

**CONVEX DECOMPOSITION OF  
POLYHEDRA AND ROBUSTNESS\***

Chanderjit Bajaj  
and  
Tamal K. Dey

Computer Sciences Department  
Purdue University  
Technical Report CSD-TR-990  
CAPO Report CER-90-27  
June, 1990

---

\* Supported in part by ARO contract DAAG29-85-C0018 under Cornell MSI, NSF grant DMS 88-16286 and ONR contract N00014-88-K-0402.

# Convex Decomposition of Polyhedra and Robustness\*

Chanderjit L. Bajaj    Tamal K. Dey

Department of Computer Science  
Purdue University  
West Lafayette, IN 47907

## Abstract

We present a simple algorithm to compute a convex decomposition of a non-convex, non-manifold polyhedron of arbitrary genus (handles). The algorithm takes a non-convex polyhedron with  $n$  edges and  $r$  *notches* (features causing non-convexity in the polyhedra) and produces a worst-case optimal  $O(r^2)$  number of convex polyhedra  $S_i$ , with  $\bigcup_i S_i = S$ , in  $O(nr^2)$  time and  $O(nr)$  space. Recently, Chazelle and Palios have given a fast  $O(nr + r^2 \log r)$  time algorithm to tetrahedralize a non-convex simple polyhedron. Their algorithm, however, works for a simple polyhedron of genus 0 and with no shells (inner boundaries). The input polyhedron of our algorithm may have arbitrary genus and inner boundaries and may be a non-manifold. We also present an algorithm for the same problem while doing only finite precision arithmetic computations.

## 1 Introduction

The main purpose behind decomposition operations is to simplify a problem for complex objects into a number of subproblems dealing with simple objects. In most cases a decomposition, in terms of a finite union of disjoint convex pieces is useful and this is always possible for polyhedral models [4, 8]. Convex decompositions lead to efficient algorithms, for example, in geometric point location and intersection detection, see [8]. Our motivation stems from the use of geometric models in SHILP, a solid model creation, editing and display system being developed at Purdue [2]. Specifically, a disjoint convex decomposition of simple polyhedra allows for more efficient algorithms in motion planning, in the computation of volumetric properties, and in the finite element solution of partial differential equations. In what follows, we use the following definitions. The surface of a polyhedron  $S$  is called a 2-manifold if for each point on the surface of  $S$ , there exists an  $\epsilon$ -neighborhood which is homeomorphic to a 1-sphere or a circle [19]. Polyhedra, which have 2-manifold surface are called manifold polyhedra. Polyhedra which are not manifold are called non-manifold polyhedra. Non-manifold polyhedra may have incidences as illustrated in the Figure 1. Manifold polyhedra with holes are homeomorphic to toruses with one or more handles. Manifold polyhedra with inner boundaries are homeomorphic to 3-dimensional annuli i.e., spheres with bubbles inside them. A *reflex edge* of a polyhedron is the one where the inner dihedral angle subtended by two incident facets is greater than  $180^\circ$ .

**Related Work:** The problem of partitioning a non-convex polyhedron  $S$  into a *minimum* number of convex parts is known to be NP-hard [16, 18]. Rupert and Seidel [20] also show that the problem of determining whether a non-convex polyhedron can be partitioned into tetrahedra, without introducing Steiner

---

\*Supported in part by ARO Contract DAAG29-85-C0018 under Cornell MSI, NSF grant DMS 88-16286 and ONR contract N00014-86-K-0402.

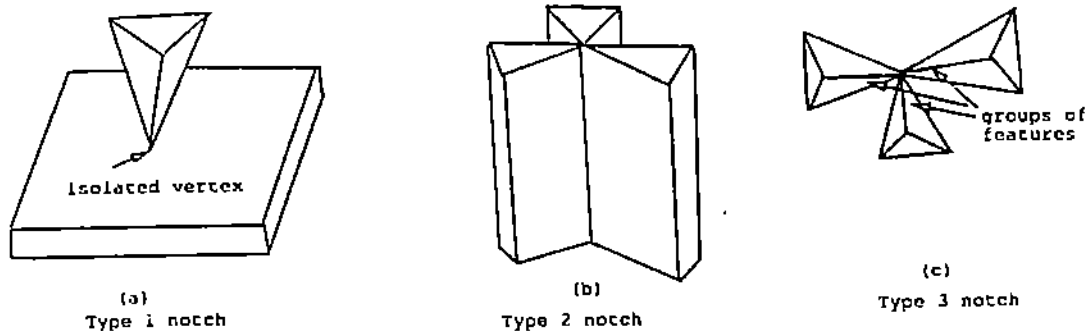


Figure 1: Non-manifold incidences or *special notches*.

points, is NP-hard. For a given polyhedron  $S$  with  $n$  edges of which  $r$  edges are *reflex*, Chazelle [4, 5] established a worst-case,  $O(r^2)$  time lower bound on the complexity of the decomposition problem, allowing Steiner points, and 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 [6], also gave an  $O(nr + r^2 \log r)$  time algorithm to tetrahedralize a subclass of non-convex polyhedra. The allowed polyhedra are all homeomorphic to a 2-sphere, i.e., have no holes (genus 0) and shells (inner boundaries) and are manifold.

**Results:** In section 3, we first 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  $\cup_i S_i = S$  in  $O(nr^2)$  time and  $O(nr)$  space. We extend this algorithm to non-manifold polyhedra which may not have abutting edges or facets but may have incidences as illustrated in Figure 1. In section 4, we give an algorithm for the same problem, which uses sophisticated heuristics based on geometric reasoning to overcome the inaccuracies involved with finite precision arithmetic computations. This algorithm runs in  $O(nr^2 + nr \log n + r^2 \log n + r^4)$  time and in  $O(nr)$  space.

## 2 Preliminaries

### 2.1 Data Structure and Definitions

Let  $S$  be a polyhedron, possibly with holes and shells, and having  $s$  vertices :  $\{v_1, v_2, \dots, v_s\}$ ,  $n$  edges :  $\{e_1, e_2, \dots, e_n\}$  and  $q$  facets :  $\{f_1, f_2, \dots, f_q\}$ .

**Polyhedron Data Structure:** The polyhedron  $S$  with arbitrary number of holes and shells, is represented by a collection of vertices, edges, and facets, each of which is maintained as structures similar to the representations of [15].

**Vertices:** Each vertex is represented 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 represented with two fields.

1. *edge.vertices*: contains pointers to the incident vertices.
2. *edge.orientededges*: contains pointers to the structures called *orientededges* which represent different orientations of an edge on each face incident on it. The orientation of an edge on a facet  $f$  is such that a traversal of the oriented edge has facet  $f$  to its right.

*Orientededges*: Each *Orientededge* is represented with three fields.

1. *orientededge.edge*: Contains pointer to the corresponding 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.

*Facets*: Each facet is represented 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. The traversal of each oriented edge cycle always has the facet to the right. Each edge cycle is represented as a linked list of structures representing the *orientededges* on the cycle. If there is a vertex touching the face, (Figure 1(a)) called an *isolated vertex*, a pointer to the vertex is included in *face.cycles* as a degenerate edge cycle.

The intersection of  $S$  with a plane  $P$  is, in general, a set of simple polygons, possibly with holes. If  $G$  is a simple polygon with vertices  $v_1, v_2, \dots, v_k$  in clockwise order, a vertex  $v_i$  is a *reflex vertex* of  $G$  if the inner angle between the edge  $(v_{i-1}, v_i)$  and  $(v_i, v_{i+1})$  is  $> 180^\circ$ . The vertices which are not *reflex vertices* are called *normal vertices* of  $G$ . The boundary of a polygon  $G$  can be partitioned into  $x$ -monotone maximal pieces called *monotone chains*, i.e., vertices of a *monotone chain* have  $x$ -coordinates in either strictly increasing or decreasing order. See Figure 2.

In general, non-manifold polyhedra have nonconvexity due to the following four types of features called *notches*.

1. *Type 1 notches*: These *notches* are caused by vertices which touch a face as illustrated in the Figure 1(a). The vertex on the face is called an *isolated vertex*.
2. *Type 2 notches*: More than two facets may be incident on an edge  $e_i$  as illustrated in the Figure 1(b). Two adjacent facets around the edge  $e_i$  which do not enclose any volume of  $S$  causes the nonconvexity or a *notch*. If there are  $2k$  ( $k > 1$ ) facets incident on  $e_i$ , 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 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 the class of *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*: An edge  $g$  of polyhedron  $S$  is a *type 4 notch* if the inner dihedral angle  $\gamma$  between two incident facets of  $g$ , is greater than  $180^\circ$ . Nonconvexity in a manifold polyhedron  $S$ , is a result of the presence of these *notches* which are also called *reflex edges*.

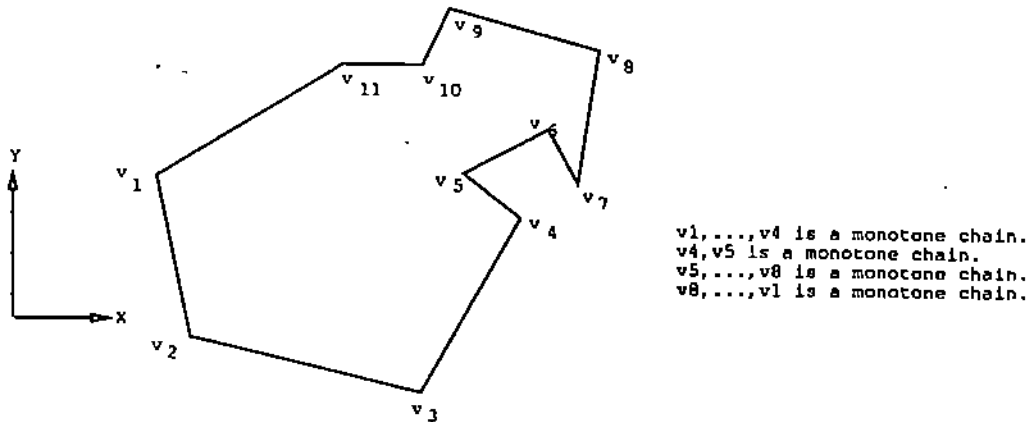


Figure 2: *Monotone chains in a polygon.*

The *notches* of type 1, type 2, type 3 are called *special notches* which are present only in non-manifold polyhedra. Our algorithm, first, removes all *special notches* from  $S$  creating manifold polyhedra and then proceeds in removing all *notches* of type 4 of the manifold polyhedra, by repeatedly cutting and splitting them with planes containing the *notches*. If an edge  $g$  is a *notch* in a manifold polyhedron, with  $f_g^-, f_g^+$  as its incident facets, a plane  $P_g$  which contains the *notch*  $g$  and subtends an inner-angle greater than  $\gamma - 180^\circ$  with both  $f_g^-$  and  $f_g^+$ , is a valid plane which resolves the *notch*  $g$ . The chosen plane  $P_g$  is also called the *notch plane* of  $g$ . 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.

## 2.2 Useful Lemmas

In the next sections we use the following Lemmas.

As discussed in [5], one can always produce a worst case optimal number ( $O(r^2)$ ) convex polyhedra by carefully choosing the *notch planes*.

**Lemma 2.1:** 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 which cannot be decomposed into fewer than  $O(r^2)$  convex pieces.

**Proof:** See [5].

**Lemma 2.2:** Let  $G$  be a simple polygon with  $r$  *reflex vertices*, then the number of *monotone chains*  $C_s$  in  $G$  is bounded as  $C_s \leq 6(1 + r)$ .

**Proof:** Follows from Theorem 3, page 22 of [4]. ♣

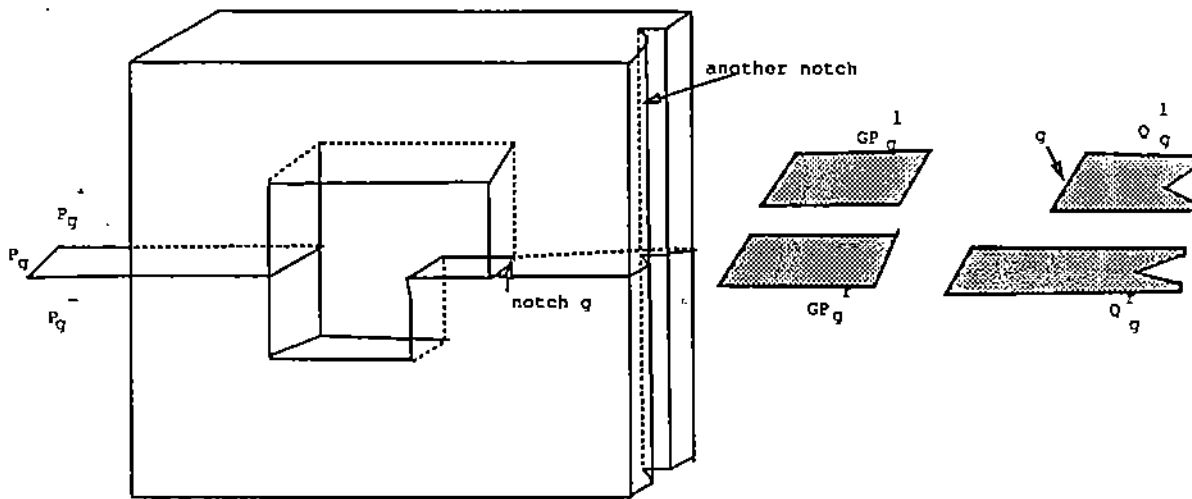


Figure 3: A notch and its notch plane, cross sectional map, cut.

Below, we use the definitions from section 2.1 of *reflex*, *normal vertices* and *monotone chains* of a polygon.

**Lemma 2.3:** Let  $G$  be a simple polygon with  $s$  *normal vertices*. There are at most  $O(s)$  *monotone chains* in  $G$ .

**Proof:** Let  $B$  be the boundary obtained by removing a vertex  $v$  and an  $\epsilon$ -ball around  $v$  from the boundary of  $G$ . Add 6 more edges to  $B$  as shown in the Figure 4 to construct a new polygon  $G'$ . The polygon  $G'$  is of opposite orientation to  $G$ . Note that the vertex  $v$  always exists such that the construction of  $G'$  is possible. In fact, any vertex which is on the convex hull of the vertices of  $G$  can be taken as  $v$ . The *normal vertices* of  $G$  are the *reflex vertices* of  $G'$  except  $v$ . Moreover, constant number of edges are added to construct  $G'$  from  $G$ . Thus,  $G'$  has  $O(s)$  *reflex vertices*. According to Lemma 2.2,  $G'$  has  $O(s)$  *monotone chains*. The polygon  $G$  cannot have more *monotone chains* than  $G'$ , which implies that  $G$  has  $O(s)$  *monotone chains*. ♣

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

**Lemma 2.4:** Let  $G$  be a simple polygon (possibly with holes) with  $r$  *reflex vertices*. No line can intersect  $G$  in more than  $r + 1$  *chords* and  $2r + 2$  points.

**Proof:** The proof proceeds inductively. The case for  $r = 0$  is trivial. In the general step, consider a polygon  $G$  with  $r = k \geq 1$  *reflex vertices*. Take an arbitrary *reflex vertex*, and resolve it by a cut through it. The cut may separate  $G$  into two polygons  $G_1$  and  $G_2$ , of  $r_1$  and  $r_2$  *reflex vertices* respectively, such that

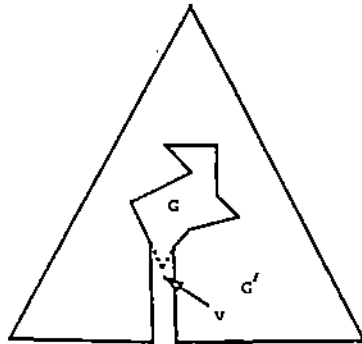


Figure 4: Constructing a polygon of opposite orientation.

$r_1 + r_2 \leq k - 1$ . Furthermore, the number of *chords* in  $G$  cannot exceed the sum of the number of *chords* in  $G_1$  and  $G_2$ . Therefore, using the induction hypothesis, one can conclude that  $L$  intersects  $G$  in no more than  $r_1 + 1 + r_2 + 1 \leq k + 1$  *chords*. If, however, the cut does not split  $G$ , one ends up with a polygon  $G'$  of at most  $k - 1$  *reflex vertices*. Since the line  $L$  may intersect the cut, just performed, the number of *chords* in  $G$  is less than or equal to that in  $G'$ , which again implies that the former is less than or equal to  $k - 1 + 1 \leq k + 1$ . ♣

### 2.3 Nesting of Polygons

The following polygon nesting problem arises as a subproblem in our polyhedral decomposition.

**Problem:** Let  $\wp$  be a set of  $k$  simple polygons  $G_i, i = 1 \dots, k$  which do not intersect along their boundaries. Corresponding to each polygon  $G_i$  we define  $ancestor(G_i)$  as the set of polygons containing  $G_i$ . The polygon  $G_k$  in  $ancestor(G_i)$  is called the *parent* of  $G_i$  if  $ancestor(G_k) = ancestor(G_i) - G_k$ . Notice that there may not exist any such  $G_k$  since  $ancestor(G_i)$  may be empty. In that case, we say that the *parent* of  $G_i$  is *null*. Any polygon with *parent*  $G_k$  is called the *child* of  $G_k$ . See Figure 5. The *nesting structure* of  $\wp$  is an acyclic directed graph (a forest of trees) in which there is a node  $n_i$ , corresponding to each polygon  $G_i$  in  $\wp$ , and a directed edge from a node  $n_i$  to  $n_j$  if and only if  $G_j$  is the parent of  $G_i$ . The polygon nesting problem is to compute the *nesting structure* of a set  $\wp$  of simple nonintersecting polygons.

**Lemma 2.5:** The problem of polygon nesting for  $k$  simple, nonintersecting polygons can be solved in  $O(s + t \log t)$  time assuming exact numerical calculations, where  $s$  is the total number of vertices and  $t$  is the total number of *monotone chains* of all input polygons.

**Proof:** See [3]. Though, the algorithm, given in [3], uses a slightly different type of *monotone chains*, called *subchains*, it also works for the *monotone chains* as defined in this paper. With this slight modification, Theorem 2.1 of [3] can be restated as Lemma 2.5 given above. ♣

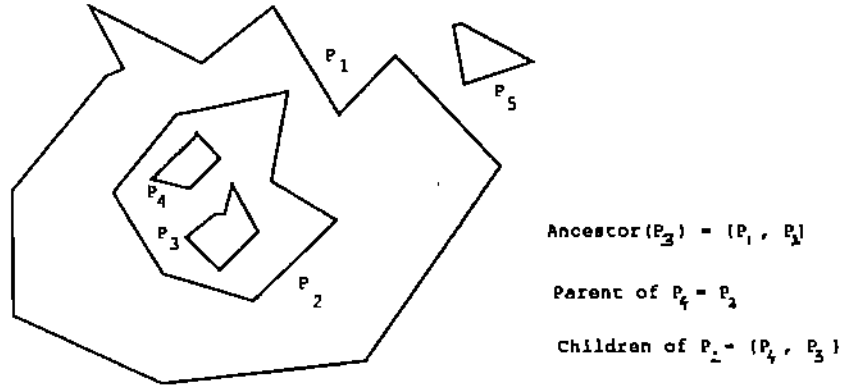


Figure 5: Polygon nesting.

### 3 Convex Decomposition

We assume the input polyhedron  $S$  to be a manifold while describing the algorithm and extend it to handle non-manifold polyhedra later. By this assumption, *notches* in  $S$  are only *reflex edges*. The algorithm for decomposing a polyhedron  $S$  with  $r$  notches consists of a sequence of intersections of polyhedra with *notch planes*. Hence, we first describe the method of cutting a polyhedron  $S$  by a *notch plane*  $P_g$  of a *notch*  $g$ .

#### 3.1 Cross Sectional Map

The *notch plane*  $P_g: ax + by + cz + d = 0$  defines two open half spaces  $P_g^+ : ax + by + cz + d > 0$  and  $P_g^- : ax + by + cz + d < 0$ . The closure of  $P_g^+$  is  $P_g^u = P_g^+ \cup P_g^l$ , where  $P_g^l : ax + by + cz + d = 0$  is the oriented plane  $P_g$  with normal  $(a, b, c)$  pointing into the exterior of  $P_g^+$ . Similarly, the closure of  $P_g^-$  is  $P_g^b = P_g^- \cup P_g^r$  where  $P_g^r : -ax - by - cz - d = 0$  is the oriented plane  $P_g$ , with normal  $(-a, -b, -c)$  pointing into the exterior of  $P_g^-$ .

Cutting a polyhedron  $S$  with the plane  $P_g$  is equivalent to computing

$$\begin{aligned}
 S \cap P_g^u &= (S \cap P_g^+) \cup GP_g^l \\
 S \cap P_g^b &= (S \cap P_g^-) \cup GP_g^r
 \end{aligned}$$

where

$$\begin{aligned}
 GP_g^l &= (\text{closure}(S \cap P_g^+)) - (S \cap P_g^+) \\
 GP_g^r &= (\text{closure}(S \cap P_g^-)) - (S \cap P_g^-).
 \end{aligned}$$

We also frequently refer to  $GP_g^l$  and  $GP_g^r$  as *cross sectional maps*. 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. However, one can observe that  $GP_g^l$  and  $GP_g^r$  are same if there is no facet of  $S$  lying on the *notch plane*. For simplicity, we assume  $GP_g^l$  and  $GP_g^r$  to be congruent and refer to it as  $GP_g$  in describing the algorithm. With minor modifications of the algorithm one may remove this restriction.



The construction of  $GP_g$  corresponding to the *notch plane*  $P_g$  is the crucial part in splitting a polyhedron  $S$  to remove  $g$ . The unique polygon  $Q_g$  (possibly with holes) in  $GP_g$ , called the *cut*, supporting the *notch*  $g$  is determined and  $S$  is split along this cut. Actually, splitting  $S$  along the cut instead of the *cross sectional map*, is sufficient to remove the *notch*  $g$  of  $S$ . Note that because of this,  $S$  may not get separated into two different pieces after the split. In Figure 3, the removal of the *notch*  $g$  through the cut  $Q_g$  does not separate  $S$ . The *notch*  $g$  may lie on the inner or the outer boundary of  $Q_g$ . We denote the boundary containing  $g$  as  $B_g$ .

- **Step I:** Determine  $Q_g$ . This calls for computing inner and outer boundaries of  $Q_g$ .
- **Step II:** Split  $S$ . While describing the algorithm we assume  $S$  is separated into two pieces by the cut  $Q_g$ . The case where  $S$  is merely spliced by  $Q_g$  instead of getting separated into two pieces does not incur any extra overhead to our algorithm.

In what follows, we use (lower case) letters,  $s, u$ , for counting vertices,  $m, n, p, t$  for counting edges, and  $q$  for counting facets. Let  $S$  have  $p$  edges of which  $r$  are *reflex*.

### 3.2 Description of the Algorithm

**Step I :** First compute all boundaries  $B$  present in the *cross sectional map*  $GP_g$ . Visit all the facets of  $S$  in turn. If a facet  $f_i$  intersect the *notch plane*  $P_g$ , all intersection points are computed. Let  $a_1^i, a_2^i, \dots, a_k^i$  be the intersection points on the edges  $e_1^i, e_2^i, \dots, e_k^i$  respectively of  $f_i$ . These intersection points can be sorted along the line of intersection  $P_g \cap f_i$  at a cost, linear in number of edges present in the facet  $f_i$  using the algorithm of [13]. Associate this sorted sequence of intersection points with  $f_i$ . Further, with each intersection point  $a_j^i$ , keep the information of the edge  $e_j^i$ . Pick an intersection point  $a_j^i$ . Continue to

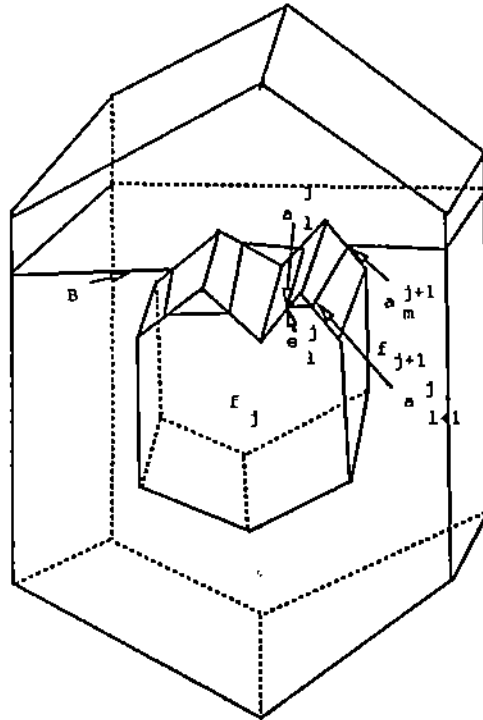


Figure 6: Computation of a boundary in the *cross sectional map*.

construct the boundary  $B$  containing  $a_j^i$  as follows. See Figure 6. One of the segments  $a_j^i a_{j+1}^i$  and  $a_j^i a_{j-1}^i$  lies inside  $f_j$ . Without loss of generality, assume  $a_j^i a_{j+1}^i$  lies inside  $f_j$ . Join  $a_j^i$  and  $a_{j+1}^i$  and continue from  $a_{j+1}^i$

to determine other edges of the boundary  $B$ . Consider the other facet  $f_{j+1}$  adjacent to the edge  $e_{i+1}^j$ . The facet  $f_{j+1}$  can be retrieved in constant time in our data structure. Determine the intersection point  $a_m^{j+1}$  adjacent to  $a_{i+1}^j$  on  $f_{j+1}$  such that the segment  $a_{i+1}^j a_m^{j+1}$  lies inside  $f_{j+1}$ . Without loss of generality, assume  $a_m^{j+1}$  is ordered after  $a_{i+1}^j$  in the sorted sequence of intersection points associated with  $f_{j+1}$ . Note that the points  $a_{m-1}^{j+1}$  and  $a_{i+1}^j$  are same. Proceed from  $a_m^{j+1}$  and continue the above procedure until the initial point  $a_i^j$  is reached. This completes the computation of  $B$ . Once a boundary computation is completed, pick up another intersection point which has not been visited yet and construct the corresponding boundary by the above method. Continuing this procedure until all intersection points are visited gives all the boundaries present in the *cross sectional map*. The adjacent points of an intersection point on a facet can be retrieved in constant time if the sorted sequence of intersection points is maintained as a doubly linked list.

Next, determine the inner and outer boundaries of  $Q_g$ . 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 orientation of the edges on the boundary. Orientation of each such edge is determined in constant time since the orientations of the *notch plane* and the facets intersecting the *notch plane* are known.

*Case(i):*  $B_g$  is an outer boundary of  $Q_g$ . Let  $I_i$  be any inner boundary of  $Q_g$ . The boundary  $I_i$  itself constitutes a simple polygon. Polygon  $I_i$  will have at least one (actually at least three) vertex, which is a *normal vertex*. Since  $I_i$  is the inner boundary of  $Q_g$ , the vertices which are *normal vertices* of polygon  $I_i$  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 point which is the intersection point of  $P_g$  with a *notch* of  $S$ . Determine the set  $W$  of boundaries having at least one point where a *notch* of  $S$  and  $P_g$  intersect. This takes  $O(u')$  time, where  $u'$  is the number of vertices present on the cross sectional map. Call the boundaries in the set  $W \cup B_g$  as *interesting boundaries*. 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* which are outer boundaries of some polygon in the *cross sectional map*, have  $O(t)$  *reflex vertices*. On the other hand, the *interesting boundaries* which are inner boundaries of some polygon in the cross sectional map have  $O(t)$  *normal vertices*. Thus, according to the Lemma 2.2 and 2.3, there are at most  $O(t)$  *monotone chains* in the *interesting boundaries*. If there are  $u$  vertices on the *interesting boundaries*, the inner boundaries of  $Q_g$  can be determined in  $O(u + t \log t)$  time using Lemma 2.5. The computation of the boundaries in the *cross sectional map* takes at most  $O(p)$  time. This is due to the fact that the sorted sequence of intersection points on each facet can be computed at a cost linear in number of edges of the facet. Thus, in this case, the inner and outer boundaries of  $Q_g$  can be determined in

$$O(p + u + u' + t \log t) = O(p + t \log t)$$

time, since  $u = O(u') = O(p)$ .

*Case(ii):*  $B_g$  is an inner boundary of  $Q_g$ . Visit all edges of the polyhedron  $S$ , being split to compute the sorted sequence of intersection points on each facet and compute all the boundaries present in the *cross sectional map*. Determine the boundaries which contain the boundary  $B_g$  inside them. Call these boundaries, together with  $B_g$ , as *interesting boundaries*. This takes  $O(p + u') = O(p)$  time. Apply the polygon nesting algorithm of [3] on these *interesting boundaries* to detect the parent polygon of  $B_g$  which is the outer boundary of  $Q_g$ . The *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 must be as many inner boundaries as outer boundaries. Hence, the number of *interesting boundaries* is bounded above by twice the number of inner boundaries present in the *cross sectional map*. As discussed in the previous case, this number must be bounded above by the number of *notches* intersected by the *notch plane*. Thus, there are  $O(t)$  *interesting boundaries*. Further, the number of *monotone chains* present in these *interesting boundaries* can be at most  $O(t)$ . Hence, as in the previous case, the inner and outer boundaries of  $Q_g$  can be determined in  $O(p + t \log t)$  time.

Step II: Separation of  $S$  corresponding to the cut  $Q_g$  is carried out by splitting facets which are intersected by  $Q_g$ . Suppose  $f_i$  is such a facet which is to be split at  $a_1^i, a_2^i, \dots, a_k^i$  which are on the edges  $e_1^i, e_2^i, \dots, e_k^i$ . The splitting of  $f_i$  consists of splitting the edges on which  $(a_1^i, a_2^i, \dots, a_k^i)$  lies. Visit only the intersection points corresponding to the vertices of  $Q_g$  and for each such intersection point spend constant time for setting relevant pointers to carry out the split operation. Create two oppositely oriented facets at the same geometric location corresponding to the cut  $Q_g$ . Adjust all the modified incidences properly. A depth first traversal in the modified vertex list completes the separation of  $S$  by collecting all the pertinent features of each piece. This process cannot take more than  $O(p)$  time. Combining the costs of *Step I* and *Step II* yields the following Lemma.

**Lemma 3.1.** A manifold polyhedron  $S$  of genus 0, having  $p$  edges can be partitioned with a *notch plane*  $P_g$  of a *notch*  $g$  in  $O(p + t \log t)$  time and in  $O(p)$  space where  $t$  is the number of *notches* intersected by  $P_g$ .

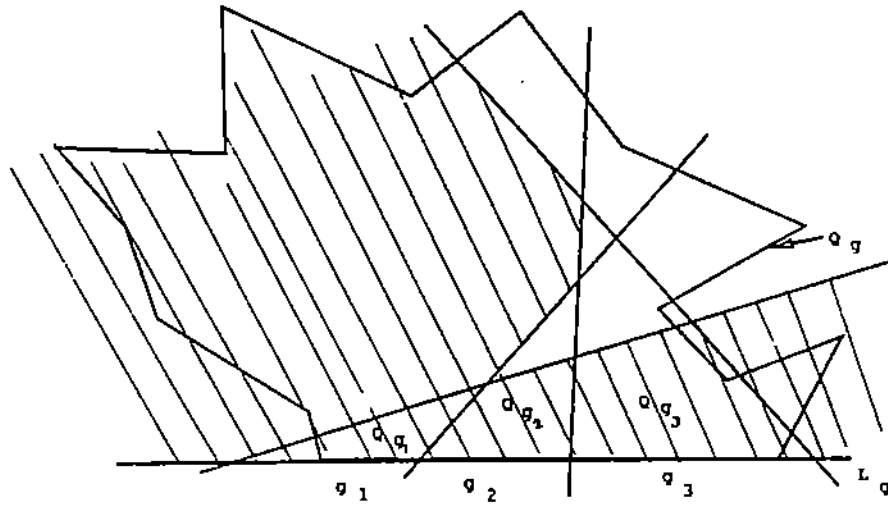
We can generalize the above result for a polyhedron of arbitrary genus. For this, as described in [5], we need to handle the situation when the cut does not separate  $S$  into two pieces, but only creates two new facets supporting the cut at the same geometric location. A depth-first search in the vertex list determines whether the cut separates  $S$  into two pieces or not.

**Lemma 3.2.** Let  $S_1, S_2, \dots, S_k$  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 on  $Q_{g_i}$ , respectively. Then  $m$  and  $u$ , the total number of edges and vertices on all the cuts supported by the *subnotches* of the *notch*  $g$  are given as  $m = \sum_{i=1}^k m_i = O(n)$  and  $u = \sum_{i=1}^k u_i = O(n)$ .

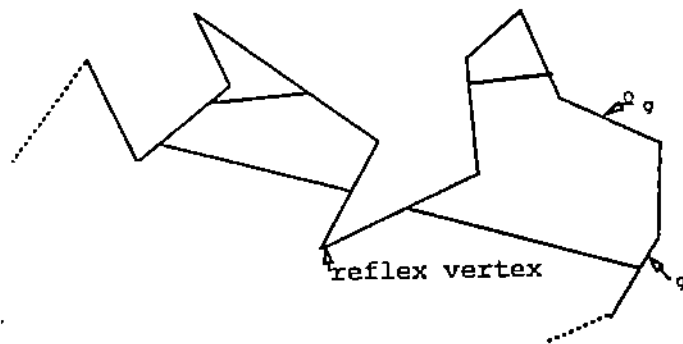
**Proof:** Consider the cut  $Q_g$  produced by the intersection of  $S$  with  $P_g$ . The region in  $Q_g$  is divided into smaller facets by *notch lines* produced by the intersection of other *notch planes* with  $P_g$ . We focus on the facets  $Q_{g_1}, Q_{g_2}, \dots, Q_{g_k}$  adjacent to the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

Consider the set of *notch lines* which divides  $Q_g$  and the line  $L_g$  corresponding to the *notch*  $g$ . They produce a line arrangement [8] on the *notch plane*  $P_g$ . Consider the facets adjacent to the line  $L_g$  in this arrangement. These are called *zones* of  $L_g$ . See Figure 7(a). Let us denote the set of these *zones* by  $Z_g$  and their vertices and edges by  $V_g$  and  $E_g$  respectively. It is proved that (Theorem 5.3, pp. 89, [8])  $|V_g| \leq 5l - 3$  and  $|E_g| \leq 5l - 1$  if there are  $l$  lines in the arrangement. Overlaying  $Q_g$  on  $Z_g$  produces  $Q_{g_1}, Q_{g_2}, \dots, Q_{g_k}$ . Let  $V'_g$  and  $E'_g$  denote the set of vertices and edges in  $Q_{g_1}, Q_{g_2}, \dots, Q_{g_k}$ . The vertices in  $V'_g$  can be partitioned into three different sets, namely,  $T_1, T_2, T_3$ . The set  $T_1$  consists of vertices formed by the intersections of *notch lines*,  $T_2$  consists of vertices formed by the intersections of edges of  $Q_g$  and  $T_3$  consists of vertices formed by the intersections of *notch lines* and edges of  $Q_g$ . Certainly,  $|T_1| \leq |V_g| = 5l - 3$  since overlaying of  $Q_g$  on  $Z_g$  cannot introduce any vertices in  $T_1$ . If  $Q_g$  has  $u'$  edges,  $|T_2| \leq u'$ .

To count the number of vertices in  $T_3$ , consider an edge  $e$  in  $E_g$  which contributes one or more segments  $e_s$  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  $e_s$ . Charge a cost of 1 to the *reflex vertex* which lies to the left (or, right) of each segment and charge a cost of 1 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. Then, that *reflex vertex* must appear between two segments of two edges in  $E_g$  as shown in Figure 7(b). As can be easily observed, all four edge segments cannot be adjacent to the regions incident on an edge  $g$  of  $Q_g$ . This contradicts our assumption that all these four 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  $r_g + 5l - 1$  where  $r_g$  is the number of *reflex vertices* present in  $Q_g$ .



(a)  
Shaded regions are zones of  $L_g$ .



(b)

Figure 7: *Zones of a line and cuts.*

This implies that as a result of intersections with  $Q_g$ , at most  $r_g + 5l - 1$  segments of edges in  $E_g$  are contributed to  $E'_g$ . Hence,  $|T_3| \leq 2(r_g + 5l - 1)$ . Putting all these together, we have

$$\begin{aligned} |V'_g| &= |T_1| + |T_2| + |T_3| \\ &\leq 5l - 3 + u' + 2r_g + 10l - 6 \\ &\leq 15l + u' + 2r_g - 9. \end{aligned}$$

Since there can be at most  $r$  notch planes,  $l \leq r$ . Certainly,  $r_g \leq r$  and  $u' \leq n$ . This gives

$$u = |V'_g| \leq 15r + n + 2r - 9 = O(n + r) = O(n).$$

Since  $Q_{g_1}, Q_{g_2}, \dots, Q_{g_k}$  form a plane graph, we have

$$m = |E'_g| = O(|V'_g|) = O(n). \clubsuit$$

**Lemma 3.3:** The total number of edges in the final decomposition of the polyhedron  $S$  with  $r$  notches and  $n$  edges is  $O(nr)$ .

**Proof:** Total number of edges in the final decomposition consists of newly generated edges by the cuts, and the edges of  $S$  which are not intersected by any notch plane. Since the total number of edges present in all the cuts corresponding to a notch is  $O(n)$ , the total number of newly generated edges by each notch plane is  $O(n)$ . Thus  $r$  notch planes generate  $O(nr)$  new edges. Hence, the total number of edges in the final decomposition is  $O(nr + n) = O(nr)$ .  $\clubsuit$

**Theorem 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  $O(nr^2)$  time and  $O(nr)$  space.

**Proof:** Decomposition of a polyhedron consists of a sequence of cuts through the notches of  $S$ . Assign a notch plane for each notch in  $S$  in  $O(r)$  preprocessing time. Remove each notch by removing all of its subnotches with the notch plane assigned to this notch. Each planar cut to remove a subnotch in a polyhedron, can be carried out by the method described above. According to the Lemma 2.1 this produces  $O(r^2)$  convex pieces at the end since all subnotches of a notch are removed by a single notch plane.

**Worst-Case Complexity:** At a generic instance of the algorithm, let  $S_1, S_2, \dots, S_k$  be the  $k$  distinct (non-convex) polyhedra in the current decomposition, where each  $S_i$  contains the subnotch  $g_i$  of a notch  $g$  which is going to be removed. Let  $S_i$  have  $p_i$  edges and  $p = \sum_{i=1}^k p_i$ . Let  $t_i$  be the number of notches intersected by  $P_{g_i}$  in  $S_i$  and  $t = \sum_{i=1}^k t_i$ .

Applying Lemma 3.1, each subnotch  $g_i$  in  $S_i$  can be removed in  $O(p_i + t_i \log t_i)$  time and in  $O(p_i)$  space. Thus, removal of a notch  $g$  can be carried out in  $O(\sum_{i=1}^k (p_i + t_i \log t_i))$  time and in  $O(\sum_{i=1}^k p_i)$  space. By Lemma 3.3,  $\sum_{i=1}^k p_i = O(nr)$ . Since a notch plane can intersect at most  $r - 1$  notches,  $t = O(r)$ . This gives,

$$\sum_{i=1}^k t_i \log t_i = O(t \log t) = O(r \log r).$$

Hence, a notch  $g$  can be removed in  $O(nr + r \log r) = O(nr)$  time. Thus, elimination of  $r$  notches takes  $O(nr^2)$  time. In Lemma 3.3, we prove that the total number of edges in the final decomposition of  $S$  is  $O(nr)$ . This implies that the space complexity of polyhedral decomposition is  $O(nr)$ .  $\clubsuit$

### 3.3 Decomposition of non-manifold polyhedra

For a non-manifold polyhedron  $S$ , nonconvexity results from *notches* of four types as discussed in section 2.1. Let  $S$  have  $n$  edges and  $r$  *notches*. Preprocess  $S$  as follows to remove the *notches* of first three types, called the *special notches*. Let this process produce a decomposition  $S_1, S_2, \dots, S_l$  where each  $S_i$  is a manifold polyhedron having notches of only the fourth type. Apply Theorem 3.1 on each of them to obtain a worst-case optimal convex decomposition.

**Removal of type 1 notches:** In this case, as can be observed from the Figure 1(a), the vertex  $v_i$  causing the nonconvexity is detached from the facet  $f_i$  on which it is incident as *isolated vertex*. Identifying these vertices and detaching them from corresponding facets take at most  $O(n)$  time.

**Removal of type 2 notches:** In this case, more than two facets are incident on an edge  $e_i$ . Let these facets be  $f_1, f_2, \dots, f_{r_i}$ . Consider a cross section  $C$  which is the intersection of the facets incident on  $e_i$  with the plane  $P$  perpendicular to the edge  $e_i$ .  $C$  consists of edges  $e_j = (f_j \cap P)$ . Sort the facets circularly around the edge  $e_i$  by the circular sort of the edges  $e_j$  which are incident on  $e_i \cap P$ . Pair the adjacent facets which enclose a volume of  $S$ . Let this pairing be  $(f_1, f_2), (f_3, f_4), \dots, (f_{r_i-1}, f_{r_i})$ . Create an edge between each pair of facets and delete the edge  $e_i$ . All these edges are at the same geometric location of  $e_i$ . Adjust all the incidences properly. Sorting of facets around the edge  $e_i$  takes  $O(r_i \log r_i)$  time. The adjustment time for all incidences in the internal representation of  $S$  cannot exceed  $O(n)$ . Thus, removal of all *type 2 notches* takes at most  $O(r \log r + nr) = O(nr)$  time.

**Removal of type 3 notches:** Let  $v_i$  be a vertex which corresponds to a *type 3 notch*. In this case, collect the features (edges, facets) incident on  $v_i$  which are reachable from one another while remaining always on the surface of  $S$  and never crossing  $v_i$ . This gives a partition of the features incident on  $v_i$  into smaller groups. For each such group, create a vertex at the same geometric location of  $v_i$  and adjust all the incidences properly. This in effect, removes the nonconvexity caused by  $v_i$ . All such vertices causing *type 3 notches* in  $S$  can be identified in  $O(n)$  time by depth first traversal in the underlying graph of  $S$ . Removal of each such notch takes at most  $O(n)$  time. Thus, all *type 3 notches* can be removed in  $O(nr)$  time.

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

**Theorem 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  $O(nr^2)$  time and  $O(nr)$  space.

**Proof:** Remove all *special notches* from  $S$  in  $O(nr)$  and  $O(n)$  space as discussed above. Let  $S_1, S_2, \dots, S_l$  be the manifold polyhedra created by this process. Let  $S_i$  have  $n_i$  edges of which  $r_i$  are reflex. Using Theorem 3.1 on each of them, we conclude that  $S$  can be decomposed into  $O(r^2)$  convex polyhedra in  $O(\sum_{i=1}^l n_i r_i^2) = O(nr^2)$  time and in  $O(\sum_{i=1}^l n_i r_i) = O(nr)$  space.

**Decomposition into Tetrahedra:** Let  $S_1, S_2, \dots, S_k$  be the convex polyhedra produced by convex decomposition of a polyhedron  $S$ . Each convex piece with  $p_i$  edges can be triangulated into  $O(p_i)$  tetrahedra in a straightforward manner (triangulate every convex facet and then tetrahedralize by choosing a point in the interior of the convex polyhedra). This takes at most  $O(p_i)$  time for each convex piece. Hence, triangulation of all pieces takes  $O(\sum_{i=1}^k p_i) = O(nr)$  time producing  $O(nr)$  tetrahedra.

## 4 Convex Decomposition under Finite Precision Arithmetic

**Motivation:** When implementing geometric operations stemming from practical applications, one cannot ignore the degenerate geometric configurations that often arise, as well as the need to make specific topological decisions based on imprecise finite precision numerical computations [12, 15, 23]. We model the inexact arithmetic computations by  $\varepsilon$ -arithmetic [10, 11] where the arithmetic operations  $+$ ,  $-$ ,  $\div$ ,  $\times$  are performed with relative error of at most  $\varepsilon$ . Under this model, the absolute error in distance computations of one polyhedral feature from another is bounded by a certain quantity  $\delta = k\varepsilon B$ , where  $B$  is the maximum value of any coordinate and  $k$  is a constant. See [17]. When making decisions about the incidence of these polyhedral features (vertices, edges, facets), on the basis of the computed distance (with sign), one can rely on the sign of the computation only if the distance is greater than  $\delta$ . On the other hand, if the computed distances are less than  $\delta$ , one also needs to consider the topological constraints of the geometric configuration to decide on a reliable choice. In particular, in regions of uncertainty i.e. within the  $\delta$ -ball, the choices are all equally likely that the computed quantity, is negative, zero or positive. Such decision points of uncertainty, 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. The algorithm which follows this paradigm would never fail, though it may not always compute a valid output. Such algorithms have been termed as parsimonious by Fortune [10].

An algorithm under  $\varepsilon$ -arithmetic, is called robust if it computes an output which is exact for some perturbed input. It is called stable if the perturbation required is small. Recently, in [10, 11, 17] authors have given robust and stable algorithms for some important problems in two dimensions. Except [14], there is no known robust algorithm for any problem in three dimensions. The difficulty arises due to the fact that the perturbations in the positions of the polyhedral features may not render a valid polyhedron embedded in  $\mathfrak{R}^3$ . In [14], Hopcroft and Kahn discuss the existence of a valid polyhedron which admits the positions of the perturbed vertices of a convex polyhedron. The case of non-convex polyhedra is perceived to be hard and requires understanding the deep interactions between topology and perturbations of polyhedral features of non-convex polyhedra.

Karasick [15] gives an algorithm for the problem of polyhedral intersection where he uses geometric reasoning to avoid conflicting decisions about polyhedral features. In this paper, we extend the results in [15] and provide an algorithm for the problem of polyhedral decomposition which also uses geometric reasoning to avoid conflicting decisions. Though, as yet we are unable to prove our algorithm to be parsimonious, we report various heuristics we have implemented in our effort to make the decomposition algorithm robust and stable. We also describe a worst case running time bound for the algorithm under the  $\varepsilon$ -arithmetic model.

**More related work:** The issue of robustness in geometric algorithms have recently taken added importance because of the increasing use of geometric manipulations in computer-aided design, and solid modeling [1]. Edelsbrunner and Mücke [9], and Yap [24], suggest using expensive symbolic perturbation techniques for handling geometric degeneracies. Sugihara and Iri [23], and Dobkin and Silver [7], describe an approach to achieve consistent computations in solid modeling, by ensuring that computations are carried out with sufficiently higher precision than 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. Segal and Sequin [21] estimate various numerical tolerances, tuned to each computation, to maintain consistency. Milenkovic [17] presents techniques for computing the arrangements of a set of lines in two dimensions robustly. He introduces the concept of *pseudo lines* which preserves some basic topological properties of lines and computes the arrangements in terms of these *pseudo lines*. Hoffmann.

Hopcroft and Karasick [12], and Karasick [15], propose using geometric reasoning and apply it to the problem of polyhedral intersections. Sugihara [22] uses geometric reasoning to avoid redundant decisions and thereby eliminate topological inconsistencies in the construction of planar Voronoi diagrams. Guibas, Salesin and Stolfi [11] propose a framework of computations called  $\epsilon$ -geometry, in which they compute an exact solution for a perturbed version of the input. So does Fortune [10] who applies it to the problem of triangulating two dimensional point sets.

#### 4.1 Intersection & Incidence Tests

In what follows, we assume the input polyhedra are manifold. Non-manifold polyhedra can be handled as discussed in the previous section. It is clear from discussion of our previous algorithm that numerical computations are needed in different types of intersections and incidence testings. We assume minimum feature criteria for polyhedra as follows. The distance between two distinct vertices or between a vertex and an edge and the dihedral angle between any two facets may not be less than a minimum value. The choice of this minimum threshold value is described in our algorithm. To decide whether an edge is intersected by a plane, one must decide the classifications 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 incident testings are carried out by 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). Edge-plane classifications are done only after vertex-plane and facet-plane classifications. In what follows, assume the equation of a plane  $P_i : a_i x + b_i y + c_i z + d_i = 0$  is normalized with  $a_i^2 + b_i^2 + c_i^2 = 1$ .

**Vertex-Plane Classification:** To classify the incidence of a vertex  $v_i = (x_i, y_i, z_i)$  w.r.t the plane  $P : ax + by + cz + d = 0$ , compute the normalized algebraic distance of  $v_i$  from  $P$  by computing  $ax_i + by_i + cz_i + d$ . 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 half space containing the plane normal  $(a, b, c)$ . Accept the sign of the computations as correct if the above distance of  $v_i$  from  $P$  is larger than  $\delta$ . Otherwise, apply geometric reasoning rules, as detailed below, to classify vertex  $v_i$  w.r.t. the plane  $P$ . In the following algorithmic version of the vertex-plane classification, the intersection between an edge  $e$  incident on  $v_i$  and the plane  $P$  is computed as follows. Let  $e$  be incident on planes  $P_1, P_2$ , where  $P_i : a_i x + b_i y + c_i z + d_i = 0$ . Compute the intersection point  $r$  of  $e$  and the plane  $P$  by computing the solution of the linear system,

$A r = d$  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.

#### Vertex-Plane-Classif( $v_i, P$ )

begin

Let  $v_i = (x_i, y_i, z_i)$  be a vertex incident on edges  $e_1 = (v_i, w_1), e_2 = (v_i, w_2), \dots, e_k = (v_i, w_k)$ .

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 distance computation\*)

if  $l > 0$  then

classify  $v_i$  as "above"

else

classify  $v_i$  as "below"



```

else
  loop
    (*Comment: If 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 future facet - plane
    classifications to decide its classification consistently.*)

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

```

**Facet-Plane Classification:** If a facet  $f_i$  is intersected by a plane  $P$  in such a way that  $f_i$  does not lie on  $P$  then the points of intersection should necessarily be (i) collinear with the line of intersection of  $f_i$  and  $P$ , and (ii) all the vertices of  $f_i$  on one side of the intersection line, should all be of the same classification w.r.t. the plane  $P$ . Vertices which have been temporarily classified as "maybeon", are classified in a consistent way, i.e., they satisfy the above two properties (i) and (ii), with perturbations of at most  $\delta$ . An algorithmic version of the facet-plane classification is given below.

**Facet-Plane-Classif ( $f_i, P$ )**

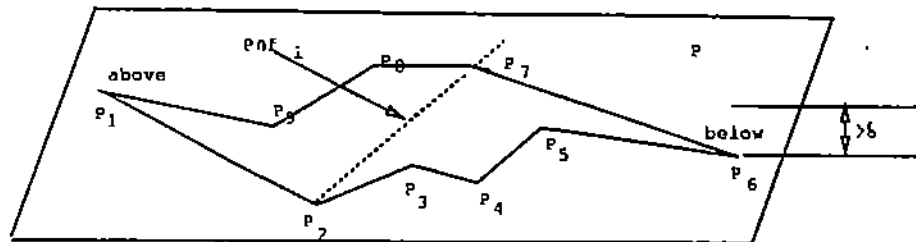
```

begin
  case
    (i) All the 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 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  $f_i \cap P$ .
endif
loop
  for each "maybeon" vertex  $v_k$  on  $f_i$  do
    if  $v_k$  is at a distance greater than  $\delta$  from  $L$  then
      if  $v_k$  and  $v_u$  lie on opposite sides of  $L$  then
        classify  $v_k$  with a classification which is opposite to that of  $v_u$ .
      else
        classify  $v_k$  with a classification which is same as that of  $v_u$ .
      endif
    endif
  endloop
  The vertices which are still not classified

```



$P_2, \dots, P_5$  and  $P_7, \dots, P_9$  are maybeon vertices.  
 $P_3, \dots, P_5$  gets the classification of  $P_6$ .  
 $P_8, \dots, P_9$  gets the classification of  $P_1$ .

Figure 8: Case(ii) of facet-plane classification.

classify them as "on" (\*Comment: These vertices are within a distance of  $\delta$  from  $L$  and hence will be collinear with  $L$  by a perturbation of at most  $\delta$ . See Figure 8.\*)

(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 on  $e$  and

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 all the points

of intersections and apply the methods of case (ii) to classify remaining "maybeon" vertices.  
*endif*  
*endcase*  
*end.*

**Edge-Plane Classification:** An edge can get any of the three classifications which are "not-intersected", "intersected", and "on". The classifications of the vertices incident on an edge are used to classify an edge  $e$ . 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  $\delta$  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.*

The following lemma related to consistent ordering of intersection points of a facet on the line of intersection is used in later sections.

**Lemma 4.1:** Let  $v$  be a vertex which is decided not to lie on the plane  $P$  and whose classification w.r.t the plane is known. Let  $e_1, e_2$  be the edges incident on  $v$  on a facet  $f$  which are classified as "intersected". Denote the intersection points of  $e_1, e_2$  with  $P$  as  $v_1$  and  $v_2$  respectively. Let  $O$  denote the ordering of  $v_1, v_2$  on the directed intersection line  $f \cap P$  which is consistent with the classification of  $v$ . If  $\frac{M}{2} \geq \frac{\delta}{\sin \alpha}$  holds,  $O$  can be determined correctly. Here  $\delta$  is the maximum absolute error in distance computations,  $\alpha$  is the angle between edges  $e_1, e_2$  on  $f$ ,  $M$  is a suitably chosen large machine representable absolute value.

**Proof:** Consider the vertex  $v$  with incident edges  $e_1, e_2$  on facet  $f$ . Let  $L = f \cap P$  be directed as shown in Fig. 4.2 and let the actual distance of  $v$  from  $P$  be  $l$ . Suppose we know the classification of  $v$  w.r.t  $P$ . We need to determine the ordering  $O$  of  $v_1, v_2$  on  $L$  which is consistent with the classification of  $v$ . Note

that the ordering of  $v_1, v_2$  on  $L$  depends on the classification of  $v$ . See Fig 4.2(a).

Define a transformation called *max translation* as follows. Translate the plane  $P : ax + by + cz + d = 0$  to  $P_{maxtranslate} : ax + by + cz - M = 0$  if  $d > 0$ , or to  $P_{maxtranslate} : ax + by + cz + M = 0$  if  $d \leq 0$ . Note that  $P_{maxtranslate}$  is the plane  $P$  translated by the amount  $M + |d|$ . In the first case  $P$  is translated to its positive side and in the latter  $P$  is translated to its negative side. Let  $v'_1, v'_2$  denote the intersection points of the lines containing the edges  $e_1, e_2$  with the plane  $P_{maxtranslate}$  and  $L'$  denote the directed line  $P_{maxtranslate} \cap f$ .

*Case(i)*: Classification of  $v$  is same as its actual position w.r.t the plane  $P$ . See Figure 4.2(b) and 4.2(c). Transform the plane  $P$  to  $P_{maxtranslate}$ . If  $P$  is translated by more than  $l$  to the same side in which  $v$  lies, the ordering of  $v_1, v_2$  is opposite to that of  $v'_1, v'_2$ , where  $l$  is the distance between  $P$  and  $v$ . Conversely, if  $P$  is translated by any amount to the side which does not contain  $v$ , the ordering of  $v_1, v_2$  is same as that of  $v'_1, v'_2$ .

*Case(ii)*: Classification of  $v$  is opposite to that of its actual position w.r.t  $P$ . Transform the plane  $P$  to  $P_{maxtranslate}$ . If  $P$  is translated by any amount to the same side in which  $v$  has been decided to lie in, the ordering of  $v_1, v_2$  is opposite to that of  $v'_1, v'_2$ . Conversely, if  $P$  is translated by more than  $l$  to the side in which  $v$  has been decided not to lie in, the ordering of  $v_1, v_2$  is same as that of  $v'_1, v'_2$ .

In both cases, if  $P$  is translated by more than  $l$ , the ordering of  $v_1, v_2$  can be determined from the ordering of  $v'_1, v'_2$ . The ordering of  $v'_1, v'_2$  can be determined exactly if the distance  $d'$  between them is greater than  $\delta$ . Let  $l'$  be the distance between  $v$  and the plane  $P_{maxtranslate}$ . From simple geometry, one can see that  $l' \sin \alpha \geq \delta$  is a sufficient condition for  $d'$  to be greater than  $\delta$ .  $P$  is translated by at most  $l + l'$ . Hence,  $l + l' \leq M + |d|$ . This implies

$$M + |d| \geq l + \frac{\delta}{\sin \alpha}$$

is a sufficient condition for determining the ordering of  $v'_1, v'_2$  exactly. Since,  $\min |d| = 0$  and  $\max |d| = \delta$ , we have

$$M \geq \delta + \frac{\delta}{\sin \alpha}$$

or

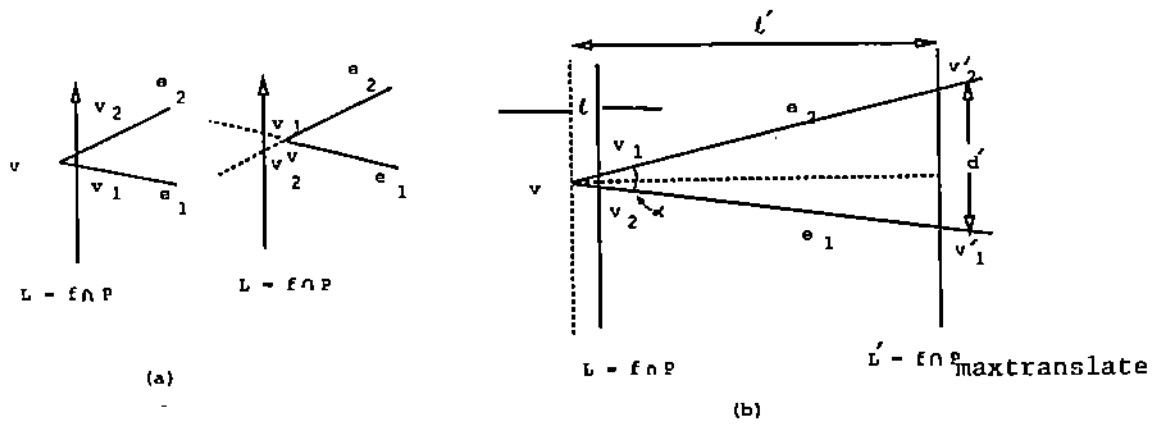
$$\frac{M}{2} \geq \frac{\delta}{\sin \alpha}$$

is a sufficient condition for determining the ordering of  $v'_1, v'_2$  exactly. The value of  $M$  is chosen to satisfy the above relation. ♣

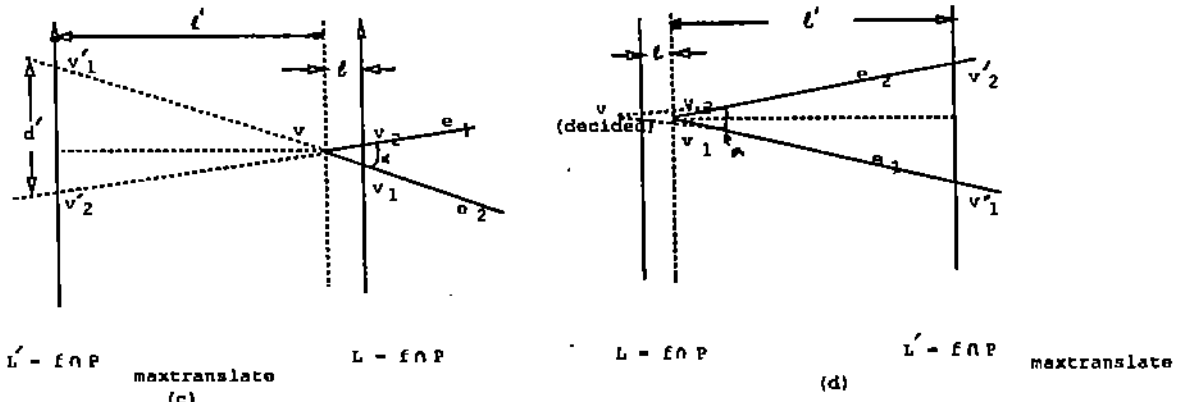
**Nesting of Polygons with Finite Precision Arithmetic:** The polygon nesting problem as discussed in section 2 can be solved with finite precision arithmetic if the polygons are restricted to a class of polygons called *fleshy polygons*. A polygon  $P$  is called *fleshy* if there is a point inside  $P$  such that a square with center (intersection of square's diagonals) at that point and with sides of length  $64\epsilon B$  lies inside  $P$ .  $B$  and  $\epsilon$  have been defined earlier.

**Lemma 4.2:** 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:** See [3]. 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 [3]. ♣

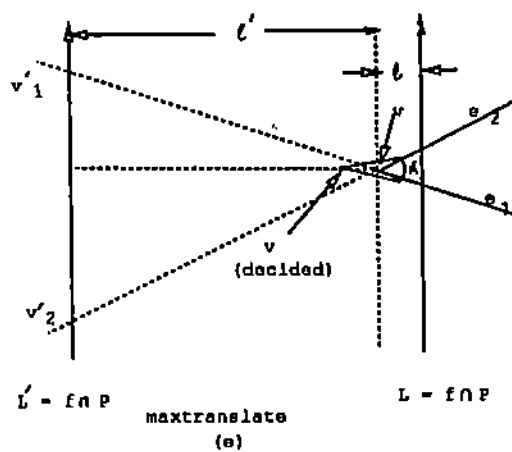


Case (i), P is translated to the side opposite to that in which v lies.



Case (i), P is translated to the side in which v lies.

Case (ii), P is translated to the side opposite to that in which v has been decided to lie in.



Case (ii), P is translated to the side in which v has been decided to lie in.

Figure 9: Maxtranslation and Lemma 4.1.

## 4.2 Description of the Algorithm

The same paradigm of cutting and splitting the polyhedron about the *cuts* is followed to produce the convex decomposition of a manifold, non-convex polyhedron. Choose one of the two planes incident on a *notch* as *notch plane*. This ensures that no new planes other than facet-planes are introduced by the algorithm and thus no additional error is introduced in the plane equations containing the facets. This also guarantees that any input assumption about the planes containing 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. For any *notch plane*  $P_g$  the two *cross sectional maps*  $GP_g^l, GP_g^r$  are constructed and the corresponding cuts  $Q_g^l, Q_g^r$  are computed in *Step I* as detailed below. In *Step II* we split the polyhedron about these cuts which completes the removal of *notch g*.

**Step I :**

**Constructing  $GP_g^l$  and  $GP_g^r$ :** The edges of  $GP_g^l$  and  $GP_g^r$  are either the edges transferred from polyhedron  $S$  called *old edges*, or edges newly generated from  $S \cap P_g$  called *new edges*. Note, all *new edges* will be present in both *cross sectional maps* while only some of the *old edges* may be present in either  $GP_g^l$  or in  $GP_g^r$ . As with the edges, some of the vertices of the *cross sectional maps* will be *old vertices* while some of them will be *new vertices*. To generate *old* and *new edges* on these *cross sectional maps*, compute the intersection points of each facet  $f$  with the *notch plane* using the vertex-plane, edge-plane, facet-plane classification as described before. After computing all intersection vertices (*new* and *old*) lying on the facet  $f$ , sort these vertices along the line of intersection  $f \cap P_g$ .

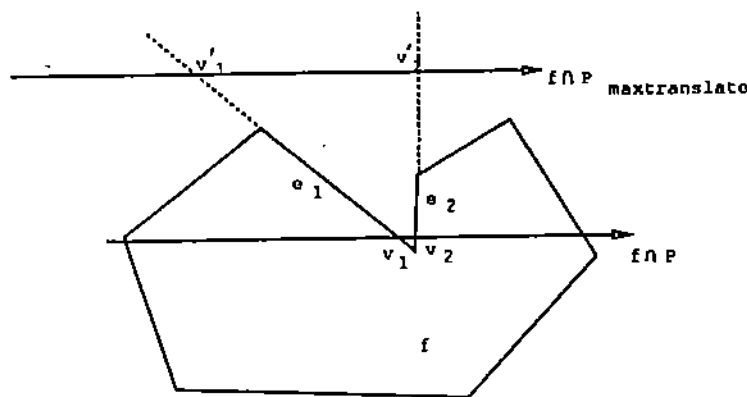


Figure 10: Consistent sorting of intersection points.

**Sorting of intersection points along line  $f \cap P_g$ :** Consider the facet  $f$  as shown in Figure 10. Let edges  $e_1$  and  $e_2$ , incident on  $v$  intersect the plane  $P_g$  at points  $v_1$  and  $v_2$ , both necessarily lying on line  $L = f \cap P_g$ . Further let  $v_1$  and  $v_2$  be *new vertices*. If  $v_1$  and  $v_2$  happen to be very close together, it may not be possible to determine their local ordering on  $L$  reliably. However, the classification of  $v$  w.r.t  $P_g$  can be used to decide this ordering consistently. Translate the plane  $P_g$  to  $P_{maxtranslate}$  and compute the points

$e_1 \cap P_{\max\text{translate}}$  and  $e_2 \cap P_{\max\text{translate}}$ . Let these intersection points be  $v'_1$  and  $v'_2$  respectively. As the angle between edges  $e_1$  and  $e_2$  cannot be arbitrarily small (minimum feature criteria for dihedral angles) there exists a certain translation such that the distance between  $v'_1$  and  $v'_2$  will be  $> \delta$ . Set the minimum dihedral angle  $\alpha_{\min}$  between any two facets to be such that  $\frac{\delta}{\sin \alpha_{\min}} \leq \frac{M}{2}$ . By Lemma 4.1, the ordering of  $v_1, v_2$  on  $L$  which is consistent with the classification of  $v$  can be determined exactly. The ambiguity in the ordering of *old vertices* and *new vertices* on the edges which are not incident on a common vertex does not arise if we assume minimum feature separation of at least  $\delta$  for elements of the input polyhedron  $S$ .

**Generating new edges:** Let  $L$  be the line of intersection of a facet  $f$  with the *notch plane*. Let  $(v_1, v_2, \dots, v_k)$  be the sorted sequence of vertices on  $L$ , corresponding to the points of intersection between the facet and the *notch plane*. One needs to decide consistently whether there should be an edge between two consecutive vertices  $v_i$  and  $v_{i+1}$  of this sorted sequence. This is done by scanning these sorted vertices from one end to the other and deciding whether we are “inside” or “outside” the facet. It is easy to see that if  $v_i$  is a *new vertex* then there would be an edge between  $v_i$  and  $v_{i+1}$  if there were no edge between  $v_{i-1}$  and  $v_i$  and vice versa. But if  $v_i$  is an *old vertex* there can be edge between  $v_i$  and  $v_{i+1}$  disregard of the presence of an edge between  $v_{i-1}, v_i$ .

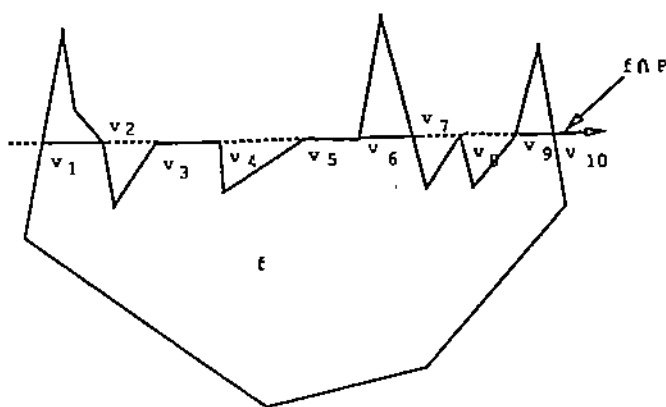


Figure 11: Generating *new* and *old* edges.

Toggling between “inside” and “outside” of the facet is carried out properly, even with degeneracies, using a *multiplicity code* at each intersection vertex. Scan the sorted sequence of intersection vertices from one end to the other and maintain a counter which is incremented by the *multiplicity code* at each vertex. Toggle between “inside” and “outside” of the facet as the counter toggles between “odd” and “even” count. For a *new vertex* put a *multiplicity code* of 1. For an *old vertex*, put a *multiplicity code* of 1 if two incident edges on the vertex on that facet lie in different half-spaces of  $P_g$  and put a *multiplicity code* of 2 if they lie in the same half-space. If there is an *old edge* between two vertices  $v_i$  and  $v_{i+1}$ , put *multiplicity codes* on them as follows. If other two incident edges on  $v_i, v_{i+1}$  on the facet  $f$  lie in the same half-space of the *notch plane*, put a *multiplicity code* of 1 on both the vertices  $v_i$  and  $v_{i+1}$ . Otherwise, put *multiplicity codes* of 1 and 2 on  $v_i$  and  $v_{i+1}$  in any order. In Figure 11, there is an *old edge* between  $v_3, v_4$ . The status (“outside”) with which one enters the vertex  $v_3$  is same as that one with which one leaves the vertex  $v_4$ . This is enforced by putting a *multiplicity code* of 1 on the two vertices which increment the counter by an “even” amount and prevent it from toggling. There is another *old edge* between  $v_5$  and

$v_6$ . The status ("outside") with which one enters the vertex  $v_5$  is different from the one with which one leaves the vertex  $v_6$ . This is enforced by putting *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. Initially, the counter is set to 0. Create a *new edge* from vertex  $v_i$  to  $v_{i+1}$  if the count is "odd" after leaving the vertex  $v_i$ . In case, there is an *old edge* between  $v_i$  and  $v_{i+1}$ , skip creating any *new edge* between them. An *old edge* may be transferred to  $GP_g^l$  or  $GP_g^r$  or to both. Transferring of *old edges* is described below.

**Transfer of old edges:** The *old edge*  $e_o$  should be transferred to  $GP_g^l$  ( $GP_g^r$  respectively.) if any facet (or a part of it) adjacent to  $e_o$  which has not been decided to be on the *notch plane*, gets transferred to  $GP_g^l$  ( $GP_g^r$  respectively.). For example, the edge  $g$  in Figure 3 should be transferred to  $GP_g^l$  but not to  $GP_g^r$ . For each *old edge*  $e_o$  decided to be on the plane  $P_g$ , check all of its *oriented edges* on different facets which have not been decided to be on the *notch plane*. Suppose  $f_o$  is such a facet. Classify any vertex  $v_o$  of  $f_o$  w.r.t the oriented edge  $e_o$  on  $f_o$ . If it is on the same side of  $e_o$  in which  $f_o$  lies then  $e_o$  should be transferred to  $GP_l$  ( $GP_r$  respectively.) if  $v_o$  has been classified to lie in  $P^+$  ( $P^-$  respectively.). It is trivial to decide the side of  $e_o$  in which  $f_o$  lies from the *oriented edge* of  $e_o$  on  $f_o$ .

Consistent vertex-plane, edge-plane and facet-plane classification takes 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  $O(1)$  time to determine the intersection points of  $S$  with the *notch plane*  $P_g$ . The sorting of intersection points on the facets adds  $O(r \log r + q')$  time where  $q'$  is the total number of facets decided to be intersected by the *notch plane*. The above bound follows from the fact that any line segment intersects a facet having  $r_i$  *reflex vertices* in no more than  $(2r_i + 2)$  points (Lemma 2.4). Once the construction of the maps  $GP_g^l$  and  $GP_g^r$  is done, it is trivial to recognize the boundary  $B_g$  containing the *notch*  $g$ . The methods as described in section 3 can be used to determine the *interesting boundaries*. Note that, if  $B_g$  is an inner boundary, the *interesting boundaries* consist of all the *ancestors* of  $B_g$ . If all the polygons in the *cross sectional maps* are *fleshy*, ancestors of  $B_g$  can be determined exactly using Lemma 3.4 of [3] at a cost of  $O(u')$  where  $u'$  is the number of vertices on the *cross sectional maps*. As discussed earlier, there are  $O(t)$  polygons and *monotone chains* in the *interesting boundaries* 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 4.2, 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*. Set up a safe minimum feature separation between polyhedral features so that the polygons generated in the *cross sectional maps* are always *fleshy*. Detection of children and parent of the polygon containing the *notch*  $g$  in effect, determines the inner and outer boundaries of  $Q_g^l(Q_g^r)$ . Obviously  $q' = O(p)$  and  $u = O(u') = O(p)$ . Combining the complexities of computing the edges of  $GP_g^l$  ( $GP_g^r$  respectively.) and detecting the inner and outer boundaries of  $Q_g^l(Q_g^r$  respectively.), we conclude that  $Q_g^l(Q_g^r$  respectively.) can be computed in  $O(p + t^2 + u'(t + \log u'))$  time.

**Step II:**  $S$  is separated corresponding to the cut  $Q_g^l(Q_g^r)$  by splitting the facets which are intersected by the *cut*  $Q_g$ . Let  $f_i$  be such a facet which is to be split at  $a_1, a_2, \dots, a_k$ . For each such point of intersection which may correspond to a *new vertex* or an *old vertex*, do the following.

**New Vertex:** Let  $e_s = (v_1, v_2)$  be the edge on which *new vertex*  $v_n$  lies. Generate edges between  $v_1, v_n$  and between  $v_2, v_n$ . Since the half spaces in which  $v_1$  and  $v_2$  lie are known, one can decide the half space in which each such *new edges* lies.

**Old Vertex:** For each *old vertex*  $v_o$  lying on the plane  $P_g$ , transfer the edges connected to  $v_o$  to the half



space in which their other vertex has been decided to lie in. Here, transferring means connecting those edges to the copy of the vertex  $v_o$  on the corresponding cut. The edges connected to  $v_o$  which have been decided to be on the plane  $P_g$  are transferred by procedure as described before. Finally, create two facets corresponding to the cuts  $Q_g^l$  and  $Q_g^r$ . Splitting each facet which are decided to be intersected by the cut  $Q_g^l$  ( $Q_g^r$ ) effectively either splits  $S$  into separate pieces or splices it about the cuts creating two facets corresponding to the cuts at the same geometric location. A depth first search starting from one vertex in each of  $P_g^+$  and  $P_g^-$  resolves this ambiguity and also collects all the features pertinent to each piece. Certainly, this separation step does not take more than  $O(p)$  time where  $p$  is the number of edges of  $S$ . Combining the time and space complexities of *Step I* and *Step II* we have the following Lemma.

**Lemma 4.3** Using heuristics to avoid conflicting decisions, a manifold polyhedron  $S$  with arbitrary genus, shells and certain minimum feature separations can be partitioned under finite precision arithmetic with a *notch plane* in  $O(p + t^2 + u'(t + \log u'))$  time and  $O(p)$  space, where  $p$  is the number of edges in  $S$ ,  $u'$  is the number of vertices on the *cross sectional maps* and  $t$  is the number of *notches* intersected by the *notch plane*.

The following combinatorial Lemma is used to derive the time complexity in Theorem 4.1.

**Lemma 4.4.** Let  $S_1, S_2, \dots, S_k$  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, and let  $u'_i$  be the total number of vertices on 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 on the *cross sectional maps* in  $S_1, S_2, \dots, S_k$ .

**Proof:** Consider the *cross sectional map*  $GP_g^l$  ( $GP_g^r$ ). The lines of intersection between  $P_g$  and other *notch planes*, called the *notch lines* divide this map into smaller facets which are present on the *cross sectional maps* in  $S_1, S_2, \dots, S_k$  i.e. on  $\cup_{i=1}^k GP_{g_i}^l$  ( $\cup_{i=1}^k GP_{g_i}^r$ ). The vertices on  $\cup_{i=1}^k GP_{g_i}^l$  ( $\cup_{i=1}^k GP_{g_i}^r$ ) can be partitioned into three sets, viz.,  $T_1, T_2$  and  $T_3$ . The set  $T_1$  consists of vertices which are created by intersections between *notch lines*. The set  $T_2$  consists of vertices on  $GP_g^l$  ( $GP_g^r$ ) and the set  $T_3$  consists of vertices which are created by intersections between edges of  $GP_g^l$  ( $GP_g^r$ ) and *notch lines*. Since there are at most  $O(r)$  *notch lines*,  $|T_1| \leq r^2$ . Certainly,  $|T_2| \leq n$ . By Lemma 2.4, each *notch line* can intersect  $GP_g^l$  ( $GP_g^r$ ) in at most  $(2r + 2)$  points since  $GP_g^l$  ( $GP_g^r$ ) can have at most  $O(r)$  *reflex vertices*. This gives  $|T_3| \leq 2r + 2$ . Thus,

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

**Theorem 4.1** Using heuristics to avoid conflicting decisions, a polyhedron  $S$  with arbitrary number of holes and shells and certain minimum feature separations can be decomposed under finite precision arithmetic into  $O(r^2)$  convex pieces in  $O(nr^2 + nr \log n + r^3 \log n + r^4)$  time and in  $O(nr)$  space, where  $r$  is the number of *notches*,  $n$  is the number of edges in  $S$ .

**Proof:** Let  $S$  be a manifold polyhedron. At a generic instance of the algorithm, let  $S_1, S_2, \dots, S_k$  be the

$k$  distinct (non-convex) polyhedra in the current decomposition, which contain the *subnotches* of a *notch*  $g$  which is to be removed. Let  $p_i$  be the number of edges in  $S_i$ ,  $u'_i$  be the number of vertices on the *cross sectional maps* 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  $t = O(r)$ . Using Lemma 4.3, we can say that the time  $\mathfrak{F}$  to remove the *notch*  $g$  is given by

$$\begin{aligned}\mathfrak{F} &= O\left(\sum_{i=1}^k (p_i + t_i^2 + u'_i(t_i + \log u'_i))\right) \\ &= O(p + r^3 + u'r + u'\log u').\end{aligned}$$

By Lemma 4.4,  $u' = O(n + r^2)$ . This gives,

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

To carry out removal of  $r$  *notches* we need  $O(nr^2 + nr\log n + r^3\log n + r^4)$  time. Obviously, the space complexity is  $O(p) = O(nr)$ . If  $S$  is a non-manifold polyhedron, remove all *special notches* from  $S$  to produce manifold polyhedra and decompose each such polyhedron into convex pieces as discussed in the previous section. The complexity remains same for this case. ♣

## 5 Conclusion

We have implemented our polyhedral decomposition algorithm under floating point arithmetic in Common Lisp on a Symbolics 3650. The numerical computations are all in C, callable from Lisp. We used  $\delta = 2^{-17}$  in the 32 bit machine with precision  $2^{-25}$ . Simple examples are shown in Figure 12. The experimental results have been very satisfying. Test polyhedra were generated by SHILP solid model creation software.

Our next goal is to develop a robust and stable algorithm for polyhedral decomposition problem. To find a robust and stable algorithm for this problem seems to be quite hard. It may be worthwhile to consider the concept of *pseudo facets*, the counterpart of *pseudo lines* in three dimensions to solve this problem.

**Acknowledgements:** We thank two anonymous referees for their astute comments. A preliminary version of this paper appeared in *Proc. of the Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Springer Verlag, No. 405, 1989, pp. 267-279.*

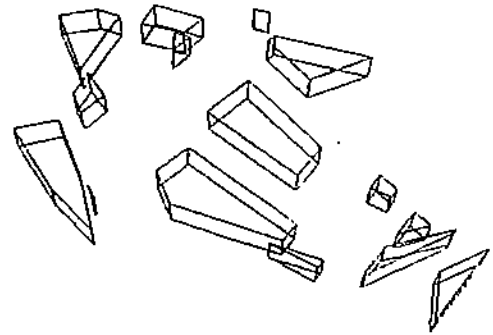
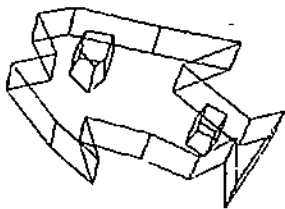
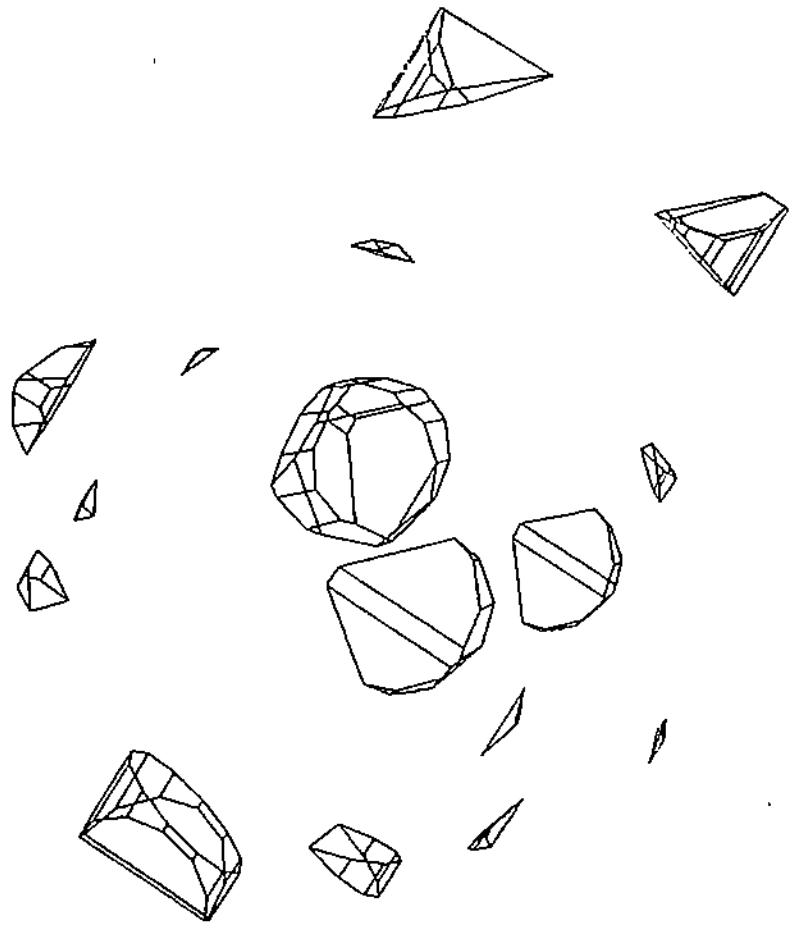
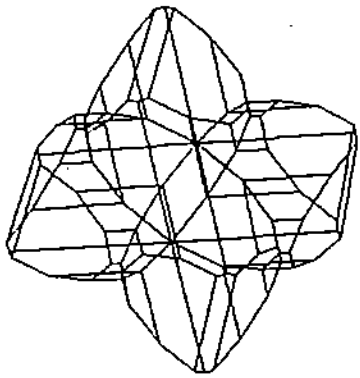


Figure 12: Examples.

## References

- [1] Bajaj, C., (1989) "Geometric Modeling with Algebraic Surfaces", *The Mathematics of Surfaces III*, edited by D. Handscomb, Oxford University Press, 3 - 48.
- [2] Anupam, V., Bajaj, C., Dey, T., Fields, M., Ihm, I., Klinkner, S., (1989), "The SHILP solid model creation. editing and display toolkit", Manuscript.
- [3] Bajaj, C., and Dey, T., (1989) "Polygon Nesting and Robustness" *Proc. Intl. Workshop on Discrete Algorithms and Complexity*, Fukuoka, Japan, 33-40. To appear in *Information Processing Letters*.
- [4] Chazelle, B., (1980), "Computational Geometry and Convexity", *Ph.D. Thesis*, CMU-CS-80-150, Computer Science, Carnegie-Mellon University.
- [5] Chazelle, B., (1984), "Convex Partitions of Polyhedra: A Lower Bound and Worst-case Optimal Algorithm", *SIAM J. on Computing*, Vol. 13, No. 3, pp. 488-507.
- [6] Chazelle, B., and Palios, L., (1989), "Triangulating a Non-convex Polytope" *Proc. of the 5th ACM Symposium on Computational Geometry*, Saarbrucken, West Germany, 393-400.
- [7] Dobkin, D., and Silver, D., (1988), "Recipes for Geometry and Numerical Analysis", *Proc. of the Fourth ACM Symposium on Computational Geometry*, Urbana, Illinois, 93 - 105.
- [8] Edelsbrunner, H., (1987), "Algorithms in Combinatorial Geometry", Springer Verlag.
- [9] Edelsbrunner, H., and Mucke, P., (1988), "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms" *Proc. of the Fourth ACM Symposium on Computational Geometry*, Urbana, Illinois, 118-133.
- [10] Fortune, S., (1989) "Stable Maintenance of Point-set Triangulations in Two Dimensions", *Proc. 30th IEEE Symposium on the Foundations of Computer Science*, 494 - 499.
- [11] Guibas, L., Salesin, D., and Stolfi, J., (1989) "Building Robust Algorithms from Imprecise Computations", *Proc. 1989 ACM Symposium on Computational Geometry*, Saarbucher, West Germany, 208-217.
- [12] Hoffmann, C., Hopcroft, J., and Karasick, M., (1987), "Robust Set Operations on Polyhedral Solids", Dept. of Computer Science, Cornell University, Technical Report 87-875.
- [13] Hoffmann, K., Mehlhorn, K., Rosenstiehl, P., and Tarjan, R., (1986), "Sorting Jordan Sequences in Linear Time using Level Linked Search Trees", *Information and Control*, 68, 170 - 184.
- [14] Hopcroft, J., Kahn, P., "A Paradigm for Robust Geometric Algorithms", Computer Science Tech. Report, Cornell University, TR 89-1044.
- [15] Karasick, M., (1988) "On the Representation and Manipulation of Rigid Solids", Ph.D. Thesis. McGill University.
- [16] Lingas, A., (1982), "The Power of Non-Rectilinear Holes", *Proc. 9th Intl. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Springer Verlag, 369 - 383.
- [17] Milenkovic, V., (1988), "Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic", Ph.D. Thesis, CMU Tech. Report CS-88-168, Carnegie Mellon Univ., Pittsburgh.

- [18] O'Rourke, J., and Supowit, K., (1983), "Some NP-hard Polygon Decomposition Problems", *IEEE Trans. Inform. Theory*, 29, 181 - 190.
- [19] Requicha, A.A.G., (1977), "Mathematical Models of Rigid Solid Objects", Tech. Memo 28, Production Automation Project, University of Rochester, Rochester, NY.
- [20] Rupert, J., and Seidel, R., (1989) "On the Difficulty of Tetrahedralizing Three Dimensional Non-convex Polyhedra", *Proc. of the Fifth ACM Symposium on Computational Geometry*, Saarbrucken, West Germany, 380 - 392.
- [21] Segal, M., and Sequin, C.. (1985), "Consistent Calculations for Solids Modeling", *Proc. of the First ACM Symposium on Computational Geometry*, 29 - 38.
- [22] Sugihara, K., (1988), "A Simple Method of Avoiding Numerical errors and Degeneracy in Voronoi diagram Constructions", Research Memorandum RMI 88-14, Department of Mathematical Engineering and Instrumentation Physics, Tokyo University.
- [23] Sugihara, K., and Iri, M., (1989), "A Solid Modeling System Free from Topological Consistency", Research Memorandum RMI 89-3, Department of Mathematical Engineering and Instrumentation Physics, Tokyo University.
- [24] Yap, C.. (1988) "A Geometric Consistency Theorem for a Symbolic Perturbation Theorem" *Proc. of the Fourth ACM Symposium on Computational Geometry*, Urbana, Illinois, 134-142.