1988

Convex Decompositions of Simple Polyhedra

Chanderjit Bajaj

Tamal K. Dey

Report Number:
88-833

CONVEX DECOMPOSITIONS OF SIMPLE POLYHEDRA

Chandrajit Bajaj
Tamal K. Dey

CSD TR-833
December 1988

CONVEX DECOMPOSITIONS OF SIMPLE POLYHEDRA*

Chanderjit Bajaj
Tamal K. Dey

Computer Sciences Department
Purdue University
Technical Report CSD-TR-833
CAPO Report CER-87-44
December, 1988

Abstract

Chazelle in [3] established a worst-case, quadratic lower bound on the complexity of the problem of partitioning a polyhedron into disjoint convex polyhedra and gave an algorithm that procudes a worst-case, optimal $O(N^2)$ number of convex polyhedra. His algorithm runs in $O(nN^3)$ time and $O(nN^2)$ space, where n is the number of edges of the polyhedron and N is the number of *notches* or reflex edges of the polyhedron. We present an algorithm which produces the same number of convex polyhedra as [3], but runs in only $O((nN + N^3 \log n)\log N)$ time and $O(nN + N^3)$ space.

* Supported in part by NSF Grant MIP 85-21356, ARO Contract DAAG29-85-C0018 under Cornell MSI and ONR contract N00014-88-K-0402.

1 Introduction

Motivation: 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 [3]. Convex decompositions lead to efficient algorithms, for example, in geometric point location and intersection detection, see [4]. Our motivation stems from the use of geometric models in a physical simulation system being developed at Purdue [1]. Specifically, a disjoint convex decomposition of simple polyhedra allows for more efficient algorithms in collision detection, in convolution generation for planning of motion under contact, and in the computation of volumetric properties.

Problem Statement: Given a simple polyhedron P in three dimensions, decompose it into pairwise disjoint convex polyhedra, whose union is exactly P.

Related Work: The problem of partitioning a polyhedron into a *minimum* number of convex parts is known to be NP-hard [6]. For the above problem however, Chazelle established a worst-case, $O(n^2)$ lower bound on the complexity of the problem and gave an algorithm that produces a worst-case, optimal number, $O(N^2)$ of convex polyhedra in $O(nN^3)$ time and $O(nN^2)$ space, where n is the number of edges of the polyhedron P and N is the number of *notches* or reflex edges of P.

Results: In this paper we present an algorithm which produces the same number of convex polyhedra as [3], but runs in only $O((nN + N^3 \log n)\log N)$ time and $O(n\,N + N^3)$ space.

Algorithm Synopsis: Let P be a simple polyhedron, possibly with holes, and having n vertices : $v_1, v_2, ..., v_n$, p edges : $e_1, e_2, ..., e_p$ and q faces : $f_1, f_2, ..., f_q$. Assume that every edge of P has exactly two incident faces. An edge g of P is a *notch* if the inner-angle γ between the two incident faces of g, is greater than 180°. Nonconvexity in P, is a result of the presence of these notches in the polyhedron. Chazelle's algorithm proceeds in removing all notches of P, by repeatedly cutting and splitting P with planes containing the notches, finally producing convex polyhedra P_i, with $\bigcup_i P_i = P$. If edge g is a notch, with f_g^-, f_g^+ as its incident faces, then a plane T which contains the notch g and subtends an inner-angle greater than $\gamma - 180°$ with both f_g^- and f_g^+, is a valid notch plane for g. The chosen plane T is also called the *notch plane* of g. Clearly there are an *infinity* of choices for T.

The notch plane T of g may intersect other notches in P, thereby producing subnotches. Subnotches are thus split reflex edges of the original polyhedron P, and which are still reflex for the current polyhedra produced from the splitting of P. At a generic instant of time, after cutting and splitting P with certain *notch planes*, there would be several polyhedra containing notches and subnotches. The process is repeated until all the notches and subnotches are removed, thereby yielding convex polyhedra.

By choosing to cut all the subnotches of a single notch with the same notch plane, [3] guarantees an $O(N^2)$ number of convex polyhedra. We also follow this strategy of selecting notch planes, yielding the same number of covex polyhedra as [3]. However, we present a more efficient procedure for the actual polyhedron cutting and splitting operations, producing a vastly improved time bound.

2 Preliminaries

We first define some of the terms and notation to be used later. The intersection between T and P is in general, a set of simple polygons, possibly with holes. We call this set of polygons as the *cutset* of T. See Figure 2.1. If G is a simple polygon with vertices $v_1, v_2, ..., v_n$ in clockwise order, a vertex v_i is a notch of G if the inner angle between the edge (v_{i-1}, v_i) and (v_i, v_{i+1}) is $> 180°$. Between any two consecutive notches v_i, v_j in the clockwise order, the sequence of vertices $(v_i, v_{i+1}, ..., v_j)$ is called a *convex polygonal-line*. Each polygonal-line can be partitioned into *convex-chains*, which are maximal pieces of a polygonal-line, with the property that its vertices form a convex polygon. Each convex-chain can be further partitioned into at most 3 x-monotone maximal pieces called *subchains*, i.e., vertices of a subchain have x-coordinates in either strictly increasing or decreasing order. See Figure 2.2.

For the next section we also need the following Lemmas, from Chazelle's thesis [2].

Lemma 1: Let G be a simple polygon with N_G notches, then number of convex-chains C_G in G is bounded as $C_G \leq 2(1 + N_G)$.

Proof: See ([2], page 22, Theorem 3). ♣

Lemma 2: Let G be a simple polygon with N_G notches, then number of subchains C_{SG} in G is bounded as $C_{SG} \leq 6(1 + N_G)$.

Proof: It is easy to see that there are atmost 3 subchains per convex-chain. This fact together with Lemma 1 yields the bound. ♣

Lemma 3: Let G be a simple polygon with N_G notches. No line can intersect G in more than $2N_G$ segments.

Proof: See ([2], page 121, Lemma 18). ♣

When N_G is zero, one line can intersect G in at most one segment. We therefore modify the above Lemma to be

Lemma 4: Let G be a simple polygon with N_G notches. No line can intersect G in more than $max(1, 2N_G)$ segments or $max(2, 2N_G + 1)$ points.

3 Convex Decomposition

We represent the polyhedron with a compact data structure described below. See also Figure 3.1. Each edge, between two vertices v_i, v_{i+1} in the polyhedron, is represented by two directed edges $e_{i_1} = (v_i, v_{i+1})$ and $e_{i_2} = (v_{i+1}, v_i)$. We assume for the present that each edge of the polyhedron is adjacent to two faces. This simplification may be dropped with slight modifications of our algorithm. We call v_i and v_{i+1} as the start and end node respectively of the edge (v_i, v_{i+1}).

Data Structure

1. *vertex-edge-face-list*: Each vertex node is connected to the set of vertex nodes adjacent to it. Each edge (v_i, v_{i+1}) (represented by the link between two vertices) has an attribute, (the faces associated with that edge) represented by a linear list called the *face-list*. Each entry for a face in the face-list of (v_i, v_{i+1}) is a pointer to a node representing v_i in a structure called the *face-structure* corresponding to that face.

2. *face-structure*: Each subchain c_i in the face is represented by a 2-3 tree T_{c_i} with leaves representing the vertices. The leaves representing the vertices are sorted in the clockwise direction around the face. Leaves are connected by doubly linked lists. Each leaf has a pointer to the face list associated with the edge starting from that vertex. Each leaf also has a pointer to the root of its tree.

Note that this data structure allows us to go in $O(1)$ time, from any vertex in the face-structure to an edge which starts from that vertex and contained in the face. Once we reach the edge we can reach the other face associated with that edge, also in $O(1)$ time.

The Algorithm

Since decomposing a polyhedron P with N notches consists of a sequence of intersections with notch planes, we first describe the method of cutting a polyhedron P by the *notch plane* T of a notch g. Recall that the intersection of P and the notch plane T is a set of polygons S called T's cutset. We determine the unique polygon Q from S called the *polygon-cut*, supporting the notch g. After determining the polygon-cut Q, we need to split P along the cut Q. Actually splitting P along the cut Q instead of cutset S, is sufficient to remove the notch g through P. Note that because of this, P may not get separated into two different pieces after the split. See also Figure 3.2.

- *Step I*: Determine Q. This calls for the following steps
  - *Step I(a)*: Determine the outer boundary of Q.
  - *Step I(b)*: Determine the inner boundary(s) of Q (if any).

- *Step II*: Separate P. While describing the algorithm we assume P is separated into two pieces by cutset Q. We later describe how we handle the case where P is merely spliced by Q instead of getting separated into two pieces.

Details of Step I

We first describe the method of determining any boundary of the set of polygons S. Suppose we have an initial point a_1 on the boundary B. We can determine other points on B in the following way. Let a_1 be on the edge u_1 of the face f_i. Let $a_2, ..., a_k$ be other intersection points of f_i with T on the edges $u_2, u_2, ..., u_k$. We need to determine $a_2, ..., a_k$ which will be on B. See Figure 3.3. Since intersection of f_i and T is a line L, in general, determining $a_2, ..., a_k$ requires nothing more than determining the intersection points of L with the simple polygon representing f_i.

Since we store the subchains of faces in a 2-3 tree, the intersection point in each subchain c_j can be determined in $O(log\ p_i)$ time where p_i is the number of vertices in the subchain c_i. By Lemma 4, $k \leq max(2, 2N_i + 1)$ where N_i is the number of notches in f_i. By Lemma 2, the number of subchains h_i in f_i is bounded by $6(1 + N_i)$. So, determination of $a_2, ..., a_k$ takes at most $\sum_{j=1}^{h_i} log\ p_j$, where $h_i \leq 6(1 + N_i)$. We sort a_i's on the line of intersection L. This takes $O(max(1, N_i\ log\ N_i)) = O(1 + N_i\ log\ N_i)$ time since $k \leq max(2, 2N_i + 1)$. We join a_i and a_2 and keep $a_3, a_4, ..., a_k$ in a list associated with f_i for future use as described below. We examine u_2 and get the face f_{i+1} associated with u_2 other than f_i. In our data structure we can find f_{i+1} in constant time. Now, in face f_{i+1}, all the intersection points might have been determined earlier. We check the list of boundary points (intersection points) associated with f_{i+1}. If the list is empty, we follow the above procedure to determine it, otherwise we join a_2 with a'_3 in the list $a'_2, a'_3, ..., a'_k$ associated with f_{i+1}. We delete a'_2, a'_3 from this list. Note a'_2 and a_2 are the same point. See for e.g., Figure 3.4. Now we proceed from a'_3 and go on following the above procedure, until we reach the initial point a_1, on the boundary B. Obviously, the time taken to determine all the points on B is

$$O(\sum_{i=1}^{h} log\ p_i + \sum_{i=1}^{r}(N_i\ log\ N_i + 1)),$$

where h is the total number of subchains in all the faces intersected by T and r is the number of such faces. Note p_i is the number of vertices in the i^{th} such subchain.

Now, we describe how to determine the outer and inner boundary(s) of Q.

Step I(a): The notch g will be on the outer boundary of Q. So, we can take any vertex of g as the initial point to start with determining the outer boundary of Q by the above method.

Step I(b): Let I_i be any inner boundary of Q. I_i itself constitutes a simple polygon. Polygon I_i will have at least one (actually at least three) vertex, which is not a notch. Since I_i is the inner boundary of Q, the vertices which are not notches of polygon I_i are notches of Q. Definitely,notches of Q lies on notches of P. This guarantees us that all inner boundaries of Q will have a point which

is the intersection point of T with a notch of P. So, we determine the set W of intersection points of all notches of P with T. We take one such point as the initial point and determine the corresponding boundary and delete all the intersection points from W, which appear on the boundary. We determine all such boundaries until W becomes empty. Maintaining W as a sorted list, we can determine whether a point belongs to W or not in $O(log\ N)$ time since $|W| = O(N)$. If there are p' points on the boundaries of cutset S this membership checking takes $O(p'\ log\ N)$ time. Sorting of W takes $O(N\ log\ N)$ time. Hence, this adds at most $O(p'\ log\ N + N\ log\ N)$ extra time, to boundary determination. After determining all such boundaries we can determine the inner boundaries of Q in $O(p' + N\ log\ N\ log\ p')$ time using the plane sweep technique, see for e.g. [4]. Combining the complexity of Step I(a) and I(b), we conclude that the outer and inner boundary(s) of Q can be determined in

$$O(\sum_{i=1}^{h} log\ p_i + N\ log\ N + p'\ log\ N + N\ log\ N\ log\ p')$$

since

$$\sum_{i=1}^{r} (N_i\ log\ N_i + 1) = O(N\ log\ N + p'),$$

r is the number of faces intersected by T. Obviously, r is $O(p')$ since each such face contributes at least one point on the boundary.

Details of Step II

Separation of P corresponding to the polygon-cut Q is carried out by updating the vertex-edge-face-list and face-structure of P. Note that since each of these is dependent on the other, an update performed on one affects the other indirectly and consequently allows us to avoid checking all edges of P. Let Q separate P in P_1 and P_2. Let the vertex x be to that side of Q which will be in P_1. We can denote P_1 by the vertex x. Similarly, we can denote P_2 by a vertex y which lies on the other side of Q. We can determine whether any vertex (or face) belongs to P_1 (or P_2) by simply checking whether it is to the same side of Q as x (or y). The face represented by Q will be present in both parts P_1 and P_2. See Figure 3.5.

From Q, we create the face-structure of Q by creating the tree structures for the subchains in Q. We create two such structures f_{Q_1} (for P_1) and f_{Q_2} for P_2. From Q, we also create two vertex-edge-face-lists V_{Q_1}, V_{Q_2} corresponding to the vertices of Q in the following way. We traverse the vertices of Q one after another as they appear on the boundaries of Q and for each edge (a_1, a_2) of Q, we create two edges $u_1 = (a_1^{Q_1}, a_2^{Q_1})$ and $u_2 = (a_1^{Q_2}, a_2^{Q_2})$. u_1 is put in V_{Q_1} and u_2 in V_{Q_2}. Since, with each such edge u_1 (resp. u_2), the face f_{Q_1} (resp. f_{Q_2}) has to be associated, we set one face-list pointer of u_1 (resp. u_2) to point to a leaf node in f_{Q_1} (resp. f_{Q_2}) which corresponds to a_1 in Q. We also set a pointer from that leaf node to the face-list of u_1 (resp. u_2). The other pointer in the face-list is set later.

Now, we split the faces of P which were intersected by the notch plane T. The edges of Q lies on these faces also. Suppose f_i is such a face which is to be split at $a_1, a_2, ..., a_k$ which are on the edges $u_1, u_2, ..., u_k$. The splitting of f_i consists of splitting the trees corresponding to the subchains in which $(a_1, a_2, ..., a_k)$ lies and inserting $a_1, a_2, ..., a_k$ in proper trees. Note that each of $a_1, a_2, ..., a_k$ has to be inserted in two trees, since each edge (a_i, a_j) will be present in two new faces created by splitting f_i. Let a_1 be inserted in f_{iQ_1} and f_{iQ2}. For a_1 in f_{iQ_1}, we actually insert a pointer to the face-list of $(a_1^{Q_1}, a_2^{Q_1})$ in V_{Q_1}. For a_1 in f_{iQ_2} we insert a pointer to the face-list of $(a_1^{Q_2}, a_2^{Q_2})$ in V_{Q_2}. One face-list pointer of the edge $(a_1^{Q_1}, a_2^{Q_1})$ in V_{Q_1} is set to point to a_1 in f_{iQ_1} Similarly, one pointer in the face-list of $(a_1^{Q_2}, a_2^{Q_2})$ in V_{Q_2} is set to point to a_1 in f_{iQ_2}. The effect of this is to set the face-list pointers of the edges in V_{Q_1} and V_{Q_2}.

We omit further details of this splitting to avoid the complications. Each face-splitting will not take more than $O(\sum_{j=1}^{h_i} log\ p_j)$ time where h_i is the number of subchains in f_i and p_j is the number of vertices in j-th subchain.

Now an existing vertex-edge-face-list of P has to be modified to incorporate V_{Q_1} and V_{Q_2}. Again, we traverse the vertices in Q one after another and for each vertex a_1 we do the following. Suppose a_1 is on the edge of u_1 of face f_1 of P. Let two end points of u_1 be m, n. We assume m will be in P_1 and n in P_2. We link m and $a_1^{Q_1}$ both ways. Similarly, we link n and $a_1^{Q_2}$ both ways. Now, we have to associate face-list pointers with the edges $(m, a_1^{Q_1})$, $(a_1^{Q_1}, m)$ and $(n, a_1^{Q_2})$, $(a_1^{Q_2}, n)$. By the previous operations, the leaf corresponding to m in face structure of f_i has been properly placed in some new faces. We need not change the face-list pointers which were associated with (m, n). We keep it associated with $(m, a_1^{Q_1})$. Similarly, we keep pointers with $(n, a_1^{Q_2})$ same as the pointers which were associated with (n, m). Now, it is easy to set the face-list pointers of $(a_1^{Q_1}, m)$ and $(a_1^{Q_2}, n)$ since a_1 is adjacent to $m(resp. n)$ in the faces pointed to by the face-list pointers of $(m, a_1^{Q_1})$ (resp. $(n, a_1^{Q_2})$). This completes the separation process.

Note that we visit only the vertices on the boundaries of Q and for each vertex we spend constant time for setting relevant pointers and additional time for splitting and insertion operations in the trees corresponding to subchains in the faces. The latter is logarithmic in the number of vertices contained in the subchain. Hence, the separation takes

$$O\left(p' + \sum_{i=1}^{h} log\ p_i\right)$$

time where p' is the number of vertices in S and h is the total number of subchains contained in the faces intersected by the notch plane T. p_i is the number of vertices in the i-th such subchain. Note that p' is also the number of edges of P intersected by T. This yields

Lemma 5. A polyhedron P of genus 0, having N notches can be partitioned with a cut in

$$O\left(\sum_{i=1}^{h} log\ p_i + N\ log\ N\ log\ p' + p'\ log\ N\right)$$

time and in $O(p)$ storage, where p is the number of edges of P, p' being the number of edges of P intersected by the plane T supporting the cut, h being the total number of subchains in all the faces intersected by T, p_i being the number of vertices in the i-th such subchain.

We can generalize the above result for polyhedron of arbitrary genus. For this, as described in [3], we have to handle the situation when the cut does not separate P into two pieces, but only creates two new faces supporting the cut at the same geometric location. In this case we can do a depth-first search in the vertex-edge-face list to determine whether the cut separates P into two pieces or not. But as described later, in the sequence of cuts which removes all notches of P, we actually do not check whether a cut breaks a polyhedron into two pieces and only when we remove all notches from P, do we resolve this ambiguity as described later.

Lemma 6. Let $P_1, P_2, ..., P_k$ be the polyhedra in the current decomposition which contians a subnotch of g to be resolved, and let v_i' be the total number of vertices in the cutset resulted from intersection of P_i with a notch plane T, then since any notch can be intersected by at most $(N-1)$ notches, we have $k = O(N)$ and $v' = \sum_{i=1}^{k} v_i' = O(n + N^2)$, where v' is the total number of vertices on all the cutsets of $P_1, P_2, ..., P_k$.

Proof: Let p_i' be the number of edges of P_i intersected by T. Since each vertex on a cutset is the intersection point between one edge of P_i and T, we have $v_i' = p_i'$. Let the notch plane T cut the face f_i in s_i segments. Then $p_i' = \sum_{i=1}^{r_i} s_i$ where r_i is the total number of faces of polyhedra P_i cut by T. By Lemma (3), $s_i \leq 2N_{f_i} + 1$ where N_{f_i} is the number of notches of face f_i. So,

$$\sum_{j=1}^{r_i} s_j = \sum_{j=1}^{r_i} 2N_{f_j} + \sum_{j=1}^{r_i} 1 = 2N_i + r_i,$$

where N_i is the number of notches in P_i, hence

$$\sum_{i=1}^{k} p_i' = \sum_{i=1}^{k} 2N_i + \sum_{i=1}^{k} r_i = O(N^2) + F$$

where F is the total number of faces intersected by the notch plane T, summed over all the polyhedra.

Now we prove $F = O(n + N^2)$. We divide the contribution to F into two classes, viz., C_1:= the faces which do not lie on the faces of the original polyhedron, and C_2:= the faces which lie on the faces of the original polyhedron. The faces in class C_2 are pieces of the faces of the original polyhedron. Now, since each cut in a polyhedron P_j, generates at most two faces, which do not lie on the faces of P_j, the number of C_1 type faces can increase by at most one in each of the resulting split pieces. Hence, each piece has at most $O(N)$ faces which are in C_1. Finally, the total number of faces F_1 in C_1 is given by $F_1 = \sum_{i=1}^{k} cN = O(N^2)$, since $K = O(N)$.

Now, let p_{F_2} be the total number of edges on the cutset which are created by the intersection of notch plane with the faces of class C_2. Obviously, these edges lie on the surface of the original

polyhedron. The number of faces F_2 in class C_2 is obviously bounded by p_{F_2}, i.e., $F_2 = O(p_{F_2})$.

Actually, these edges are the parts of the edges of the cutset S_0 generated by the intersection of T with the original polyhedron. Parts of the edges of S_0 are generated by other cuts. Obviously, the number of edges on S_0 is $O(n)$. There are at most N planes or equivalently N lines, which split boundaries of S_0. If a boundary B_i has N_i notches, by Lemma 4, each line intersects at most $2 \cdot (2N_i + 1)$ edges of B_i and thus generates $2 \cdot 2\,(2N_i + 1)$ new parts of edges. If there are b number of boundaries in S_0, N lines produces at most $\sum_i^b 4\,(2N_i + 1)N = O(N^2)$ new parts of the edges of S_0. So, $p_{F_2} = O(N^2 + n)$, hence $F_2 = O(N^2 + n)$, hence $F_1 + F_2 = F = O(N^2 + n)$, and

$$v' = \sum_{i=1}^{k} p_i' = O(N^2) + O(N^2 + n) = O(N^2 + n) \quad \clubsuit$$

Lemma 7: The total number of edges in the final decomposition of P with N notches is $O(n\,N + N^3)$.

Proof: Total number of edges in the final decomposition consists of newly generated edges by the polygon-cuts, and the edges of P which are not intersected by any notch plane. Now since the total number vertices in all the cutsets of a partial decomposition is $O(n + N^2)$ as proved in Lemma 6, the total number of newly generated edges by each notch plane is $O(n + N^2)$. Thus N notch planes generate $O(n\,N + N^3)$ new edges. Hence, the total number of edges in the final decomposition is $O(n\,N + N^3 + n) = O(n\,N + N^3)$ ♣

Theorem 8: A polyhedron P of arbitrary genus having N notches and n edges can be decomposed into $O(N^2)$ convex parts in $O((nN + N^3\ log\ n)\ log\ N)$ time and in $O(n\,N + N^3)$ space.

Proof: Decomposition of a polyhedron consists of a sequence of cuts through the notches of P. We can assign a notch plane for each notch in P in $O(N)$ preprocessing time. Now, we remove each notch by removing all of its subnotches by the cutting plane assigned to this notch. Each planar cut to remove a subnotch in a polyhedra, can be carried out by the method described above. This produces $O(N^2)$ convex pieces at the end. Now a single cut may not produce separate pieces for a polyhedron in the partial decomposition. We do not pay attention to this until we reach the end. Recall that we recognize a polyhedron by one vertex in that polyhedron. We can keep all these vertices in a separate sorted list L_P. At the end, we take one vertex from this list and do a depth first search in the vertex-edge-face-list to determine all the vertices in that polyhedron and remove all vertices from L_P which are encountered during this depth-first search. We do this until L_P becomes empty. This has the effect of recognizing all the new polyhedra without any ambiguity. Certainly, this takes $O(p\ log\ D)$ time where p is the total number of edges in the final decomposition and D is the total number of convex parts in the final decomposition. Since $D = O(N^2)$, the above method takes $O(p\ log\ N)$ time.

Time analysis: At a generic instance of the algorithm, let $P_1, P_2, ..., P_k$ be the k distinct (non-convex) polyhedra in the current decomposition, which contains the subnotches of a notch g which we are going to remove. Let p_i' be the number of edges in P_i intersected by the notch plane. Using Lemma 5, we can say the time $\Im$ to remove the notch g is given by

$$\Im = O\left(\sum_{i=1}^{k}(\sum_{m=1}^{h_i} log\ p_m^{(i)} + N_i\ log\ N_i\ log\ p_i' + p_i'\ log\ N)\right)$$

where h_i is the total number of subchains in the faces intersected by the notch plane in P_i and $p_m^{(i)}$ is the number of vertices in the m^{th} such subchain in P_i.

We can write $\sum_{i=1}^{k}\sum_{m=1}^{h_i} log\ p_m^{(i)} = O(\sum_{i=1}^{h} log\ p_i)$, where h is the total number of subchains in the faces of $P_1, P_2, ..., P_k$ which are intersected by the notch plane. Now from Lemma 2, we know any face with N_P notches can not have more than $6(1 + N_P)$ subchains. This immediately gives $h = O(\sum_{i=1}^{k} r_i + \sum_{i=1}^{k} N_i)$, where r_i is the number of faces intersected by notch plane in P_i. Since $r_i = O(p_i')$ we have $h = O(\sum_{i=1}^{k}(p_i' + N_i))$.

Let p be the total number of edges in all the polyhedron $P_1, P_2, ..., P_k$ and v' be the total number of vertices on all cutsets. Certainly $\sum_{i=1}^{k} p_i' = v'$. Since any notch can be cut by at most $(N + 1)$ notch planes $\sum_{i=1}^{k} N_i = O(N^2)$. This gives $h = O(v' + N^2)$. Now since the *log* function is a monotonic increasing and concave, and $\sum_{i=1}^{k} p_i = O(p)$ we can write $\sum_{i=1}^{h} log\ p_i = O(h\ log\ \frac{p}{h})$. Furthermore, since $h \leq p$, $h\ log\ \frac{p}{h}$ becomes maximum when h becomes maximum and $h = O(v' + N^2)$, we can see that $O(h\ log\ \frac{p}{h}) = O((v' + N^2)\ log(\frac{p}{v' + N^2}))$ and $\sum_{i=1}^{k} N_i\ log\ N_i\ log\ p_i' = O(N^2\ log\ N\ log\ v')$.

All this yields

$$\Im = O\left((v' + N^2)\ log\left(\frac{p}{v' + N^2}\right) + N^2\ log\ N\ log\ v' + v'\ log\ N\right)$$

In Lemma 6, we prove that $v' = O(n + N^2)$ and in Lemma 7, we prove that the total number edges in the final decomposition of P is $(n\ N + N^3)$. This gives $p = O(nN + N^3)$ and

$$\begin{aligned}\Im &= O((n + N^2)\ log\left(\frac{nN + N^3}{n + N^2}\right) + N^2\ log\ N\ log\ n\ (n + N^2)) \\ &= O((n + N^2)\ log\ N + N^2\ log\ N\ log\ n) \\ &= O((n + N^2\ log\ n)\ log\ N)\end{aligned}$$

Since, we carry out removal of N notches, the total time complexity for polyhedron decomposition is $O((nN + N^3\ log\ n)\ log\ N)$.

Space Analysis: In Lemma 7, we prove that the total number of edges in the final decomposition of P is $O(n\ N + N^3)$. Since, p = $O(n\ N + N^3)$, the space complexity of polyhedron decomposition can be seen to be also $O(n\ N + N^3)$. ♣

4 Conclusion

We have presented an efficient algorithm which produces a disjoint convex decompositions of simple polyhedra. Our next goal is to achieve an implementation of this algorithm, together with a better understanding of the important, underlying robustness issues. For robust computations which always yield consistent boundary topologies, one needs to make specific topological decisions based on imprecise numerical data, [5], [7]. The methodology we adopt is to live with uncertainity. Namely, the choices that some evaluated quantity ϵ is negative, zero or positive, are equally likely. Decision points, where several choices may exist, are to be considered either "independent" or "dependent". At independent decision points, any choice may be made from the finite set of possibilities while the choice at dependent decision points ensures the invariant state of global consistency. This consistency, for now, is to be achieved by means of topological reasoning.

References

[1] Bajaj, C., Dyksen, W., Hoffmann, C., Houstis, E., Korb, T., and Rice, J, (1988), "Computing About Physical Objects", *Proc. of the 12th IMACS World Congress*, Paris, 642 - 645.

[2] Chazelle, B., (1980), "Computational Geometry and Convexity", *Ph.D. Thesis*, CMU-CS-80-150, Computer Science, Carnegie-Mellon University.

[3] 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.

[4] Edelsbrunner, H., (1987), "Algorithms in Combinatorial Geometry", Springer Verlag.

[5] Hoffmann, C., Hopcroft, J., and Karasick, M., (1988), "Robust Operations for Polyhedral Models", *Proc. of the Fourth ACM Symposium on Computational Geometry*, Urbana, Illinois, 106-118.

[6] O'Rourke, J., and Supowit, K., (1983), "Some NP-hard Polygon Decomposition Problems", *IEEE Trans. Inform. Theory*, 29, 181 - 190.

[7] Sugihara, K., and Iri, M., (1988), "Geometric Algorithms in Finite Precision Arithmetic", Research Memorandum RMI 88-10, Department of Mathematical Engineering and Instrumentation Physics, Tokyo University.

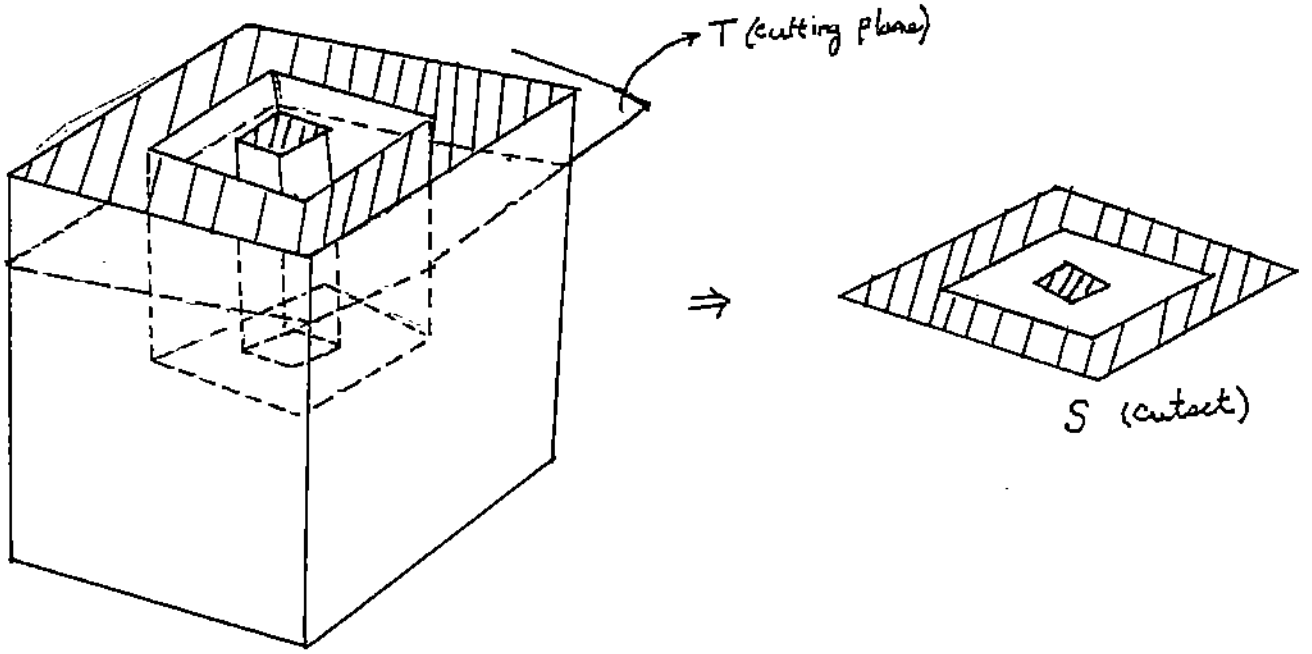

Fig 2.1 (a)

Fig 2.1 (b)

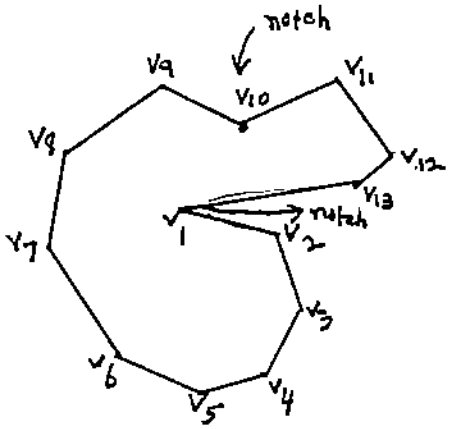

Fig. 2.2 (a)

$v_1, \ldots, v_{10}$ is a convex polygonal line
$v_{10}, \ldots, v_1$ is another convex polygonal line

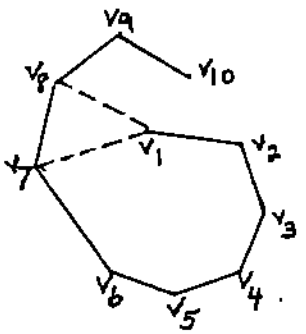

Fig. 2.2 (b)

$v_1, \ldots, v_7$ is a convex chain
$v_7, \ldots, v_{10}$ is another convex chain.

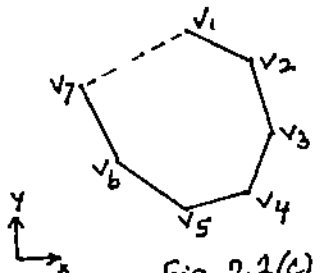

Fig. 2.2 (c)

$v_1, \ldots, v_3$ is a sub chain.
$v_-, \ldots, v_-$ is another sub chain.

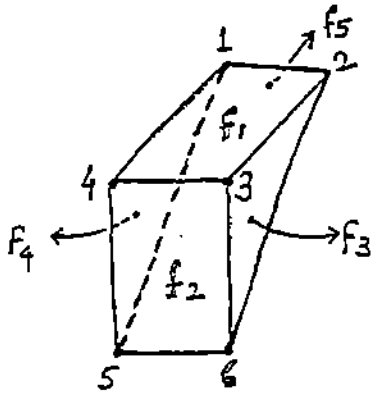

Fig 3.1(a)

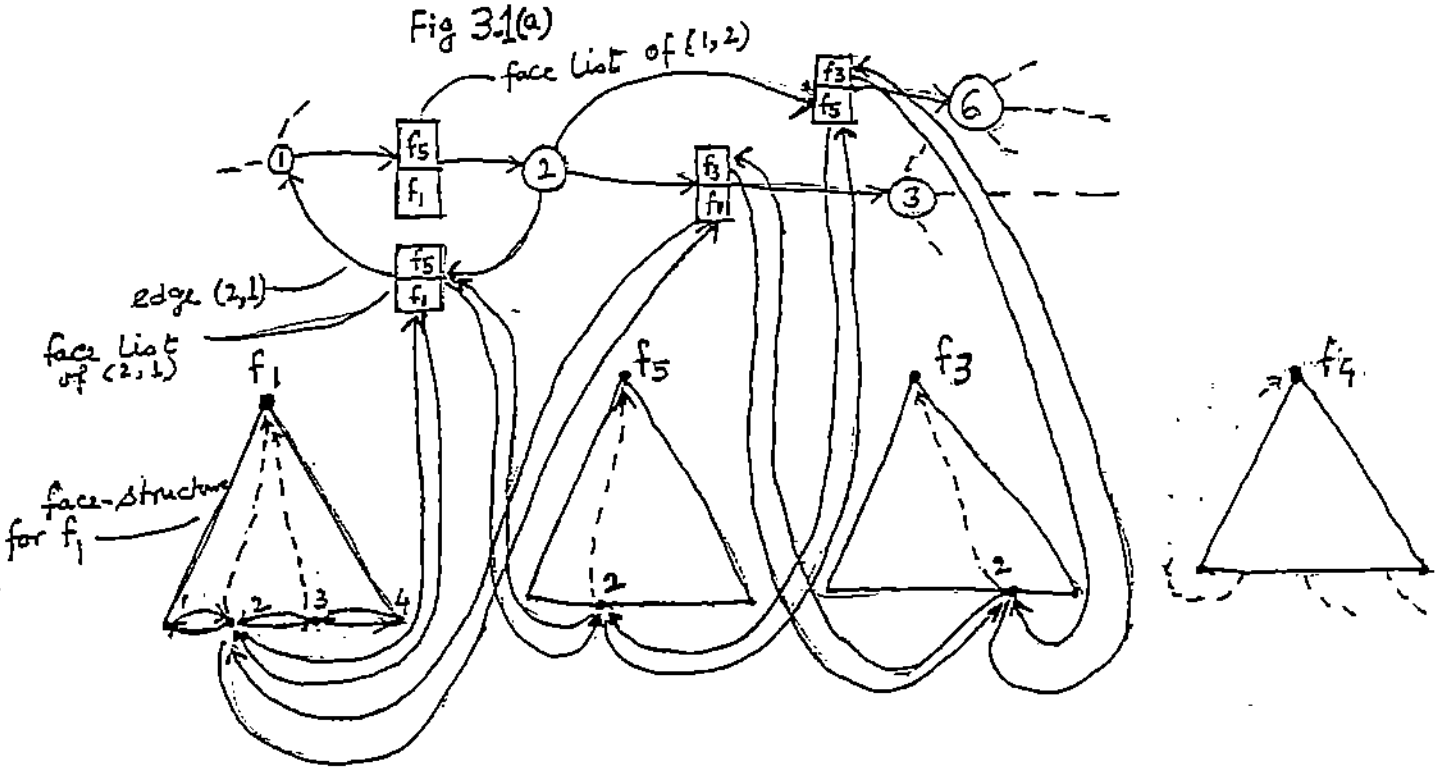

Fig 3.1 (b)

We show all the face list pointers for three edges from vertex 2.

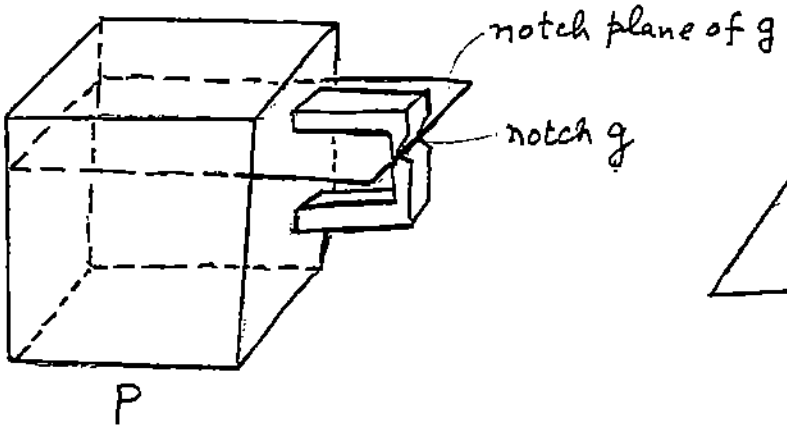

Fig. 3.2 (a)

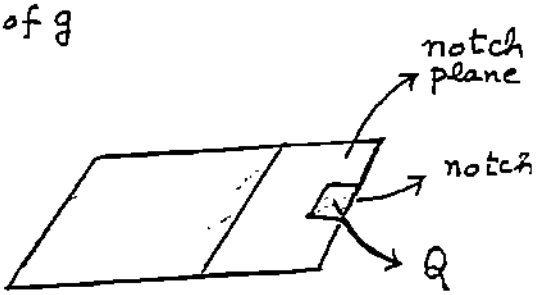

Fig. 3.2 (b)

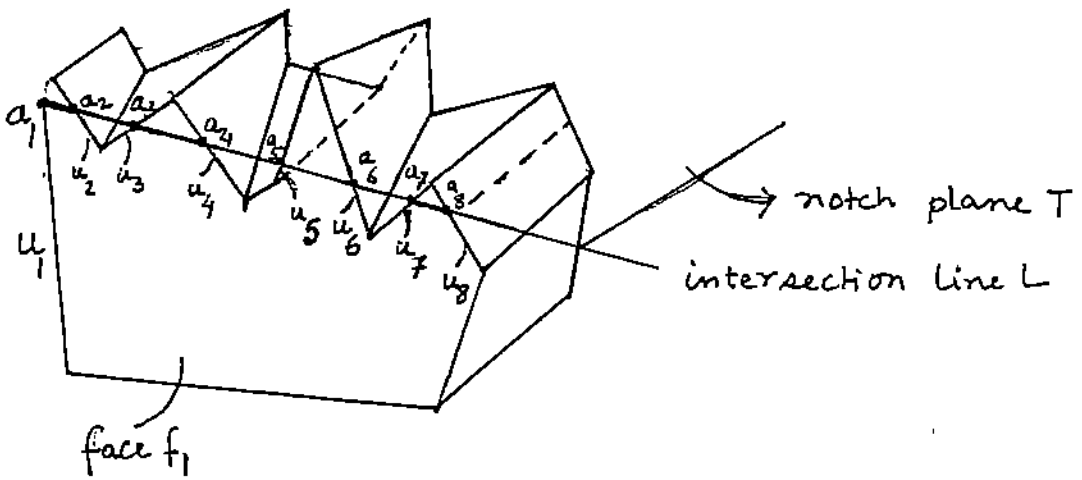

Fig 3.3

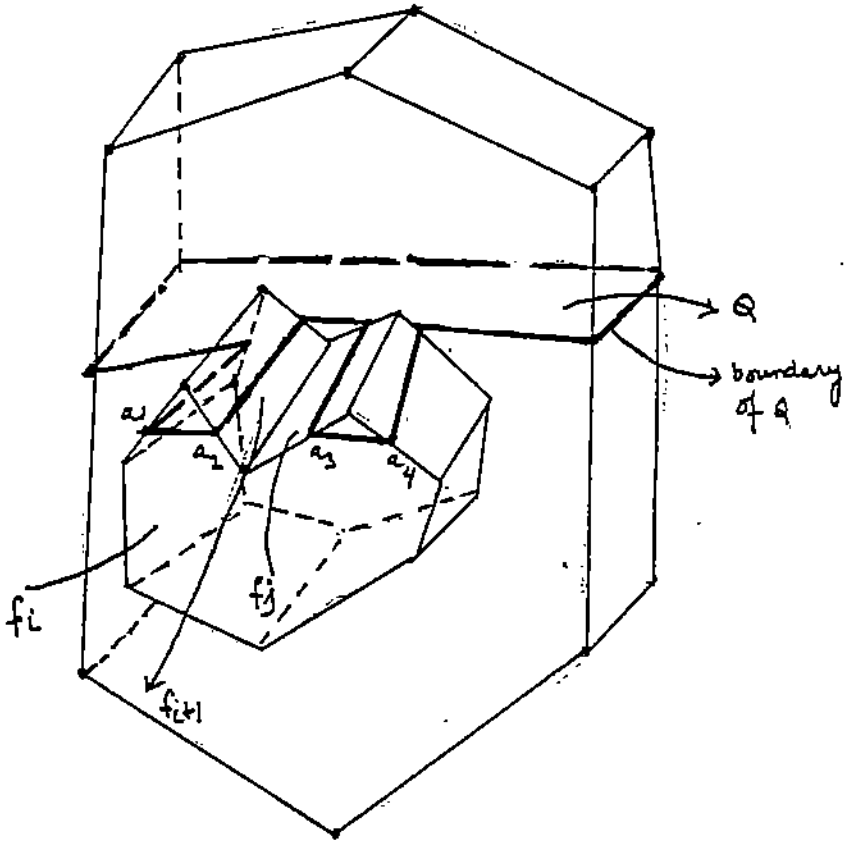

Fig 3.4

for face f_i
a_1, a_2, a_3, a_4 are determined.
When f_k is again encountered after f_j we utilize the list (a_3, a_4) of intersection points which were already determined

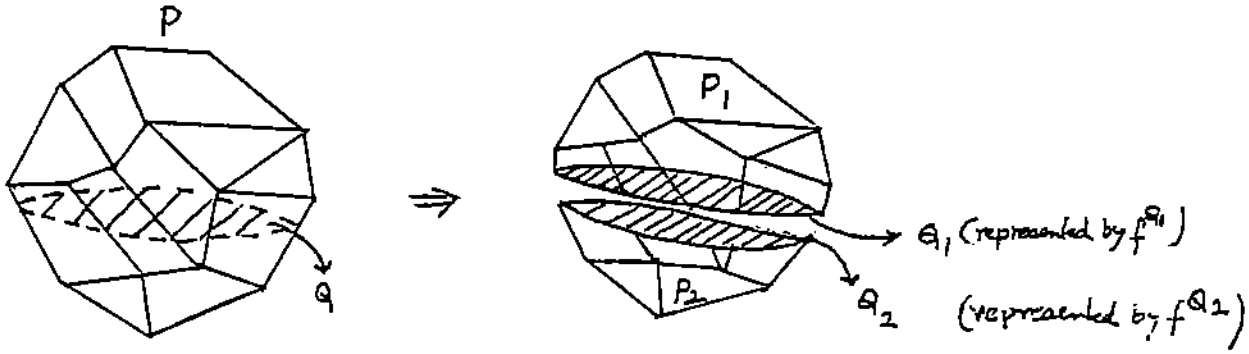

Fig. 3.5