SIAM J. COMPUT.
Vol. 21, No. 2, pp. 339–364, April 1992

010

CONVEX DECOMPOSITION OF POLYHEDRA AND ROBUSTNESS*

CHANDERJIT L. BAJAJ† AND TAMAL K. DEY†

Abstract. This paper presents a simple algorithm to compute a convex decomposition of a nonconvex polyhedron of arbitrary genus (handles) and shells (internal voids). For such a polyhedron S with n edges and r *notches* (features causing nonconvexity in polyhedra), the algorithm produces a worst-case optimal $O(r^2)$ number of convex polyhedra S_i, with $\cup_{i=1}^{k} S_i = S$, in $O(nr^2 + r^{7/2})$ time and $O(nr + r^{5/2})$ space. Recently, Chazelle and Palios have given a fast $O((n + r^2)\log r)$ time and $O(n + r^2)$ space algorithm to tetrahedralize a nonconvex polyhedron. Their algorithm, however, works for a simple polyhedron of genus zero and with no shells (internal voids). The algorithm, presented here, is based on the simple cut and split paradigm of Chazelle. With the help of zone theorems on arrangements, it is shown that this cut and split method is quite efficient. The algorithm is extended to work for a certain class of nonmanifold polyhedra. Also presented is an algorithm for the same problem that uses clever heuristics to overcome the numerical inaccuracies under finite precision arithmetic.

Key words. computational geometry, robust computations, geometric modeling, finite element analysis, computational complexity

AMS(MOS) subject classifications. 68U05, 65Y25, 68Q25

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 [5], [12]. Convex decompositions lead to efficient algorithms, for example, in geometric point location and intersection detection; see [12]. Our motivation stems from the use of geometric models in SHILP,[1] a solid model creation, editing, and display system developed at Purdue [1]. 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.

The surface δS of a polyhedron S is called a 2-manifold if each point on δS has an ϵ-neighborhood that is homeomorphic to an open 2D ball or half-ball [2]. Polyhedra, having 2-manifold surfaces are called manifold polyhedra. Nonmanifold polyhedra may have incidences as illustrated in Fig. 1. Manifold polyhedra with holes are homeomorphic to torii with one or more handles. Manifold polyhedra with *internal voids* are homeomorphic to three-dimensional annuli, that is, spheres with internal voids.

We represent polyhedra with their boundaries, which consist of zero-dimensional faces, called vertices; one-dimensional faces, called edges; and two-dimensional faces, called facets. A *reflex edge* of a polyhedron is an edge where the inner dihedral angle subtended by two incident facets is greater than 180°. Manifold polyhedra can be nonconvex only due to reflex edges. Notches in manifold polyhedra refer to reflex edges only. In nonmanifold polyhedra, however, notches refer to other types of incidences as well; see Fig. 1.

The problem of partitioning a nonconvex polyhedron S into a minimum number of convex parts is known to be NP-hard [22], [24]. Rupert and Seidel [25] also show that

*Received by the editors December 20, 1989; accepted for publication (in revised form) April 1, 1991. A preliminary version of this paper appeared in Proc. Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 405, Springer-Verlag, 1989, pp. 267–279. This research was supported in part by National Science Foundation grant CCR 90-02228, Office of Naval Research contract N00014-88-K-0402, and Air Force Office of Scientific Research contract 91-0276.

†Department of Computer Science, Purdue University, West Lafayette, Indiana 47907.

[1] SHILP stems from the Sanskrit word SHILP-SHASTRA, the science of sculpture.

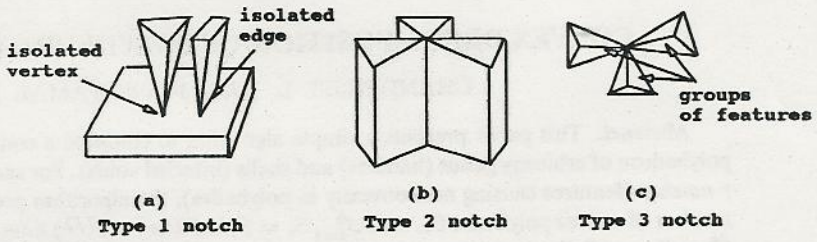

FIG. 1. *Nonmanifold incidences or special notches.*

the problem of determining whether a nonconvex 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 [5], [6] 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 in $O(nr^2)$ space. Recently, Chazelle and Palios [7] have given an $O((n + r^2)\log r)$ time and $O(n + r^2)$ space algorithm to tetrahedralize a subclass of nonconvex 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).

Results. In §3, we present an algorithm to compute a disjoint convex decomposition of a manifold polyhedron S that 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 $\cup_{i=1}^{k} S_i = S$ in $O(nr^2 + r^{7/2})$ time and in $O(nr + r^{5/2})$ space. We extend this algorithm to work for nonmanifold polyhedra that do not have abutting edges or facets but may have incidences as illustrated in Fig. 1. The algorithm presented in this paper is based on the repeated cutting and splitting of polyhedra with planes that resolve notches. Chazelle, in [5], first used this method. We improve this method to obtain better time and space bounds based on a refined complexity analysis and the use of efficient algorithms for certain subproblems. In §4, we give an algorithm for the same convex decomposition problem that uses sophisticated heuristics based on geometric reasoning to overcome the inaccuracies involved with finite precision arithmetic computations. This algorithm runs in approximately $O(nr^2 + nr\log n + r^4)$ time and $O(nr + r^{5/2})$ space.

2. Preliminaries.

2.1. Notches. Our algorithm applies to polyhedra that are nonconvex due to the presence of the following four features, called *notches*.

1. *Type* 1 *notches*: These notches are caused by isolated vertices and edges on a facet. An isolated vertex or edge on a facet is not adjacent to any other edge of the facet. See Fig. 1(a).
2. *Type* 2 *notches*: These notches are caused by the edges along which more than two facets meet, as illustrated in the Fig. 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 Fig. 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, and type 3 are called *special notches*, as they are present only in nonmanifold polyhedra. In our algorithm, we first remove all special notches from S, creating only manifold polyhedra. Subsequently, type 4 notches of the manifold polyhedra are removed by repeatedly cutting and splitting the polyhedra 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 notch. Clearly, for each notch g, there exist infinite choices for P_g. Note that P_g may intersect other notches, thereby producing *subnotches* as well. See Fig. 2.

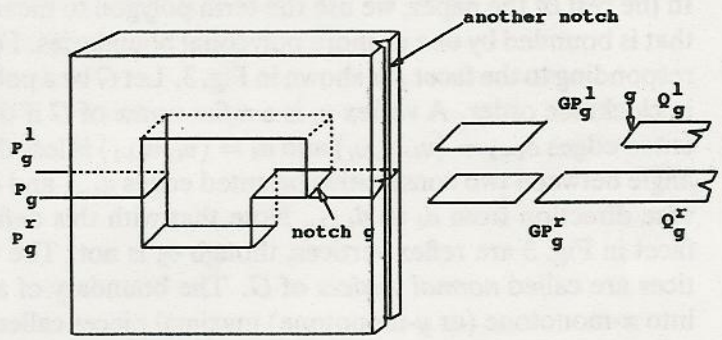

FIG. 2. *A notch and its notch plane, the cross-sectional map, and a cut.*

2.2. Data structure. Let S be a polyhedron, possibly with holes and shells, and having s vertices: $\{v_1, v_2, \cdots, v_s\}$, n edges: $\{e_1, e_2, \cdots, e_n\}$, and q facets: $\{f_1, f_2, \cdots, f_q\}$. These lists of vertices, edges, and facets of S are stored similarly to the *star-edge* representation of polyhedra [19].

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 pointers to the defining edge.
2. *orientededge.facet*: contains pointers 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.

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. The traversal of each oriented edge cycle always has the facet to the right. Each oriented edge cycle is represented with a linked list of orientededges on the cycle. If there is an isolated vertex on the facet (Fig. 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 nonmanifold polyhedron, a facet may have configurations as shown in Fig. 3 where a vertex or an edge is considered more than once in the oriented edge cycles, though an oriented edge is included only once.

2.3. Useful lemmas. Let the *polygonal boundary* refer to an oriented edge cycle embedded on a plane with no edge intersecting the other except at their endpoints. The traversal of a polygonal boundary may pass through an edge or a vertex more than once. In the rest of the paper, we use the term polygon to mean a *connected* region on a plane that is bounded by one or more polygonal boundaries. For example, such a polygon corresponding to the facet f is shown in Fig. 3. Let G be a polygon with vertices $v_1, v_2, \cdots, v_k$ in clockwise order. A vertex v_i is a *reflex vertex* of G 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 less than or equal to 180°. 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}. Note that with this definition, v_4, v_5 of the nonsimple facet in Fig. 3 are reflex vertices, though v_3 is not. The vertices that are not reflex vertices are called *normal vertices* of G. The boundary of a polygon G can be partitioned into x-monotone (or y-monotone) maximal pieces called *monotone chains*, i.e., vertices of a monotone chain have x-coordinates (or y-coordinates) in either strictly increasing or decreasing order. See Fig. 4.

In subsequent sections, we use the following lemmas.

LEMMA 2.1. *Let G be a polygon with r reflex vertices. The number of monotone chains c in G is bounded as $c \leq 6(1 + r)$.*

Proof. The proof follows from Theorem 3, [5, p. 22]. □

LEMMA 2.2. *Let G be a polygon with s normal vertices. There are at most $O(s)$ monotone chains in G.*

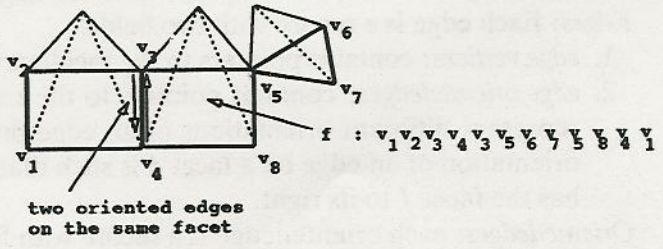

FIG. 3. *A nonsimple facet.*

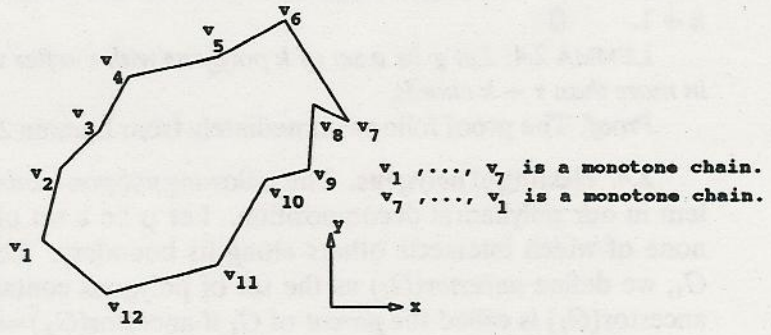

FIG. 4. *Monotone chains in a polygon.*

Proof. Let v be the vertex of G 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 G. Add six more edges to B, as shown in Fig. 5, to construct a new polygon G'. The polygon G' is oppositely oriented with respect to G. Note that each reflex vertex of G' corresponds to a normal vertex of G. Thus G' has no more than s reflex vertices, and according to Lemma 2.1, its boundary is partitioned into $O(s)$ monotone chains. The polygon G cannot have more monotone chains than G', which implies that G has $O(s)$ monotone chains. □

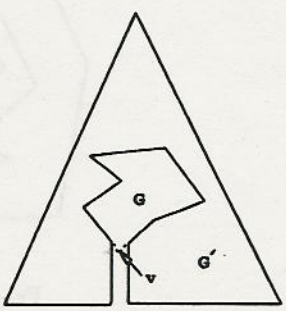

FIG. 5. *Constructing a polygon of opposite orientation.*

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

LEMMA 2.3. *Let G be a polygon (possibly with holes) with r reflex vertices. No line can intersect G 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 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 $r_1 + r_2 \leq k - 1$. Furthermore, the number of chords of a line L 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 the line 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 $k-1+1 \leq k+1$. □

LEMMA 2.4. *Let $\wp$ be a set of k polygons with r reflex vertices. No line can intersect $\wp$ in more than $r+k$ chords.*

Proof. The proof follows immediately from Lemma 2.3. □

2.4. Nesting of polygons. The following *polygon nesting* problem arises as a subproblem in our polyhedral decomposition. Let $\wp$ be a set of k polygons $G_i, i = 1, \cdots, k$, none of which intersects others along its boundary. 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$(G_i) - G_k$. Note 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. In Fig. 6, ancestor$(G_3) = \{G_1, G_2\}$, parent$(G_4) = (G_2)$, children$(G_2) = \{G_3, G_4\}$, ancestor(G_5) =null=children(G_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 of nonintersecting polygons.

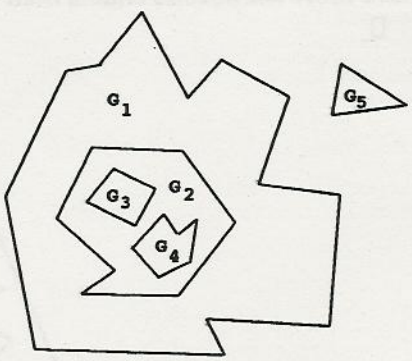

FIG. 6. *Nested polygons.*

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

Proof. See [4]. Though the algorithm given in [4] uses a slightly different type of monotone chains, called *subchains*, it also works for the monotone chains as defined in this paper. Further, the algorithm of [4] can be straightforwardly adapted to the input set of polygons as defined in this paper. □

3. Convex decomposition.

3.1. Sketch of the algorithm. Given a polyhedron S, it is first split along the 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 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 nonmanifold subpolyhedra with special notches. See Fig. 7. As before, these nonmanifold 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.

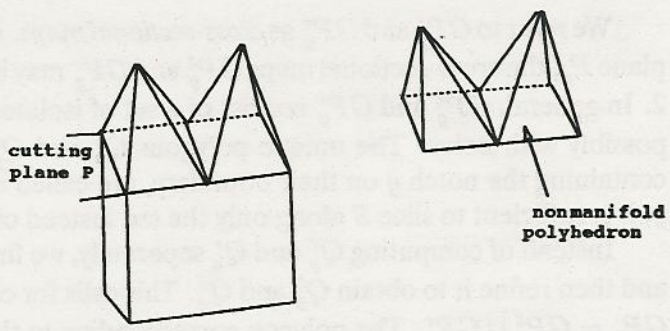

FIG. 7. *An example where the manifold property is not preserved after a cut.*

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 $g_1, g_2, \cdots, g_k$ be the subnotches of a notch g present in the polyhedra $S_1, S_2, \cdots, S_k$. Let P_g be the notch plane assigned to g. Remove $g_1, g_2, \cdots, g_k$ from $S_1, S_2, \cdots, S_k$ 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 §3.3. 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 §3.2.

3.2. Intersecting a manifold polyhedron with a notch plane. Let S be a manifold polyhedron with r notches and p edges. By S, we denote any polyhedron $S_1, S_2, \cdots, S_k$ 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

$$S^\ell = \text{cl}(\text{int}(P_g^\ell) \cap \text{int}(S)),$$
$$S^r = \text{cl}(\text{int}(P_g^r) \cap \text{int}(S)),$$

where $\text{cl}(O)$ and $\text{int}(O)$ 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 and 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, which are given by

$$GP_g^l = P_g \cap \delta S^\ell,$$
$$GP_g^r = P_g \cap \delta S^r.$$

We 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, Fig. 2. In general, GP_g^ℓ and GP_g^r consist of a set of isolated points, segments, and polygons, possibly with holes. The unique polygons Q_g^ℓ and 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 and 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 nonmanifold. See Fig. 7. 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 Fig. 2. In this case, two facets corresponding to Q_g^ℓ and Q_g^r are created that may overlap geometrically and be 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 §3.2.1 and Step II in §3.2.2.

3.2.1. Computation of the cut Q_g. *Step* A. First, all boundaries present in the cross-sectional map GP_g are computed. To do this, 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 $a_1, a_2, \cdots, a_k$ 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 we 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}, a_i.

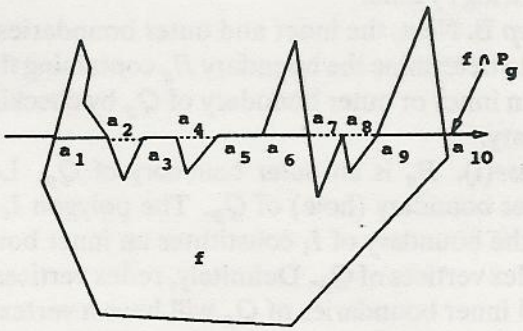

FIG. 8. *Generating new and old edges.*

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 another 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 Fig. 8, there is an old edge between a_3 and 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 the vertex a_i to a_{i+1} is created if the count is "odd" on leaving the 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 2.3, 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 that is normal. Since the boundary of I_i constitutes 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 Lemmas 2.1 and 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 2.5). 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.

3.2.2. 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 $a_1, a_2, \cdots, a_k$. The splitting of f consists of splitting the edges on which a new intersection vertex lies and the old intersection vertices. 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 relevant pointers. The facet f may be split into several subfacets. The inner boundaries of f that are not intersected by P_g remain 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. Elimination of special notches and its analysis. For a nonmanifold polyhedron S, nonconvexity results from four types of notches, as discussed in §2.1. Let S have n edges and r notches. The counting of special notches is described in §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 Fig. 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 $f_1, f_2, \cdots, f_{r_i}$. 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 $(f_1, f_2), (f_3, f_4), \cdots, (f_{r_i-1}, f_{r_i})$. 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 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, the 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 on 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.

3.4. Worst-case complexity analysis. Combining the costs of the "cut computation" of §3.2.1 and the "splitting operation" of §3.2.2 yields the following lemma.

LEMMA 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.*

The following two-dimensional subproblem is essential for the analysis of ConvDecomp. Let L be a set of r lines in two-dimensions that form a line arrangement A[12]. Let E be a set of edges removed from A such that all cells in $A - E$ are convex. Let us denote the new arrangement $A - E$ as A^-. Let C be a set of cells in A^- intersected by a line l. The total number of edges in the cells in C determines the zone complexity $z(l, A^-, r)$ of l in A^-. Of course, the contribution of a line in any single cell is counted only once, although it may have several consecutive segments on it in that cell. Let $q(r) = \max\{z(l, A^-, r) | l \text{ is any line in any such arrangement } A^-\}$. In Lemma 3.2 below, we derive a nontrivial upper bound for $q(r)$. Now suppose that a polyhedron S with n edges and r notches has been sliced with $y \leq r$ notch planes so far. Let $S_1, S_2, \cdots, S_k$ be the polyhedra in the current decomposition, where each S_i contains a subnotch g_i of a notch g in S. Let x_i be the number of edges on Q_{g_i}.

LEMMA 3.2. $x = \sum_{i=1}^{k} x_i = O(n + r^{3/2})$.

Proof. Consider the cut Q_g produced by the intersection of S with P_g. The region in Q_g is divided into smaller cells by the segments of *notch lines* produced by the intersection of other notch planes with P_g. It is important to note that consecutive segments of a notch line may have gaps in them. We focus on the cells $Q_{g_1}, Q_{g_2}, \cdots, Q_{g_k}$ adjacent to the subnotches $g_1, g_2, \cdots, g_k$ of the notch g.

Consider separately the set of notch line segments that divides Q_g. These line segments and the line L_g corresponding to the notch g produce an arrangement T of line segments on the notch plane P_g. Notice that the arrangement T can be thought of as

an arrangement A^- for some arrangement A of y lines. The cells 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. Note that in each single cell of Z_g, consecutive segments of a line form a single edge. Actually one can verify that this notion of edges is consistent with our notion of cuts. Overlaying Q_g on T produces $Q_{g_1}, Q_{g_2}, \cdots, Q_{g_k}$. See Fig. 9. These are the cells in $T \cup Q_g$ that are adjacent to the line L_g. Let V'_g and E'_g denote the sets of vertices and edges, respectively, in $Q_{g_1}, Q_{g_2}, \cdots, Q_{g_k}$. 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, $|T_1| = O(|E_g|) = O(q(y))$. 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 holes. 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 such successive edge segments of e. Charge one unit cost to the reflex vertex that lies to the left (or right) of each segment, and charge one 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 Fig. 9(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 our 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 $O(r_g + q(y))$, 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 $O(r_g + q(y))$ segments of edges in E_g contribute to E'_g. Hence $|T_3| = O(r_g + q(y))$.

Consider next the case where Q_g has holes. We refer to the polygon corresponding to a hole in Q_g as a *hole-polygon*. From Q_g create a polygon Q'_g that does not have any hole, by 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 Fig. 10 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

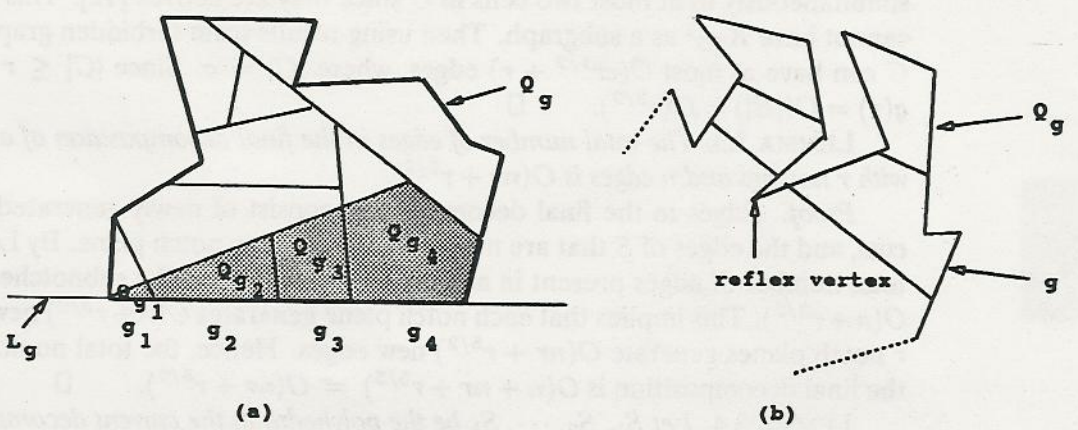

FIG. 9. *Superimposing a cut on an arrangement of notch line segments.*

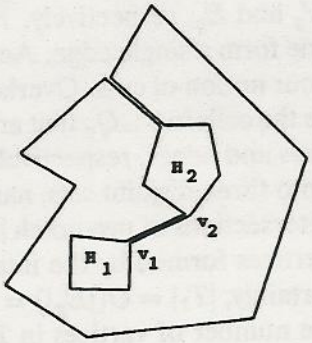

FIG. 10. *Merging polygons to create Q'_g from Q_g.*

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 $|T_3| \leq |T'_3| = O(u' + q(y))$.

Putting all these together, we have $|V'_g| = |T_1| + |T_2| + |T_3| = O(r_g + q(y) + u')$. Certainly, $r_g \leq r$, $y \leq r$, and $u' \leq n$. This gives $|V'_g| = O(n + q(r))$. Since $Q_{g_1}, Q_{g_2}, \cdots, Q_{g_k}$ form a planar graph, we have $x = |E'_g| = O(|V'_g|) = O(n + q(r))$. Now we show that $q(r) = O(r^{3/2})$, which completes the proof.

Let C be the set of cells intersected by a line in an arrangement A^- formed out of an arrangement A of r lines. Form a bipartite graph $G = (V_1 \cup V_2, E)$, where each node in V_1 corresponds to a cell in C and each node in V_2 corresponds to a line in A. An edge $e \in E$ connects two vertices $v_1 \in V_1$, $v_2 \in V_2$ if the line corresponding to v_2 contributes an edge to the cell corresponding to v_1. Observe that any four lines in A can contribute simultaneously to at most two cells in C since they are convex [12]. This means that G cannot have $K_{2,5}$[2] as a subgraph. Then using results from forbidden graph theory [20], G can have at most $O(cr^{1/2} + r)$ edges, where $|C| = c$. Since $|C| \leq r + 1$, we have $q(r) = O(|E|) = O(r^{3/2})$. □

LEMMA 3.3. *The total number of edges in the final decomposition of a polyhedron S with r notches and n edges is $O(nr + r^{5/2})$.*

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 3.2, the total number of edges present in all cuts corresponding to the subnotches of a notch is $O(n + r^{3/2})$. This implies that each notch plane generates $O(n + r^{3/2})$ new edges. Thus, r notch planes generate $O(nr + r^{5/2})$ new edges. Hence, the total number of edges in the final decomposition is $O(n + nr + r^{5/2}) = O(nr + r^{5/2})$. □

LEMMA 3.4. *Let $S_1, S_2, \cdots, S_k$ be the polyhedra in the current decomposition, where*

[2] It is a complete bipartite graph $G = (V_1 \cup V_2, E)$ where $|V_1| = 2$ and $|V_2| = 5$.

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 $S_1, S_2, \cdots, S_k$, 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 intersections of 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 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 2.4, 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 altogether. This gives $|T_3| = O(r^2)$. Thus,

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

□

As discussed in [6], one can always produce a worst-case optimal number ($O(r^2)$) of convex polyhedra by carefully choosing the notch planes.

LEMMA 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 [6] for the proof. □

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 $O(nr^2 + r^{7/2})$ time and $O(nr + r^{5/2})$ 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 3.5, 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 $S_1, S_2, \cdots, S_k$ 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 $t = \sum_{i=1}^{k} t_i$ and u_i be the number of vertices in GP_{g_i} of S_i and $u = \sum_{i=1}^{k} u_i$.

Applying Lemma 3.1, 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 $m = \sum_{i=1}^{k} m_i = O(nr + r^{5/2})$, $\sum_{i=1}^{k} r_i = O(r^2)$, $u = O(n+r^2)$, and since a notch plane can intersect at most $r-1$ notches giving $t = O(r)$, we have $O(\sum_{i=1}^{k}(m_i + t_i \log t_i + (u_i + r_i)\log r_i)) = O(nr + r^{5/2})$.

As described before, elimination of a notch may produce nonmanifold 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 nonmanifold 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 nonmanifold polyhedra produced as a result of cutting *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 $S_1, S_2, \cdots, S_k$.*

manifold polyhedra with notch planes can be carried out in totally $O(m) = O(nr + r^{5/2})$ time.

Thus, each notch elimination step takes $O(nr + r^{5/2})$ time, and Step 3 of ConvDecomp, which eliminates r notches, takes $O(nr^2 + r^{7/2})$ time. Combining the complexities of Step 2 and Step 3, we obtain an $O(nr^2 + r^{7/2})$ time complexity for convex decomposition of a manifold polyhedron. The space complexity of $O(nr + r^{5/2})$ follows from Lemma 3.3. □

THEOREM 3.2. *A nonmanifold 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 $O(nr + r^{5/2})$ 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 $S_1, S_2, \cdots, S_l$ 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 $O(\sum_{i=1}^{l} n_i r_i^2 + r_i^{7/2}) = O(nr^2 + r^{7/2})$ time and in $O(\sum_{i=1}^{l} n_i r_i + r_i^{5/2}) = O(nr + r^{5/2})$ space. □

4. Convex decomposition under finite precision arithmetic. 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 [19], [27]. We model the inexact arithmetic computations by ε-arithmetic [15], [17] where the arithmetic operations $+, -, \div, \times$ are performed with relative error of at most ε. Under this model, the absolute error in the 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, e.g., [23]. 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 uncertainty, 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 the independent decision points, any choice may be made from the finite set of local topological possibilities while the choices at the dependent decision points should ensure that they do not contradict any previous topological decisions. The algorithm that follows this paradigm would never fail, though it may not always compute a valid output. Such algorithms have been termed *parsimonious* by Fortune [15].

An algorithm under ε-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 [15], [16], [21], authors have given robust and stable algorithms for some important geometric problems in two dimensions. Except [18], 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 $\Re^3$. In [18], Hopcroft and Kahn discuss the existence of a valid polyhedron that admits the positions of the perturbed vertices of a convex polyhedron. The case of nonconvex polyhedra is perceived to be hard and requires understanding the deep interactions between topology and perturbations of polyhedral features of nonconvex polyhedra.

Karasick [19] 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 [19] and provide an algorithm for the problem of polyhedral decomposition that also uses geometric reasoning to avoid conflicting decisions. 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 more reliable in the presence of numerical errors in arithmetic computations. These heuristics are useful in the sense that they give better results in practice than the other algorithms, which assume exact arithmetic. We give an estimate of the worst-case running time bound for the algorithm under the ε-arithmetic model.

Related work. The issue of robustness in geometric algorithms has recently taken on added importance because of the increasing use of geometric manipulations in computer-aided design and solid modeling [3]. Edelsbrunner and Mucke [14] and Yap [29] suggest using expensive symbolic perturbation techniques for handling geometric degeneracies. Sugihara and Iri [28] and Dobkin and Silver [11] 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 estimate a priori for complex problems. Segal and Sequin [26] estimate various numerical tolerances, tuned to each computation, to maintain consistency. Milenkovic [23] presents techniques for computing the arrangements of a set of lines in two dimensions robustly. He introduces the concept of *pseudolines* that preserves some basic topological properties of lines and computes the arrangements in terms of these pseudolines. Karasick [19] proposes using geometric reasoning and applies it to the problem of polyhedral intersections. Sugihara [27] uses geometric reasoning to avoid redundant decisions and thereby eliminate topological inconsistencies in the construction of planar Voronoi diagrams. Guibas, Salesin, and Stolfi [17] propose a framework of computations, called ε-geometry, in which they compute an exact solution for a perturbed version of the input. So does Fortune [15], who applies it to the problem of triangulating two-dimensional point sets. For more details on robustness, see [8].

4.1. Intersection and incidence tests. In what follows, we assume the input polyhedra to be manifold. Nonmanifold polyhedra can be handled as discussed in the earlier sections. It is clear from the discussions of our algorithm in §3 that numerical computations arise in various intersections and incidence tests. 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., $a_i^2 + b_i^2 + c_i^2 = 1$.

Vertex-plane classification. To classify the incidence of a vertex $v_i = (x_i, y_i, z_i)$ with respect to 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 computations 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 with respect to 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 $\mathbf{r}$ of e_j and the plane P is determined by solving the linear system $A\mathbf{r} = \mathbf{d}$ where

$$A = \begin{bmatrix} a & b & c \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \quad \text{and} \quad \mathbf{d} = [-d, -d_1, -d_2,]^T .$$

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 $e_1 = (v_i, w_1), e_2 = (v_i, w_2), \cdots, e_k = (v_i, w_k)$.
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"
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.
```

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 with respect to 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 ∩ 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 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
    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 Fig. 11.*)
```

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

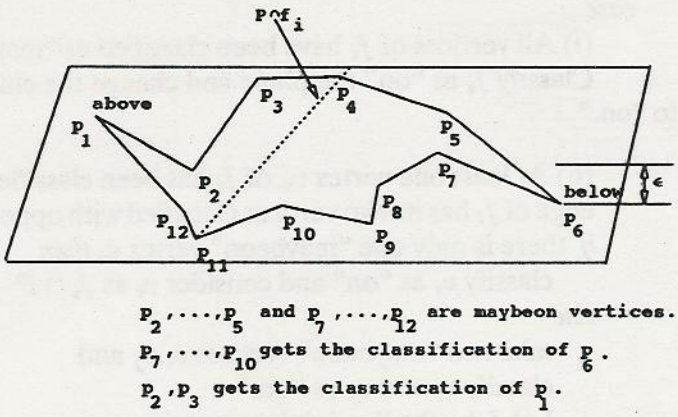

FIG. 11. *Case* (ii) *of the facet-plane classification.*

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.

Nesting of polygons with finite precision arithmetic. The polygon nesting problem as discussed in §2.4 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 the center (intersection of square's diagonals) at that point and with the sides of length $64\varepsilon B$ lies inside P. B and ε have been defined earlier.

LEMMA 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. See [4]. 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 [4]. □

4.2. Description of the algorithm. 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 meeting 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 §3. The polygon nesting algorithm, used for this purpose, is adapted to cope with the inexact numerical computations, as stated in Lemma 4.1. The polygon nesting algorithm with inexact arithmetic computations requires all input polygons to be fleshy. Though 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 with respect to 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. Though for the original polyhedron it is valid, it may not be preserved throughout the entire cutting process. The method described in [26] can be used to eliminate this problem.

Complexity analysis. We use Lemmas 2.3 and 3.4 in our analysis, which is 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 of 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 §3 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 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 $S_1, S_2, \cdots, S_k$ 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

given by

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

By Lemma 3.4, $u = O(n + r^2)$. This gives

$$\Im = O(p + r^3 + (n + r^2)r + (n + r^2)\log n)$$
$$= O(nr + n\log n + r^2 \log n + r^3)$$
$$= O(nr + n\log n + r^3).$$

To eliminate r notches, we need $O(nr^2 + nr\log n + r^4)$ time. Obviously, the space complexity is $O(p) = O(nr + r^{5/2})$. If S is a nonmanifold 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. □

5. Conclusion. 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 Figs. 12 and 13.

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 [1]. The convex pieces generated can be easily triangulated to generate a triangulation of the input nonconvex polyhedra. Of course, the facet triangulations between convex pieces have to be kept consistent. To achieve this, the slicing along a notch has to be carried out through all subpolyhedra intersecting the notch plane. The time complexity does not increase for decompositions with this type of slicing, though space complexity increases to $O(nr + r^3)$. Details can be found in [9].

In finite element methods with triangular elements, nicely shaped tetrahedra are preferred to reduce ill-conditioning as well as discretization error. In [10], we have given a method to produce guaranteed quality triangulations of convex polyhedra. In Fig. 14, an example of this triangulation is shown. For clarity, we show only the triangulations on the facets. The convex decomposition method, coupled with this guaranteed quality triangulation, gives a method for guaranteed quality triangulations of nonconvex polyhedra. 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 guaranteed quality triangulations of nonconvex polyhedra directly is more practical. Currently, research is going on to find such an algorithm.

intersected by the notch plane in S_i. Let $p = \sum_{i=1}^{k} p_i$, $u = \sum_{i=1}^{\kappa} u_i$, and $t = \sum_{i=1}^{\kappa} t_i$. 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

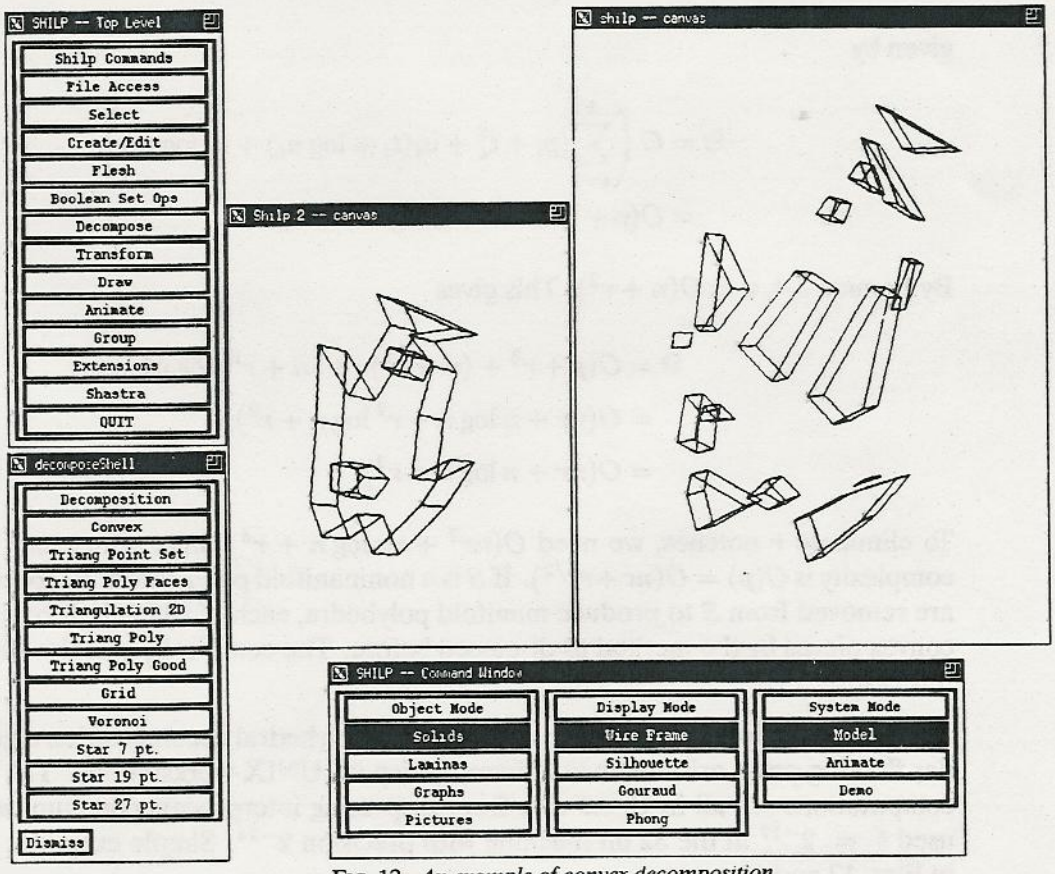

FIG. 12. *An example of convex decomposition.*

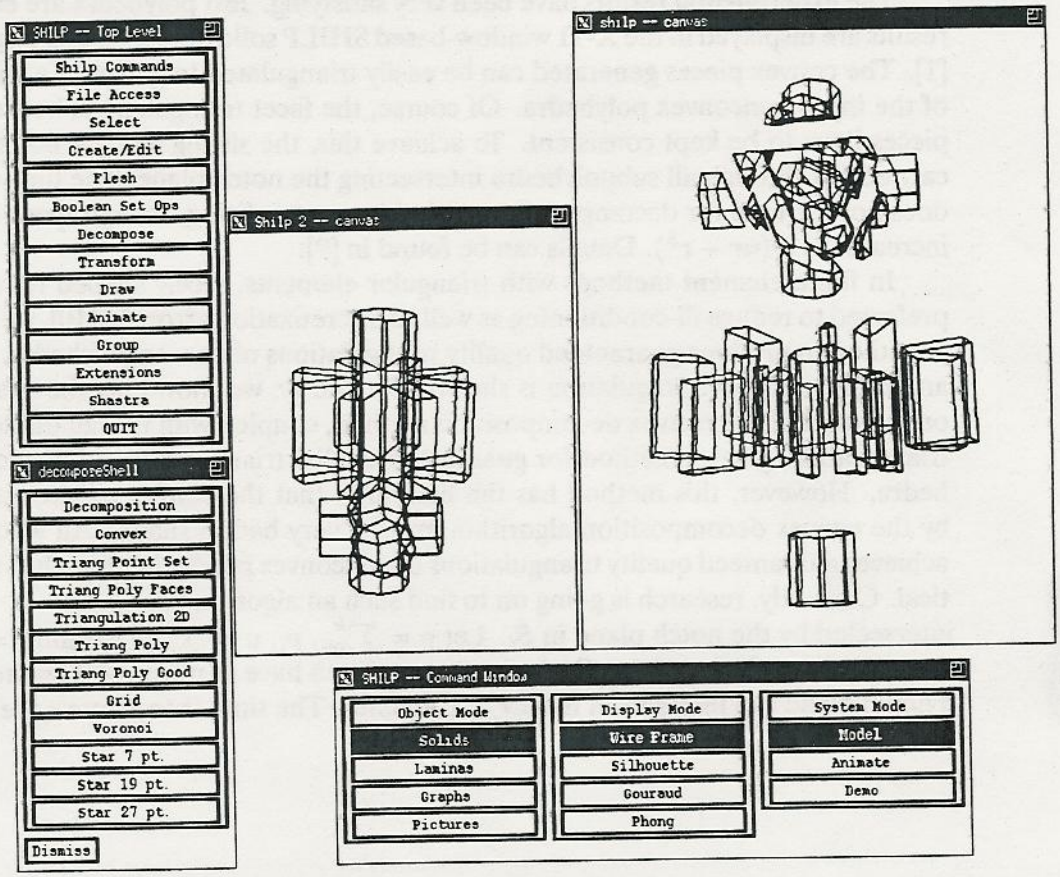

FIG. 13. *Another example of convex decomposition.*

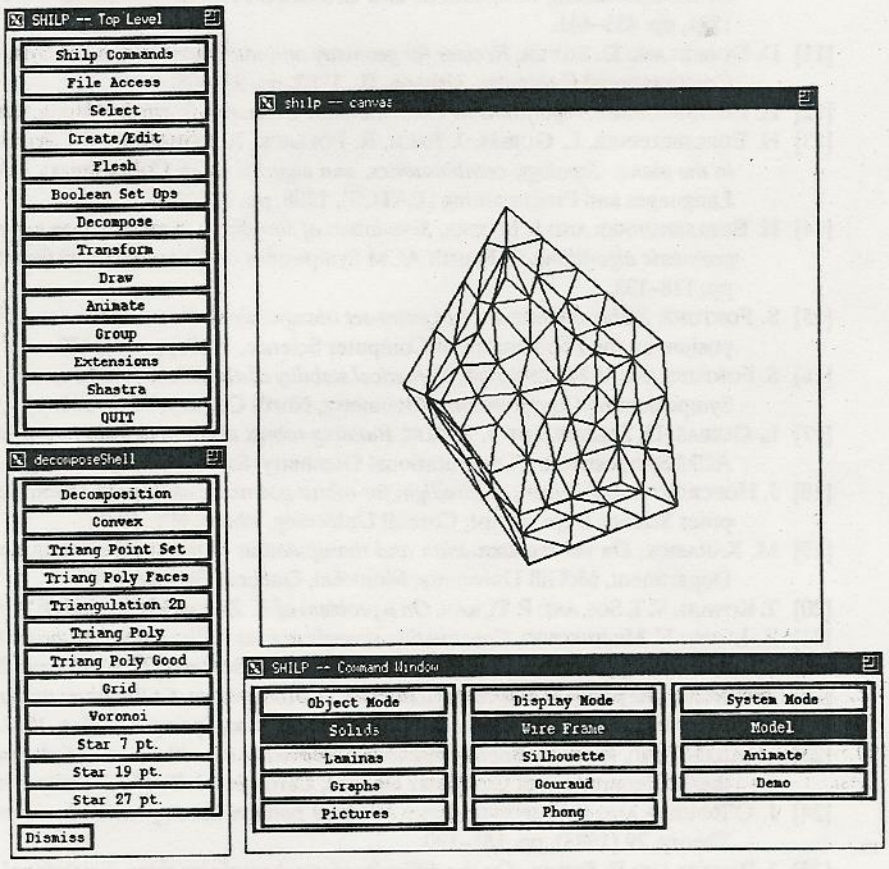

FIG. 14. *An example of a triangulation of a convex polyhedron.*

Acknowledgment. We thank Jack Snoeyink and three anonymous referees for their astute comments and suggestions.

REFERENCES

[1] V. ANUPAM, C. BAJAJ, T. DEY, I. IHM, AND S. KLINKNER, *The shilp modeling and display toolkit*, Tech. Report CSD-TR-988, Computer Science Department, Purdue University, West Lafayette, IN, 1989.

[2] M. A. ARMSTRONG, *Basic Topology*, McGraw-Hill, London, 1979.

[3] C. BAJAJ, *Geometric Modeling with Algebraic Surfaces, The Mathematics of Surfaces III*, Oxford University Press, 1988.

[4] C. BAJAJ AND T. DEY, *Polygon nesting and robustness*, Inform. Process. Lett., 35 (1990), pp. 23–32.

[5] B. CHAZELLE, *Computational geometry and convexity*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[6] ———, *Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm*, SIAM J. Comput., 13 (1984), pp. 488–507.

[7] B. CHAZELLE AND L. PALIOS, *Triangulating a non-convex polytope*, Discrete & Comput. Geom., 5 (1990), pp. 505–526.

[8] T. K. DEY, *Decompositions of polyhedra in three dimensions*, Ph.D. thesis, Department of Computer Science, Purdue University, West Lafayette, IN, 1991.

[9] ———, *Triangulation and CSG representation of polyhedra with arbitrary genus*, in Proc. 7th. ACM Symposium on Computational Geometry, 1991, pp. 364–372.

[10] T. K. DEY, C. BAJAJ, AND K. SUGIHARA, *On good triangulations in three dimensions*, in Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications (ACM/SIGGRAPH), Austin, TX, 1991, pp. 431–441.

[11] D. DOBKIN AND D. SILVER, *Recipes for geometry and numerical analysis*, in Fourth ACM Symposium on Computational Geometry, Urbana, IL, 1988, pp. 93–105.

[12] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, Heidelberg, 1987.

[13] 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 Internat. Colloquium on Automata, Languages and Programming (EATCS), 1988, pp. 214–229.

[14] 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, Urbana, IL, 1988, pp. 118–133.

[15] S. FORTUNE, *Stable maintenance of point-set triangulations in two dimensions*, in Proc. 30th IEEE Symposium on the Foundations of Computer Science, 1989, pp. 494–499.

[16] S. FORTUNE AND V. MILENKOVIC, *Numerical stability of algorithms for line arrangements*, in Seventh ACM Symposium on Computational Geometry, North Conway, NH, 1991, pp. 93–105.

[17] L. GUIBAS, D. SALESIN, AND J. STOLFI, *Building robust algorithms from imprecise computations*, in Fifth ACM Symposium on Computational Geometry, Saarbruchen, West Germany, 1989, pp. 208–217.

[18] J. HOPCROFT AND P. KAHN, *A paradigm for robust geometric algorithms*, Tech. Report TR 89-1044, Computer Science Department, Cornell University, Ithaca, NY, 1989.

[19] M. KARASICK, *On the representation and manipulation of rigid solids*, Ph.D. thesis, Computer Science Department, McGill University, Montréal, Québec, Canada, 1988.

[20] T. KOVARI, V. T. SOS, AND P. TURAN, *On a problem of K. Zarankiewicz*, Colloq. Math., 3 (1954), pp. 50–57.

[21] Z. LI AND V. MILENKOVIC, *Constructing strongly convex hulls using exact or rounded arithmetic*, in Sixth ACM Symposium on Computational Geometry, Berkeley, CA, 1990, pp. 235–242.

[22] A. LINGAS, *The power of non-rectilinear holes*, in 9th Internat. Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Springer-Verlag, 1982, pp. 369–383.

[23] V. MILENKOVIC, *Verifiable implementations of geometric algorithms using finite precision arithmetic*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[24] J. O'ROURKE AND K. SUPOWIT, *Some NP-hard polygon decomposition problems*, IEEE Trans. Inform. Theory, 29 (1983), pp. 181–190.

[25] J. RUPERT AND R. SEIDEL, *On the difficulty of tetrahedralizing three-dimensional nonconvex polyhedra*, in Fifth ACM Annual Symposium on Computational Geometry, 1989, pp. 380–392.

[26] M. SEGAL AND C. SEQUIN, *Consistent calculations for solid modeling*, in First ACM Symposium on Computational Geometry, Baltimore, MD, 1985, pp. 29–38.

[27] 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, Quebec, Canada, 1989.

[28] ———, *A solid modeling system free from topological inconsistency*, Research Memorandum RMI 89-3, Department of Mathematical Engineering and Instrumentation Physics, Tokyo University, Tokyo, Japan, 1989.

[29] C. YAP, *A geometric consistency theorem for a symbolic perturbation theorem*, in Fourth ACM Symposium on Computational Geometry, Urbana, IL, 1988, pp. 134–142.