D space bounded by a closed oriented manifold satisfies conditions C1, C2 and C3.

Consider a vertex v_i in the simplicial decomposition. The underlying graph constituted by the bases of all tetrahedra with apex v_i form the star of v_i. This graph with bases as triangular faces can be embedded on an oriented manifold that is homeomorphic to a plane. Thus, it can be embedded on a plane with those triangular faces except possibly one face (C1). Each triangular face is incident on at most two tetrahedra and thus appears in at most two stars (C2). The stars with orientations of the faces on corresponding tetrahedra must match each other since the faces with these orientations match each other in the simplicial decomposition (C3). ♣

Lemma 5.4.2 A topological triangulation G has zero genus only if the following conditions are satisfied.

1. C4: The space S of its embedding has one connected surface.
2. C5: $|V'| - |E'| + |F'| = 2$ where V', E', F' are the vertices, edges, faces present on the surface.

Proof: Since G is a topological triangulation, it has an embedding in R^3 that gives a simplicial decomposition of a space S bounded by the closed oriented manifolds. To be homeomorphic to a sphere, there must be one connected surface. Any space S with one connected oriented surface must be homeomorphic to torii with handles [GT87]. Any simplicial decomposition of such a space must satisfy $|V'| - |E'| + |F'| = 2 - g$ where g is number of handles and V', E', F' are the set of vertices, edges, faces on the surface. For the surface of S to be homeomorphic to that of a sphere, $|V'|, |E'|, |F'|$ must satisfy the above equation with $g = 0$.

5.4.4 Incremental Robust Delaunay Triangulation

We observe that the underlying graph of a Delaunay triangulation (in fact any triangulation) in 3D is a genus zero topological triangulation. We use the conditions C1 through C5 to design a type-2 robust algorithm for the Delaunay triangulation of a point set in 3D.

This robust algorithm is obtained by modifying the well known incremental algorithm of Watson [Wat81]. In this incremental approach, the Delaunay triangulation of the current point set is modified locally to incorporate a new point. In this algorithm, each face is maintained as two oriented faces with opposite orientations on its cycle. The algorithm is given below.

5.4.5 The Algorithm with Exact Arithmetic

Algorithm DT-Exact

Input: A point set $P = \{p_1, p_2, ..., p_k\}$ in three dimensions.

Step 1. Construct the tetrahedron $t_1 = (p_1, p_2, p_3, p_4)$ and set $T = \{t_1\}$.

Step 2. For each point p_i, $i = 5, 6, ..k$, carry out the following steps.

Step 2.1. Find out the faces (if any) on the boundary of T that contain the point p_i on the side opposite to that containing T. Let B be the boundary constituted by these triangular faces.

Step 2.2. Find out the set W of tetrahedra whose circumscribing spheres contain the point p_i inside or on it. Let $B' = bd(\mathrm{W})$ where bd(W) denotes the boundary of the union of the tetrahedra in W.

Step 2.3. Compute $B'' = (B \cup B') - (B \cap B')$.

Step 2.4. Delete tetrahedra in W from T. Add tetrahedra to T that are created by taking the point p_i as the apex and the triangular faces of B'' as bases.

In the above algorithm, numerical computations are carried out at two places. In Step 2.1, we need numerical computations to determine whether the point p_i is inside the circumscribing sphere of a tetrahedron or not. Let the tetrahedron $t = (p_1, p_2, p_3, p_4)$ have an oriented face $f = (p_1, p_2, p_3)$. To determine the location of p_i w.r.t. the circumscribing sphere s of t, we compute the determinant

$$C(t, p_i) = \begin{bmatrix} x_1 & y_1 & z_1 & x_1^2 + y_1^2 + z_1^2 & 1 \\ x_2 & y_2 & z_2 & x_2^2 + y_2^2 + z_2^2 & 1 \\ x_3 & y_3 & z_3 & x_3^2 + y_3^2 + z_3^2 & 1 \\ x_4 & y_4 & z_4 & x_4^2 + y_4^2 + z_4^2 & 1 \\ x_i & y_i & z_i & x_i^2 + y_i^2 + z_i^2 & 1 \end{bmatrix}.$$

Here x_i, y_i, z_i are the coordinates of the point p_i. The location of the point p_i w.r.t. s is determined by the sign of $C(t, p_i)$. In Step 2.2, we need numerical computations to determine the side of a face that contains the point p_i. To classify the point p_i w.r.t. an oriented face $f = (p_1, p_2, p_3)$, we compute

$$H(f, p_i) = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_i & y_i & z_i & 1 \end{bmatrix}.$$

The location of the point p_i w.r.t. the face f is determined by the sign of $H(f, p_i)$. With numerical errors, we cannot rely on the signs of $C(t, p_i)$, $H(f, p_i)$ when p_i is very close to s and f respectively.

5.4.6 The Algorithm under Finite Precision Computations

With erroneous numerical computations, the boundary B may have more than one connected component due to numerical errors and $B \cap B'$ may be empty even though B and B' are not. In these cases, the boundary B'' has more than one connected component. This implies that star(p_i) is disconnected which violates the condition for topological triangulations. Further, the underlying graph of B' may not be planar triangular making star(p_i) not to be planar triangular which violates a necessary condition for topological triangulations.

Both these problems, however, go off if we carry out a careful depth first search for the faces in B and the tetrahedra in W. Let B_i be the boundary constituted by the faces that have been decided to be in B so far. We maintain a list (Blist) of faces that are in B_i and have at least one adjacent face that is not in B_i. By adjacent faces we mean only those faces that are adjacent by an edge. We expand B_i by picking a face from this list and testing the unexplored adjacent faces for their inclusion in B. Blist and B_i are updated accordingly. This guarantees that the final boundary B is connected and planar triangular.

In the exact algorithm, it is possible that the boundary of W is disconnected. In that case, each component of bd(W) must have a nonempty intersection with B in such a way that the final boundary B'' has only one component. With numerical errors, we compute a connected component of W as follows and look for other possible components. We collect tetrahedra in one connected component of W in a depth first manner. Let W_i be the set of tetrahedra that have been decided to be in one component of W so far. We maintain a list (Tlist) of tetrahedra that are in W_i and have at least one adjacent tetrahedron that is not in W_i. By adjacent tetrahedra we mean those tetrahedra that are adjacent by a face. To continue the search for

new tetrahedra in one component of W, we pick a tetrahedron from this list and test whether the unexplored adjacent tetrahedra are member of W or not. We update Tlist and W_i accordingly. Computing W this way ensures that the boundary of one component of W is never disconnected. Of course, care should be taken to ensure that each component of bd(W) remains planar triangular.

Algorithm DT-Robust

Input: A point set $P = \{p_1, p_2, ..., p_k\}$ in three dimensions.

Step 1. Construct the tetrahedron $t_1 = (p_1, p_2, p_3, p_4)$ and set $T = \{t_1\}$.

Step 2. For each point p_i in P, carry out the following steps.

Step 2.1. Find out a face f (if any) on the boundary of T such that $H(f, p_i) \geq 0$. Initialize $B = \{f\}$ and put f into Blist. Repeat steps 2.1.1 and 2.1.2 until no more face can be added to B.

Step 2.1.1. Pick the face f from Blist that is adjacent to a face f' satisfying the following properties.

1. The face f' is not in B.
2. $H(f', p_i) \geq 0$.
3. Adding f' to B does not destroy its planar triangular property.

Step 2.1.2. Set $B = B \cup f'$ and put f' into Blist. If all adjacent faces of f are in B, delete it from Blist.

Step 2.2. If B is not empty, check for a tetrahedron t adjacent to a face in B for which $C(t, p_i) \geq 0$. If such a tetrahedron t is found, put t into Tlist and set $W = \{t\}$. In case B is empty, check for any tetrahedron t that satisfies $C(t, p_i) \geq 0$. In case no such tetrahedron is found, pick the tetrahedron t for which the value of $C(t, p_i)$ is the largest. Set $W = \{t\}$ and put t into Tlist. Repeat steps 2.2.1, 2.2.2 and 2.2.3 until

no more tetrahedron can be added to W.

Step 2.2.1. If at any point of iteration, Tlist is empty, check for a not yet visited tetrahedron t adjacent to a face in B for which $C(t, p_i) \geq 0$. If such a tetrahedron is found, put t into the Tlist and set $W = W \cup t$.

Step 2.2.2. Pick a tetrahedron t from Tlist that is adjacent to a tetrahedron t' satisfying the following properties.

1. The tetrahedron t' is not in W.
2. $C(t', p_i) \geq 0$.
3. There is no vertex in t' for which all other incident tetrahedra have been decided to be in W (to prevent isolated vertices).
4. Adding t' to W does not destroy the planar triangular property of bd(W).
5. If t' has a face in B, then that face is adjacent to other faces in B that are also decided to be in bd(W) (to prevent more than one non triangular faces in B'').

Step 2.2.3. Set $W = W \cup t'$. Put t' into Tlist. If all adjacent tetrahedra of t are in W, delete it from Tlist.

Step 2.3. Compute $B'' = (B \cup B') - (B \cap B')$.

Step 2.4. Delete tetrahedra in W from T. Add tetrahedra to T that are created by taking the point p_i as the apex and the triangular faces of B'' as bases.

5.4.7 Degree-2 robustness of DT-Robust

Let T_1 denote the triangulation which consists of the single tetrahedron $t_1 = (p_1, p_2, p_3, p_4)$ and T_i $(i = 2, .., k-3)$ denote the triangulation obtained by adding p_{3+i} to T_{i-1} at the ith stage.

Lemma 5.4.3 Let T be a triangulation constructed by the algorithm DT-Robust at any stage. The underlying graph G of T satisfies C1 and C2.

Proof: We prove it by induction. Definitely, C1 and C2 are true for the first triangulation T_1 which consists of a single tetrahedron. We assume that the triangulation T_i satisfies C1 and C2 and prove that the triangulation T_{i+1} satisfies them too.

Removing tetrahedra in W and the faces that are in $B \cap B'$ affects only the stars of the vertices in B''. Note that an edge is removed only when all faces adjacent to it are removed. An internal face is removed when both tetrahedra incident on it are removed. A face on the boundary is removed if it appears both on B and B'. Consider any vertex v on B''. Consider the planar embedding of star(v) that is matched with other stars.

Consider the case when B' is not empty. Since tetrahedra in one component of W are collected through face adjacency and $B \cap B'$ is kept connected for each component, removal of edges to create B'', in effect, removes a connected subgraph from star(v).

This creates either a hole in the embedding of star(v) or a "dip" in its boundary. Figure 5.13 shows a hole and a "dip" created by removing connected subgraphs from the planar triangular graphs of the triangulations shown in Figure 1.2. Joining p_i to the faces in B'' has the following effects on the star(v). In case a hole is created, p_i is joined to the vertices of the hole. Otherwise, p_i is connected to the consecutive edges on the modified boundary of star(v). In both cases, star(v) remains to be planar triangular.

Consider the other case when B' is empty giving $B'' = B$. In this case, nothing is deleted from star(v). The new vertex p_i is connected to the consecutive vertices

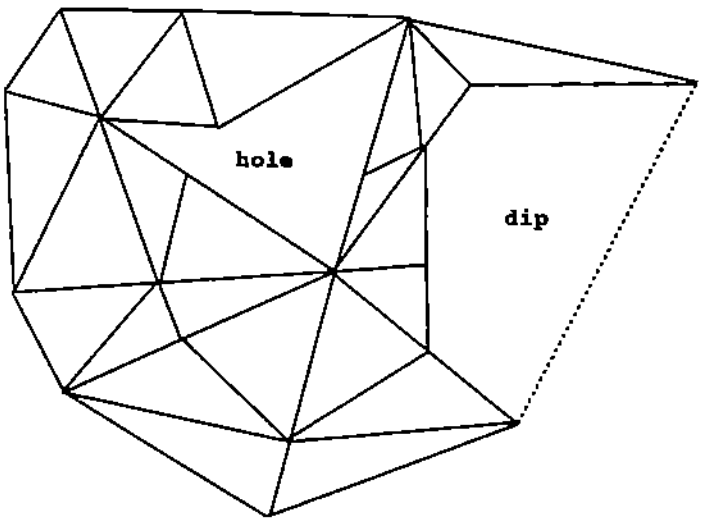


Figure 5.13 A hole and a "dip" in a star embedding.

on the outer face of star(v). Thus, star(v) remains to be planar triangular. Finally, since B'' is planar triangular star(p_i) is planar triangular.

New faces created by joining p_i to the faces on B'' appears in at most two tetrahedra and thus appears in the stars of at most two vertices.

Lemma 5.4.4 Let T be the triangulation constructed by the algorithm DT-Robust at any stage. The underlying graph G of T satisfies C3.

Proof: We have to prove that all stars match with respect to some orientations. Consider the set of tetrahedra incident on a vertex v of T. The underlying graph of the structure formed by the bases of the tetrahedra with apex v constitutes star(v). Let the orientation of star(v) be specified by the orientations of these bases (faces) on corresponding tetrahedra. We prove by induction that all stars in G match with these orientations.

Certainly, the hypothesis is true for the first triangulation T_1 which consists of a single tetrahedron. Let it be true for the triangulation T_i at the ith stage. Let f be a face on B''. While making the tetrahedra with f as the base and p_i as the apex, the face f is given the orientation as follows. If f is incident on bd(W), it is oriented in

the same way as it is in the tetrahedron $t \epsilon W$. If f is on the boundary of T_i incident on the tetrahedron $t' \epsilon T_i$, the face f is given the orientation that is opposite to its orientation in t'. With these orientations, the faces in B'' match to each other and any face in B'' which appears on two tetrahedra gets opposite orientations. Thus, star(p_i) matches with other stars with the orientations of the faces of corresponding tetrahedra. Insertion of p_i affects the stars of the vertices on B''. New faces are generated by joining p_i to the edges on B''. Let f' be such a new face incident on a tetrahedron t'' with p_i as the apex and $f'' \epsilon B''$ as the base. Let f''' be adjacent to f'' on B'' by the edge on which f' is incident. The face f' appears on another tetrahedron t''' that has p_i as the apex and f''' as the base. Since f'' and f''' match each other, f' must get opposite orientations on the two tetrahedra t'' and t'''. Thus, all faces get opposite orientations on adjacent tetrahedra implying the matching of all stars.

Lemma 5.4.5 Let T be the triangulation constructed by the algorithm DT-robust at any stage. The underlying graph G satisfies C4 and C5.

Proof: In Lemma 5.4.3 and 5.4.4, we considered the oriented faces of the tetrahedra of T in the embedding of G. Thus, the faces on the boundary of T constitute the surface of the embedding (S) of G. Since we maintain a single connected boundary of T throughout the algorithm DT-Robust, S has a single connected surface (C4). By induction, we can prove that the boundary of T is planar triangular with all triangular faces. This type of graph can be embedded on the surface of a sphere and thus satisfies C5. The initial triangulation T_1 satisfies it trivially. Let the boundary of T_i be planar triangular with all triangular faces. In case B is empty in the algorithm DT-Robust, the boundary remains to be the same in the next stage. In the other case when B is not empty, Steps 2.1 and 2.2 remove a connected portion from the boundary which in effect creates a hole in it. The new point p_i is connected to the vertices of the hole while creating new tetrahedra. This, in effect, keeps the boundary to be planar triangular with all triangular faces and thus maintains the condition C5.

Theorem 5.4.1 The algorithm DT-Robust is type-2 robust.

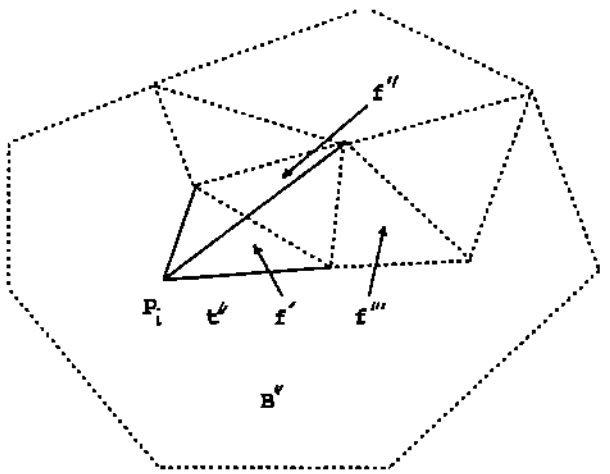


Figure 5.14 Joining p_i to the faces in B'' with proper orientations.

Proof: We prove that DT-Robust has "non-failing" and "convergence" properties. Steps 2.1 through 2.3 always produce a nonempty boundary B'' that is planar triangular without failing since every possibility is taken care of while searching for faces in B and B'. Step 2.4 can never fail since it does only symbolic computations of deleting and adding tetrahedra. Thus, the algorithm DT-Robust can never fail.

The algorithm DT-Robust produces the same output as DT-Exact under infinite precision. Under infinite precision Step 2.1 of both algorithms produces the same boundary B. Similarly, Step 2.2 of both algorithms produces the same boundary B' under infinite precision. This ensures that Step 2 of the algorithm DT-Robust becomes equivalent to that of DT-Exact under infinite precision. Thus, given the same input and infinite precision, the algorithm DT-Robust produces the same output as DT-Exact. This implies that the output produced by the algorithm DT-Robust converges to the true solution under infinite precision. By Lemma 5.4.3, 5.4.4, 5.4.5, it always produces satisfies the conditions C1 through C5, no matter what the precision is.

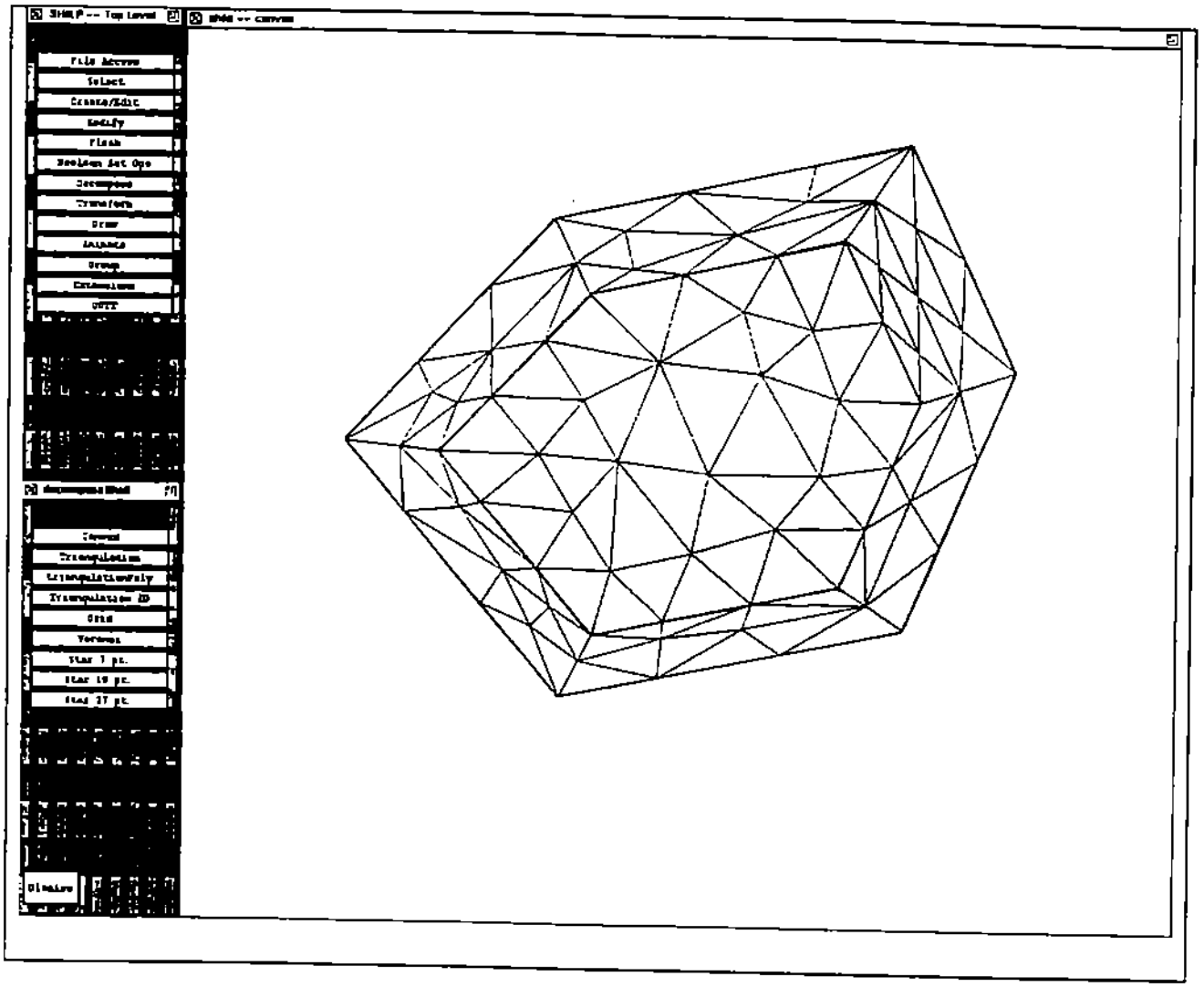


Figure 5.15 Good triangulation of a convex polyhedron

5.5 Conclusions

The good triangulation algorithm of convex polyhedra together with the convex decomposition algorithm through complete cuts gives a method for good triangulations of nonconvex polyhedra as well. However, this method has the limitation that the convex polyhedra produced by the convex decomposition algorithm may be very bad in shape. An algorithm that achieves good triangulations directly for nonconvex polyhedra is more practical.

Although in our algorithm we avoided type(i) through type(iv) tetrahedra, we could not avoid some special type of slivers, i.e., type(v) tetrahedra. Our immediate goal is to find a new method or to modify this algorithm so that we can avoid these slivers too. The difficulty with the avoidance of these slivers comes from the fact that an upper bound on the radius of the circumscribing sphere and a lower bound on the lengths of the edges of a tetrahedron do not prohibit it to be a type(v) tetrahedron. A lower bound on the radius of the inscribing sphere together with an upper bound on the radius of the circumscribing sphere of a tetrahedron avoids such tetrahedra.

We have devised a type-2 robust algorithm for the Delaunay triangulations in 3D. We have used thresholded computations (with threshold equal to zero) in our attempt to make it type-4 robust, though we could not prove it. Designing a provably type-4 or type-5 robust algorithm for this problem is a crucial open question. Another open question is: can this type-2 algorithm be generalized for higher dimensions? We believe that the properties C1 through C5 of topological triangulations generalize in higher dimensions and thus the type-2 robust algorithm can be generalized for higher dimensions too.

6. CONCLUSIONS AND FUTURE STUDIES

6.1 Contributions

This thesis focuses on efficient algorithms for decompositions of polyhedra and their robust implementations. Decompositions of polyhedra may have different flavors depending on the desired shape and size of simpler components. We have concentrated on two types of decompositions, namely convex decompositions and triangulations.

It is often the case that an efficient algorithm works on a restricted class of input. There are efficient algorithms for convex decompositions, triangulations and Peterson-style CSG decompositions for restricted class of polyhedra. In practice, however, polyhedra that do not belong to this restricted class are very common. Hence, there is a pressing need for devising efficient algorithms for more general class of polyhedra. In this thesis, we have presented efficient algorithms for convex decompositions, triangulations and Peterson-style CSG decompositions for more general class of polyhedra.

The convex decomposition algorithm is based on the cut and split paradigm of Chazelle [Cha80]. This simple paradigm led to efficient triangulation and CSG decomposition algorithms. With the help of a classic theorem on arrangements, we show that the cut and split method can be efficient. We believe that the combinatorial facts revealed through the analysis of the sequence of cuts and complete cuts in Sections 3.3.3, 4.3 will find their use in other related algorithms.

In some applications, it is desired that the simpler components are well shaped. Finite element simulations with the triangular elements need a triangular mesh with well shaped elements. There is no known algorithm for triangulating polyhedra with guaranteed quality. We have showed that a Delaunay triangulation based 2D algorithm can be extended in 3D to generate guaranteed quality tetrahedra for the convex

hull of a point set. This is the first algorithm for the problem of this kind in three dimensions.

Geometric algorithms, when implemented, often fail due to numerical errors and degenerate cases. One goal of this thesis is to devise algorithms that are implementable robustly. The definition of robustness depends on the desired output. In some applications, outputs that are "close" to the true solution are acceptable, and in others only exact solutions are acceptable. Producing an exact output even with imprecise arithmetic computations must need some assumptions on the input to buffer the information lost through erroneous computations. For the problems that ask for only combinatorial output, it is possible to produce exact output with certain minimum feature assumptions on the input. We have shown such an algorithm in Section 2.4 for polygon nesting. On the other hand, for problems that have both geometric and combinatorial parts in their solutions, it is almost impossible to produce exact outputs with inaccurate computations. In those cases, we can only expect outputs that are "close" to the true output. In three dimensions, however, it is often very hard to devise type-4 or type-5 robust algorithms. However, it may be easier to devise type-2 or type-3 robust algorithms for them. The algorithm in Section 5.4 supports this assertion. It is our hope that type-2 and type-3 robust algorithms become actually type-4 and type-5 robust with thresholded computations, though proving this fact is hard.

6.2 Future Work

This work has introduced some new ideas in designing, analyzing, and implementing algorithms in decompositions of polyhedra. However, much remains to be done. Below, we give some of the open problems in this area.

It is an open question whether we can further reduce the complexities of convex decomposition for polyhedra with holes and shells. We believe that using the concepts of constructing arrangements of planes in three dimensions, it may be possible to reduce the time complexity.

Minimum convex partition is known to be NP-hard for polyhedra with holes in their facets. It remains an open question whether minimum convex partition is still NP-hard for polyhedra without any hole in their facets.

Designing a type-4 or type-5 robust algorithm for convex decompositions is a very important open problem. To have any success in this respect, we have to understand the deep interactions between the underlying topology of polyhedra and perturbations in their features.

We have proved non trivial lower bounds of $O(p^2)$ for CNF and DNF Peterson-style formulae for polyhedra. Proving a non trivial lower bound for general Peterson-style formulae in case of polyhedra remains open. We suspect that this lower bound is also $O(p^2)$.

The good triangulation algorithm of convex polyhedra together with the convex decomposition algorithm through complete cuts gives a method for good triangulations of nonconvex polyhedra as well. However, this method has the limitation that the convex polyhedra produced by the convex decomposition algorithm may be very bad in shape. An algorithm that achieves good triangulations directly for nonconvex polyhedra is more practical.

Although in our algorithm we avoided type(i) through type(iv) tetrahedra, we could not avoid some special type of slivers, i.e., type(v) tetrahedra. The difficulty with the avoidance of these slivers comes from the fact that an upper bound on the radius of circumscribing sphere and a lower bound on the lengths of the edges of a tetrahedron do not prohibit it to be a type(v) tetrahedron. A lower bound on the radius of the inscribing sphere together with an upper bound on the radius of the circumscribing sphere of a tetrahedron avoids such tetrahedra. Generating a triangulation where all five types of bad tetrahedra are avoided remains as an open problem.

In mesh generation, it is often desired that the mesh density vary with the changes in the shape of the domain. Thus, at places where the shape changes rapidly, the mesh density should be relatively high. This type of adaptive mesh generation ensures

a balance between accuracy and efficiency. Generating an adaptive as well as good triangulation of a polyhedral domain is an important problem. We believe that the Delaunay triangulation based algorithm can be modified to generate an adaptive and good triangulation by tuning the parameter r properly in different regions.

Although a type-5 robust algorithm for 2D point set triangulations exist, there is no such algorithm in 3D. In particular, the problem of generating type-5 robust Delaunay triangulations is hard even in 2D.

It would be interesting to know how to decompose polyhedra into simpler components other than convex pieces such as star polyhedra (there is an internal point from which entire polyhedron is visible).

Decomposition of curved solids into convex pieces is another exciting problem. Not all curved surfaces are convex decomposable. So, we may seek a convex decomposition of a curved solid in terms of finite union and differences of convex components. In 2D, any polygon with algebraic curves as boundaries admits such a decomposition [BK88]. In 3D, this is possible only if the surface of the solid can be decomposed into convex, concave and planar patches. The hyperbolic surface as described in [HCV32] does not admit such decomposition. An algebraic surface of degree 2 can be decomposed into canonical patches, elliptic (Gaussian curvature > 0), hyperbolic (Gaussian curvature < 0), and parabolic (Gaussian curvature $= 0$). The problem of computing a decomposition of a curved solid with algebraic surfaces of arbitrary degree in terms of finite union and differences of components having only canonical surfaces remains open.

BIBLIOGRAPHY

BIBLIOGRAPHY

[AE86] D. Avis and H. Elgindy. Triangulating simplicial point sets in space. In *Second Ann. Symposium on Computational Geometry*, pages 133–141, 1986.

[Arm79] M. A. Armstrong. *Basic Topology.* McGraw-Hill, London, 1979.

[Arn62] B. H. Arnold. *Intutive Concepts in Elementary Topology.* Prentice-Hall, N.J., 1962.

[BA76] Babuska and A. K. Aziz. On the angle condition in the finite element method. *SIAM J. on Numerical Analysis*, 13:214–226, 1976.

[Bak89] T. J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation. *Engineering with Computers*, 5:161–175, 1989.

[BD90] C. Bajaj and T. Dey. Polygon nesting and robustness. *Information Processing Letters*, 35:23–32, 1990.

[BD91] C. Bajaj and T. Dey. Convex decompositions of polyhedra and robustness. *SIAM J. on Computing, to appear*, 1991.

[BE91] M. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. In *Seventh Ann. Symposium on Computational Geometry (ACM)*, pages 342–350, 1991.

[BEG90] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. In *31st Annual IEEE Symposium on Foundations of Computer Science*, pages 231–241, 1990.

[BGR88] B. S. Baker, E. Grosse, and C. S. Rafferty. Nonobtuse triangulation of polygons. *Discrete and Computational Geometry*, 3:147–168, 1988.

[BK88] C. Bajaj and M. Kim. Algorithms for planar geometric models. In *Proceedings of 15 th. Intl. Coloq. on Automata, Languages, and Programming, Lecture Notes in Computer Science, Springer Verlag 317*, pages 67–81, 1988.

[Cha80] B. Chazelle. *Computational Geometry and Convexity.* PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, 1980.

[Cha84] B. Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM J. on Computing*, 13:488–507, 1984.

[Che89] L. P. Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Cornell University, 1989.

[CP90] B. Chazelle and L. Palios. Triangulating a non-convex polytope. *Discrete and Computational Geometry*, 5:505–526, 1990.

[DBS91] T. K. Dey, C. Bajaj, and K. Sugihara. On good triangulations in three dimensions. In *Symposium on Solid Modeling Foundations and CAD/CAM Apllications (ACM/SIGGRAPH)*, pages 431–441, Austin, Texas, 1991.

[Dey90] T. K. Dey. Good triangulations in plane. In *Proc Second Canadian Conference in Computational Geometry*, pages 102–106, 1990.

[Dey91] T. K. Dey. Triangulation and CSG representation of polyhedra with arbitrary genus. In *Seventh Annual Symposium on Computational Geometry (ACM)*, pages 364–372, 1991.

[DGHS88] D. Dobkin, L. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Computer Graphics*, 22:31–40, 1988.

[DK91] V. J. Dielissen and A. Kaldewaij. Rectangular partition is polynomial in two dimensions but NP-Complete in three. *Information Processing Letters*, 38:1–6, 1991.

[DS88] D. Dobkin and D. Silver. Recipes for geometry and numerical analysis. In *Fourth Annual Symposium on Computational Geometry (ACM)*, pages 93–105, Urbana, Illinois, 1988.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, Berlin Heidelberg, 1987.

[Ede89] H. Edelsbrunner. Spatial triangulations with dihedral angle conditions. In *Proc Intl. Workshop on Discrete Algorithms and Complexity*, pages 83–89, Fukuoka, Japan, 1989.

[EG89] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. *J. of Computer System Science*, 38:165–194, 1989.

[EGP+88] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of arcs in the plane: Topology, combinatorics, and algorithms. In *15th Int. Colloq. on Automata, Languages and Programming (EATCS)*, pages 214–229, 1988.

[EM88] H. Edelsbrunner and P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. In *Fourth ACM Symposium on Computational Geometry*, pages 118–133, Urbana, Illinois, 1988.

[EPW86] H. Edelsbrunner, F. P. Preparata, and D. B. West. Tetrahedrizing point sets in three dimensions. Technical Report UIUCDCS-R-86-1310, University of Illinois at Urbana-Champaign, 1986.

[ETW90] H. Edelsbrunner, T. S. Tan, and R. Waupotitsch. An o(nlogn) time algorithm for the minmax angle triangulation. In *Sixth Annual Symposium on Computational Geometry (ACM)*, pages 44–52, Berkeley, California, 1990.

[FM91] S. Fortune and V. Milenkovic. Numerical stability of algorithms for line arrangements. In *Seventh Annual Symposium on Computational Geometry (ACM)*, pages 93–105, North Conway, New Hampshire, 1991.

[For89] S. Fortune. Stable maintenance of point-set triangulations in two dimensions. In *30th. IEEE Symposium on the Foundations of Computer Science*, pages 494–499, 1989.

[Fri72] I. Fried. Condition of finite element matrices generated from nonuniform meshes. *AIAA Journal*, 10:219–221, 1972.

[GSS89] L. Guibas, D. Salesin, and J. Stolfi. Building robust algorithms from imprecise computations. In *Fifth Annual Symposium on Computational Geometry (ACM)*, pages 208–217, Saarbruchen, West Germany, 1989.

[GT87] J.L. Gross and T.W. Tucker. *Topological Graph Theory.* John Wiley and Sons, 1987.

[HCV32] D. Hilbert and S. Cohn-Vossen. *Geometry and Imagination.* Chelsea, New York, 1932.

[HHK87] C. Hoffmann, J. Hopcroft, and M. Karasick. Robust set operations on polyhedral solids. Technical Report TR-87-875, Dept. of Computer Science, Cornell University, 1987.

[HK89] J. Hopcroft and P. Kahn. A paradigm for robust geometric algorithms. Technical Report TR 89-1044, Computer Science, Cornell University, Ithaca, NY, 1989.

[Joe89] B. Joe. Three-dimensional triangulations from local transformations. *SIAM J. on Sci. Stat. Comput.*, 10:718–741, 1989.

[Kar88] M. Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, Dept. of Computer Science, Mcgill University, 1988.

[Kei85] J.M. Keil. Decomposing a polygon into simpler components. *SIAM J. on Computing*, 14:799–817, 1985.

[Lin82] A. Lingas. The power of non-rectilinear holes. In *9th. Intl. Colloq. on Automata, Languages and Programming (EATCS)*, pages 369–383, 1982.

[LL86] D. T. Lee and A. K. Lin. Generalized delaunay triangulation for planar graphs. *Discrete and Computational Geometry*, 1:201–207, 1986.

[LM90] Z. Li and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *Sixth Annual Symposium on Computational Geometry (ACM)*, pages 235–242, Berkeley, California, 1990.

[Mil88] V. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, 1988.

[OR87] J. O' Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[ORS83] J. O' Rourke and K. Supowit. Some NP-hard polygon decomposition problems. *IEEE Transaction on Information Theory*, 29:181–190, 1983.

[Pet84] D. Peterson. Halfspaces representation of extrusions, solids of revolutions, and pyramids. SANDIA Report SAND84-0572, Sandia National Laboratories, 1984.

[PS86] F. P. Preparata and M. I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1986.

[PY90] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden surface removal and solid modeling. *Discrete and Computational Geometry*, 5:485–503, 1990.

[RS89] J. Rupert and R. Seidel. On the difficulty of tetrahedralizing three dimensional nonconvex polyhedra. In *5th Annual Symposium on Computational Geometry (ACM)*, pages 380–392, 1989.

[SI89a] K. Sugihara and M. Iri. Construction of the voronoi diagram for one million generators in single precision arithmetic. In *First Canadian Conference on Computational Geometry*, Montreal, Canada, 1989.

[SI89b] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. Research Memorandum RMI 89-3, Dept. of Mathematical Engineering and Instrumentation Physics, Tokyo University, 1989.

[Sib78] R. Sibson. Locally equiangular triangulations. *Computer Journal*, 21:243–245, 1978.

[SNT90] V. Srinivasan. L. R. Nackman, and J. M. Tang. Automatic mesh generation using the symmetric axis transformation of polygonal domains. Research Report, RC 16132, IBM, 1990.

[SS85] M. Segal and C. Sequin. Consistent calculations for solid modeling. In *1st Annual Symposium on Computational Geometry (ACM)*, pages 29–38, 1985.

[TWM85] J.F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation.* Elsevier Science Publishers, 1985.

[Wat81] D. F. Watson. Computing the n-dimensional tesselation with applications voronoi polytopes. *The Computer Journal.* 24:167–172, 1981.

[Yap88] C. Yap. A geometric consistency theorem for a symbolic perturbation theorem. In *Fourth Annual Symposium on Computational Geometry (ACM)*, pages 134–142, Urbana, Illinois, 1988.

VITA

Tamal Krishna Dey was born on July 26, 1963 in West Bengal, India. He received his Bachelor of Engineering degree in July, 1985 from Jadavpur University, Calcutta, India. He received his Master of Engineering degree from Indian Institute of Science, Bangalore in computer science in December, 1987. He worked in the Research and Development center of CMC Ltd., India from February, 1987 to May, 1988. He joined the computer science department of Purdue University in August, 1988 and completed his PhD. in computer science in August, 1991. He was also awarded a David Ross Fellowship from Purdue.