

SPACE SWEEP SOLVES INTERSECTION OF  
TWO CONVEX POLYHEDRA ELEGANTLY

Stefan Hertel<sup>\*</sup>  
Kurt Mehlhorn<sup>\*</sup>  
Martti Mäntylä<sup>\*\*</sup>  
Jurg Nievergelt<sup>\*\*\*</sup>

Fachbereich 10  
Universität des Saarlandes  
6600 Saarbrücken  
West Germany

A 84/02  
March 1984

<sup>\*</sup> Fachbereich 10, Universität des Saarlandes,  
D-6600 Saarbrücken, West Germany

<sup>\*\*</sup> Lab. of Inf. Proc. Science, Helsinki University of Technology  
Otakaari 1, SF-02150, Espoo 15, Finland

<sup>\*\*\*</sup> Institut für Informatik, ETH-Zentrum,  
CH-8092, Zürich, Switzerland

# SPACE SWEEP SOLVES INTERSECTION OF TWO CONVEX

## POLYHEDRA ELEGANTLY

Stefan Hertel, Kurt Mehlhorn, Universität des Saarlandes, Saarbrücken

Martti Mäntylä, Helsinki University of Technology, Helsinki

Jürg Nievergelt, Eidgenössische Technische Hochschule, Zürich

Plane-sweep algorithms form a fairly general approach to two-dimensional problems of computational geometry. No corresponding three-dimensional space-sweep algorithms for geometric problems in 3-space are known, however. We derive concepts for such space-sweep algorithms that yield an elegant solution to the problem of solving any set operation (union, intersection, ...) of two convex polyhedra. Moreover, our solution matches the best known time bound of  $O(n \log n)$  where  $n$  is the combined number of corners of the two polyhedra.

Index terms: Computational geometry, sweep algorithms, intersection problems, convex polyhedra.

## I. INTRODUCTION

In recent years *plane-sweep algorithms* have become prominent in 2-dimensional computational geometry, beginning with the influential paper of Shamos and Hoey [SH 76]. Bieri and Nef [BN 82] trace the idea back to Hadwiger [Ha 55] who considered the so-called "Konvexring", the class of all finite unions of convex and compact subsets of  $\mathbb{R}^d$ . He gave an inductive existence proof for Euler's characteristic of the "Konvexring" - a measure is assigned to an element  $S$  of this ring by advancing a  $(d-1)$ -dimensional hyperplane  $H$  orthogonal to the  $x$ -axis from left to right, and by considering the measure of  $S \cap H$  in  $\mathbb{R}^{d-1}$  which only changes at finitely many  $x$ -values. Much later [Ha 68] Hadwiger extended this approach and defined the principle (which he called "Schnittrekursion") in a systematic way.

The name of these algorithms comes from their characteristic property that a figure in the plane is processed by advancing a "brush" (often a straight line) from left to right across the figure. Processing is strictly local: No backing up ever occurs, and the lookahead reaches to the next "transition point" only.

Plane-sweep algorithms promise to be efficient for many applications of practical interest in computer-aided design (e.g. for VLSI), computer graphics (for instance, scan conversion), and geographic data processing (consistency checking of map data). Due to this motivating factor, the scope of problems that can be handled by plane-sweep algorithms has been extended in various papers, such as [BO 79] and [NP 82].

Let us first review the concepts needed to understand plane-sweep algorithms and introduce the motivation and terminology used in the rest of this paper (see also Nievergelt and Preparata [NP 82]).

Consider a configuration of geometric figures given by a collection of straight line segments, as shown in figure 1-1.

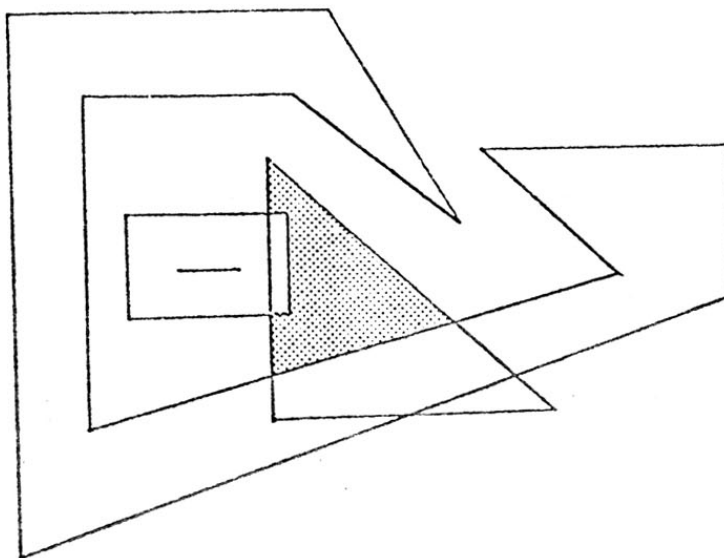


Figure 1-1. A configuration consisting of 4 figures: "line", "triangle", "rectangle", and "spiral". The figure has a total of  $n = 16$  line segments and  $s = 6$  unknown intersections.

Interesting topological and geometric questions regarding the figure include *containment* (the rectangle contains the line but is not contained by the spiral), *intersection* (the triangle intersects the rectangle and the spiral), *region identification* (the shaded region of the plane is covered by the triangle only), and *measurement* (area or length of perimeter of the shaded region). Solutions by plane-sweep algorithms working in time  $O((n+s) \log n)$  are known where  $n$  is the number of line segments, and  $s$  the number of (initially unknown) intersections. Thus  $n$  measures the size of the input data, and  $s$  the complexity of the data (and often the size of the output).

Some of the problems above can be solved by obvious exhaustive search algorithms that work in time  $O(n^2)$  by checking every pair of line segments for an intersection. Since  $s = O(n^2)$ , plane-sweep algorithms have a worst case time behavior of  $O(n^2 \log n)$  which at first sight seems to make them unattractive. However, such "dense" configurations characterized by  $s = O(n^2)$  rarely occur in practice; realistic configurations

tend to have  $s = O(n)$ . For these applications plane-sweeps are very useful as they - surprisingly - solve many seemingly complex problems at the asymptotic cost  $O(n \log n)$  of sorting.

Plane-sweep algorithms superimpose an x-y coordinate system on the geometric configuration to be processed. This arbitrary choice of a sweep direction for problems that are independent of coordinate systems is an esthetic blemish, but for some applications, such as raster scan conversion or drum plotters, it mirrors in a natural way the constraints imposed by the mode of operation of devices.

The x-y coordinate system is represented by two data structures common to all plane-sweep algorithms: The x-structure  $X$  and the y-structure  $Y$ . In addition, there are one or more problem-dependent data structures.  $X$  is a queue representing the tasks still to be accomplished. At any time, it contains the known transition points to the right of the brush, sorted according to x-coordinate. As the next transition point is processed it gets deleted from  $X$ , and newly discovered transitions get inserted. The y-structure which is usually implemented by a balanced tree represents the state of the current cross section of the configuration, at the position of the brush. The information contained in  $Y$  remains unchanged for a slice between two transition points; it must be updated as the brush passes a transition point. Figure 1-2 illustrates these concepts.

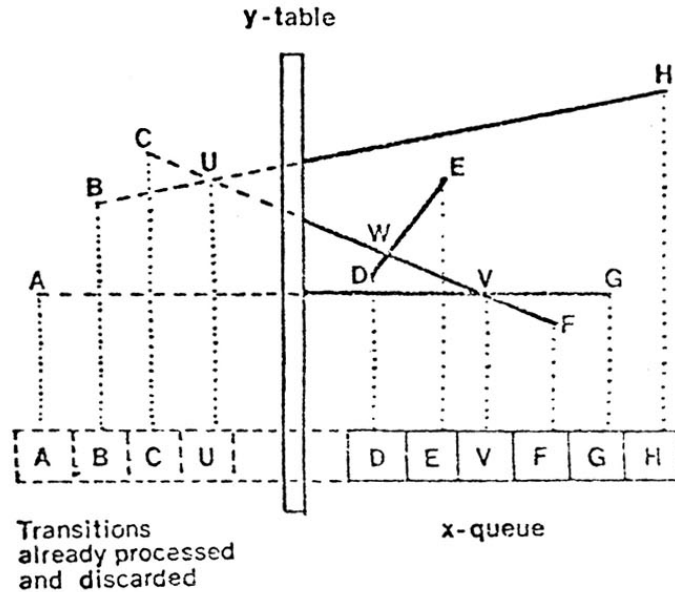


Figure 1-2. The data structures common to all plane-sweep algorithms. A through H are the transition points known before the sweep starts; U, V, W are discovered during the sweep. A, B, C, U have been processed and discarded; D, E, V, F, G, H are known at this time and wait to be processed. W is not yet in the queue, as it will be discovered only at transition D.

Plane-sweep algorithms work according to the following schema:

proc SWEEP:

X ← all transition points known initially, sorted by  
x-coordinate;

Y ← ∅;

Initialize problem-dependent data structures;

while X ≠ ∅ do

    P ← MIN(X);

    TRANSITION(P)

od

corp.

The core of the algorithm, procedure TRANSITION, basically consists of the following:

1. With  $y$  of  $P(x,y)$ , locate the entry in the  $y$ -structure  $Y$  with  $y$ -coordinate identical or nearest to  $y$ , then update  $Y$ .
2. Check adjacent line segments for intersection.
3. Insert newly found transition points into  $X$ .
4. Problem-dependent operations.

This skeleton already allows a rough analysis of the asymptotic time performance. With  $n$  line segments given, at most  $2n$  transition points are known initially.  $s$  transitions (intersections) are discovered during the sweep. For each of the  $2n + s$  transitions we perform the operation  $P \leftarrow \text{MIN}(X)$  and the four steps described above:

- (1)  $P \leftarrow \text{MIN}(X)$

This can be done in time  $O(\log(2n + s)) = O(\log n)$  (since  $s = O(n^2)$ ) if  $X$  is implemented by a tree structure, or even in time  $O(1)$  if all transition points are known initially.

- (2) Locate  $y$  and update  $Y$

As  $Y$  contains at most  $n$  entries sorted according to  $y$ -coordinate, and as update operations are local in the vicinity of  $y$ , these operations can be done in time  $O(\log n)$ .

- (3) Check adjacent line segments for intersections

Time  $O(1)$  is needed.

- (4) Insert intersections found into  $X$

Time  $O(\log(n + s)) = O(\log n)$  is needed.

- (5) Problem-dependent operations

For many problems of interest these can be done in time  $O(1)$  or  $O(\log n)$ .

This sums up to  $O((n + s) \log n)$  as mentioned earlier. In special circumstances a time performance of  $O(n \log n + s)$  can be achieved (e.g. [MS 83]). As we are interested in the generality of plane-sweep algorithms, we do not discuss these cases.

As a summary of this brief review of plane-sweep algorithms: they are well understood, very general in their applicability to different problems, and efficient for the large class of applications where data "spread evenly across the working plane", characterized by  $s = O(n)$ .

In contrast to the two-dimensional case, it is not yet clear whether efficient multi-dimensional *space-sweep algorithms* exist for problems of practical interest. As examples of initial investigations into this question, let us mention the use of plane-sweep techniques for hidden-surface elimination (Schmitt [Sch 81]), and of space-sweep algorithms for computing the subdivision of the space given by a finite number of hyperplanes by Bieri and Nef [BN 82]. The former could well be called a "two-and-a-half-dimensional" problem (superposition of several two-dimensional problems), and thus it is not clear whether its results generalize to  $k \geq 3$  dimensions. The latter is a truly  $k$ -dimensional problem, and provides an interesting example worth extending.

In this paper, we present in an intuitive but systematic way a space-sweep algorithm that completes the intersection of two convex polyhedra in time  $O(n \log n)$ . This upper bound has previously been achieved by Muller and Preparata [MP 78]. We find it interesting to show how a sweep algorithm achieves the same result by an entirely different technique.



## 2. SIMPLIFICATION OF THE PROBLEM

In this paper we study the problem

ICP (Intersection of two Convex Polyhedra): "Given two convex polyhedra in the form of a boundary representation, calculate their intersection."

This problem has previously been studied by Muller and Preparata [MP 78] who established an upper bound of  $O(n \log n)$  for polyhedra having a total of  $n$  vertices. Later, attention has shifted to testing the intersection of two preprocessed polyhedra; Dobkin and Kirkpatrick [DK 82] are able to do this in time  $O((\log n)^2)$  after  $O(n^2)$  preprocessing. More recently, the same authors [DK 83] have presented a linear time algorithm that allows to detect whether two convex polyhedra intersect. We return to the harder problem of calculating the intersection, rather than merely testing for intersection, and achieve the same time bound as Muller and Preparata by an entirely different and simpler technique.

Our algorithm works by first determining some points in the intersection of the surfaces of the two polyhedra  $P_0$  and  $P_1$ , using space-sweep. Starting from there it constructs all edges of  $P_0 \cap P_1$  and thus the resulting convex polyhedron by graph exploration methods. Minor modifications in the graph exploration phase allow to construct  $P_0 \cup P_1$  or  $P_0 \setminus P_1$  instead of  $P_0 \cap P_1$ .

Throughout this paper we assume that no two corners of one of the polyhedra have identical  $x$ -coordinates. This can always be achieved by a slight rotation of the coordinate system, requiring linear time in addition to the initial sorting of the corners. We also assume that all faces of the polyhedra  $P_0$  and  $P_1$  are triangulated from their respective point of minimal  $x$ -coordinate by *improper edges*. The "boundary representation" mentioned above is basically the doubly connected edge list introduced by Muller and Preparata [MP 78],

enriched by the improper edges and by pointers between faces and the improper edges contained therein.

In the following the word face (bounding triangle) always refers to a bounding face of one of the polyhedra without (with) consideration of improper edges. Thus a face may consist of several co-planar bounding triangles.

Problem ICP is reduced to a simpler problem. The edge set  $E$  of the resulting convex polyhedron  $P_0 \cap P_1$  consists of a set  $E_1$  of line segments on the surface of both polyhedra, and a set  $E_2$  of line segments that are edges or part of edges of only one of the polyhedra. Figure 2-1 gives an example.

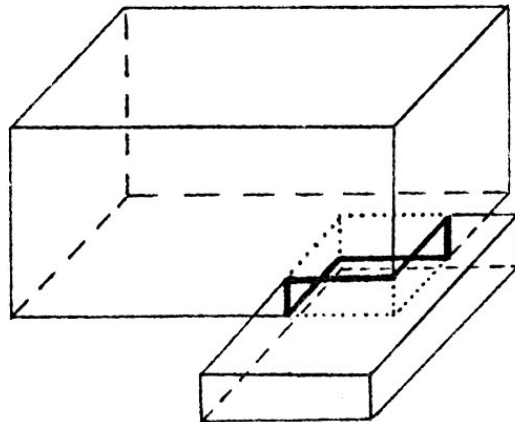


Figure 2-1 Two bricks penetrating each other. Heavy lines are edges in  $E_1$ , dotted lines edges in  $E_2$ .

$E_1$  is naturally composed of connected components; the example in figure 2-1 has only one component. It will be shown (lemma 1) that  $P_0 \cap P_1$  can be systematically constructed in time  $O(n)$  if at least one point (on an edge) of each component of  $E_1$  is known. With this in mind, we define the following problem that lends itself more directly to the space-sweep approach:

Problem ICP': "Given two convex polyhedra  $P_0$  and  $P_1$  with a total of  $n$  corners, compute a set of intersection points of proper edges of  $P_1$  with bounding triangles of  $P_{1-i}$ ,  $i = 0$  or  $i = 1$ . This set must contain at least one point of each connected component of  $E_1 \subseteq P_0 \cap P_1$ ."

Lemma 1: a solution to problem ICP' allows to compute

a)  $P_0 \cap P_1$  ,            b)  $P_0 \cup P_1$  ,            c)  $P_0 \setminus P_1$   
in time  $O(n)$ .

Proof: We argue about part a) only. Parts b) and c) solely differ in the definition of  $E_2$ , and thus in a modified determination of the edges in  $E_2$ .

The basic idea is to consider the solution set to ICP' as part of the set of vertices of a graph, and, starting from these vertices, to explore the graph by a systematic method. To have a sensible stopping criterion we want to know  $E_1$  completely before we start exploring  $E_2$ .

Let  $S$  be the solution set to ICP'. As shown in section 4, we can get  $x \in S$  as intersection of  $e$ , the proper edge separating the bounding triangles  $F'$  and  $F''$  of  $P_1$ , with the bounding triangle  $F$  of  $P_{1-i}$ . We process all points  $x \in S$  as follows: We construct all edges of  $E_1$  incident to  $x$ . On the proper edges of  $P_1$  ( $P_0$ ) extending from  $x$  into the interior of  $P_0$  ( $P_1$ ) (observe that there may be more than one of these if  $x$  is vertex of  $P_0$  or  $P_1$ ), the *candidates for  $E_2$* , we mark  $x$ , and store these edges in a set. If such an edge is marked already, it penetrates one of the two polyhedra; in this case we add the segment between the two markings to the set  $E_2$ . Compare figure 2-2.

We have to distinguish three cases concerning the respective position of  $x$ , namely

(case 1)  $x$  is no polyhedron vertex, and lies in the interior of bounding triangle  $F$  of  $P_{1-i}$  ,

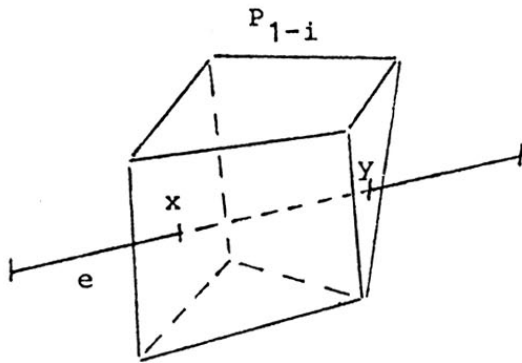


Figure 2-2 Marking of candidates for  $E_2$ .

$e$  is edge of  $P_i$  and intersects  $P_{1-i}$  in  $x$ .  
 $e$  penetrates  $P_{1-i}$  from  $x$  on to the right; this part of  $e$  becomes a candidate for  $E_2$ , and  $x$  is marked on  $e$ . If  $e$  is later explored starting from  $y$  ( $y$  may belong to a different component of  $E_1$ ), the mark  $x$  is detected; we then add  $\overline{xy}$  to the set  $E_2$ .

(case 2)  $x$  is the intersection of two edges but no polyhedron vertex, and

(case 3)  $x$  is vertex of  $P_i$  or/and of  $P_{1-i}$ .

Case 1 - as illustrated in figure 2-3 - is the standard case.

In case 2 let  $x$  be the intersection of edge  $e$  with edge  $f$  separating bounding triangles  $F$  and  $F''$  of  $P_{1-i}$ . Ordinarily (and analogously to case 1), we can compute the structure around  $x$  in time  $O(1)$  by intersecting  $F$  and  $F''$  with  $F'$  and  $F'''$ . A problem arises, however, if  $F$  or  $F''$  is co-planar with  $F'$  or  $F'''$ . Say,  $F$  is co-planar with  $F'$ .  $F'$  is part of face  $G_i$ ,  $F$  is part of face  $G_{1-i}$ . To avoid getting irrelevant and undesired intersections in the interior of  $G_i \cap G_{1-i}$ , we treat the faces as a whole and compute  $G_i \cap G_{1-i}$  in time  $O(\deg_i(G_i) + \deg_{1-i}(G_{1-i}))$ , using the known method due to

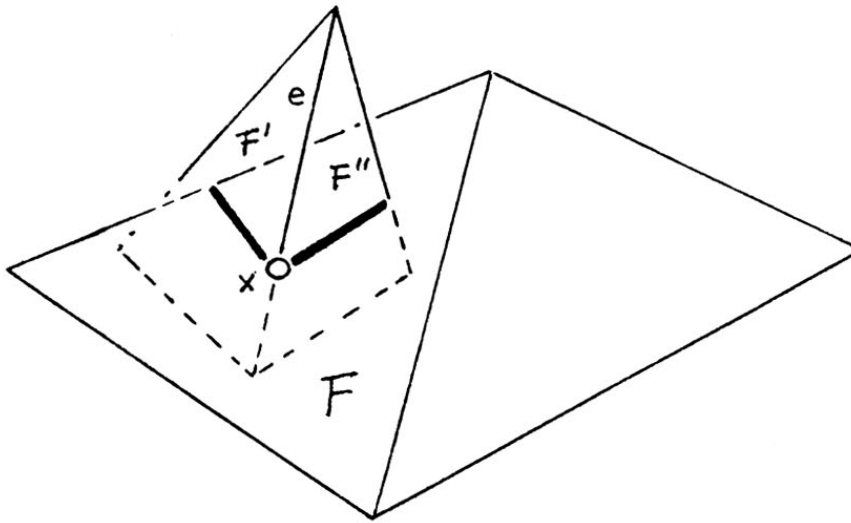


Figure 2-3 Edge  $e$  intersects bounding triangle  $F$  in  $x$ . Edges  $F \cap F'$  and  $F \cap F''$  belong to  $E_1$ . The part of  $e$  extending into the interior of the other polyhedron is candidate for  $E_2$ ;  $x$  is marked on  $e$ .

Shamos [Sh 75].  $\text{deg}_i(G)$  denotes the number of vertices of convex polygon  $G$  in  $P_i$ . We mark the two faces as "done", and then process the vertices of the intersection polygon.

Case 3 can be divided into three subcases. Let  $x$  be a vertex of  $P_0$ . Then  $x$  can lie in the interior of a bounding triangle  $F$  of  $P_1$ , or on an edge  $f$  of  $P_1$ , or it can even be a vertex of  $P_1$  as well. In the first case we intersect  $F$  with all bounding triangles of  $P_0$  incident to  $x$ , and thus get all edges in  $E_1$  (and candidates for  $E_2$ ) incident to  $x$  in time  $O(\text{deg}_0(x))$ . Here  $\text{deg}_i(x)$  denotes the degree of  $x$  in polyhedron  $P_i$ , improper edges included. In addition, we might have to treat co-planar faces as described in case 2 above. In general, we will get two edges in  $E_1$ , and some candidates for  $E_2$ , as shown in figure 2-4.

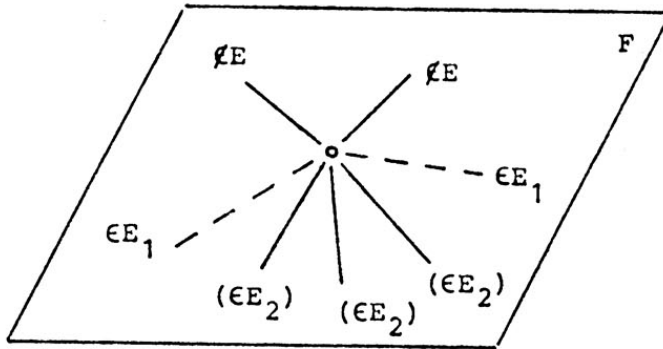


Figure 2-4 Vertex  $x$  of  $P_i$  lies in bounding triangle  $F$  of  $P_{1-i}$ .

The second subcase is analogous; time  $O(\deg_0(x))$  suffices, apart from the time for treating co-planar faces.

If  $x$  is vertex of both polyhedra, however, the situation is considerably more subtle. The naive approach might result in quadratic running time. Therefore we transform the problem in a suitable manner to a two-dimensional problem. Basically, we find a plane  $D$  intersecting all edges of  $P_0$  incident to  $x$  in time  $O(\deg_0(x))$ .  $P_0 \cap D$  is a convex polygon,  $P_1 \cap D$  a convex polygonal region. We intersect these two plane figures in time  $O(\deg_0(x) + \deg_1(x))$ , using Shamos' method. The structure around  $x$  can be inferred from the resulting polygon in a straightforward manner. Again, co-planar faces might have to be handled in addition.

In all three subcases we afterwards mark  $x$  in the polyhedron it belongs to.

This way we have constructed all edges in  $E_1$  (and candidates for  $E_2$ ) incident to a point  $x \in S$  in total time  $O(n)$ . Total time  $O(n)$  is also sufficient for the treatment of co-planar faces since the computation described in case 2 is performed at most once for each face.

Starting from these edges we first want to explore  $E_1$  completely. This is done by working with two sets  $W_1$  ( $W_2$ ), that initially contain all the known members of  $E_1$  (candidates for  $E_2$ ). As long as  $W_1$  contains any element an edge  $e = (x,y)$  is removed from  $W_1$  and processed. Processing an edge means checking whether its endpoints were processed already; an endpoint that was not is processed according to the pertinent case as described above, thereby possibly adding new edges to  $W_1$ ,  $E_1$ ,  $W_2$ ,  $E_2$ . The time for the whole procedure sums up to  $O(n)$  by arguments above if one can decide fast (i.e., in constant time) whether an edge endpoint was processed already. This is indeed possible if we mark points that are not corners of  $P_0$  or  $P_1$  on the edges they lie on in a sensible way, such that no edge eventually bears more than two markings. The only question as to where one should mark arises in case 2 above ( $x = e_i \cap e_{1-i}$ ) if  $e_i$  lies (partly) in a face of  $P_{1-i}$ . We mark  $x$  on  $e_{1-i}$  if this edge does not lie in a face of  $P_i$ ; otherwise (two co-planar faces),  $e_i$  and  $e_{1-i}$  will be proper edges (the algorithm in section 4 will deliver no intersection with an improper edge in this case), and we can safely mark  $x$  on both of them (details left to the reader).

By now we have explored all of  $E_1$  in time  $O(n)$ , and candidates for  $E_2$  can be added to set  $E_2$ . Starting from the interior endpoints of these candidates it is easy to find the remaining edges that belong to  $E_2$  in time  $O(n)$ .  $\square$

More details of the proof above, especially of the corner-in-corner subcase, can be found in [He 84].

### 3. BASIC CONCEPTS

By analogy to plane-sweep algorithms, a space-sweep algorithm operates by advancing a plane through space. The x-structure is a queue  $X$  containing (for our problem) all  $n$  corners of the two polyhedra. The yz-structure that replaces the y-structure represents the state of the cross section. The latter has (for each one of the two polyhedra) the form of a convex polygon.

The set of edges (of one polyhedron) intersected by the sweep plane forms a cycle whose neighbor edges bound the same bounding triangle of the polyhedron. This leads us to

Definition 1: Let  $P_i$ ,  $i = 0, 1$ , be one of the polyhedra. Let  $e_j$ ,  $0 \leq j < n_i$ , be the cyclically ordered sequence of edges intersected by the sweeping plane. A *prong* is the portion of a bounding triangle bounded by two consecutive edges  $e_j$  and  $e_{(j+1) \bmod n_i}$ , and, to the left, by the sweeping plane. The cyclically ordered set of prongs forms the *crown*  $C_i$ .

Observe that, because of the triangulation chosen, the part of a face of  $P_i$  to the right of the sweeping plane is either completely or not at all part of the crown  $C_i$ .

Obviously all prongs are either triangles or quadrilaterals, and we can classify crown edges as follows:

Definition 2: Let  $P_i$  and  $e_j$  be as in Definition 1. The edges of the polygon formed by connecting the intersection points in a circular fashion are called *base edges*, their x-coordinate is the *base*; the edges  $e_j$  of the original polyhedron are called *forward edges*, and all other edges of the crown are called *prong edges* (they connect tips of prongs).

These definitions are pictured in figure 3-1.

Thus we can represent a cross section in a way analogous to the one-dimensional cross section of plane-sweep algorithms:



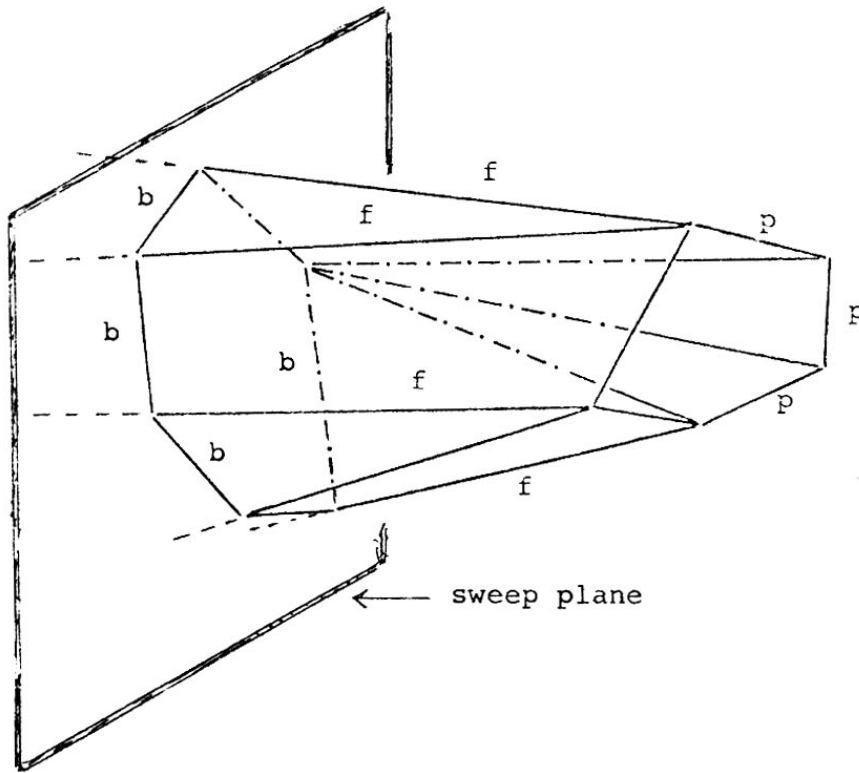


Figure 3-1 Edge classification of a crown.  
b, f, p denote the three edge types.  
Point q is the polyhedron corner just processed.  
Forward edges starting left of q are shown dashed  
to the left of the sweeping plane.

here the linearly ordered y-structure is replaced by a circularly ordered yz-crown. Due to this circular ordering, the crown can be searched in logarithmic time by binary search for angular arguments, to determine the relative position of a new transition point w.r.t. the crown edges.

To store crown  $C_i$  in a balanced tree we select an axis line  $L_i$ , e.g. the line connecting the vertices of  $P_i$  having the minimum and maximum x-coordinate, respectively. This axis always intersects the sweeping plane, and we can represent  $p \in \mathbb{R}^3$  w.r.t.  $L_i$

by the cylindrical coordinates ( $x$ ,  $\alpha$ , radius). Here  $x$  is  $p$ 's  $x$ -coordinate, and ( $\alpha$ , radius) are  $p$ 's polar coordinates in the  $yz$ -plane through  $p$ , with pole on  $L_1$  and  $\alpha$ 's measured against, say, the positive  $y$ -axis. A forward edge is represented by the cylindrical coordinates w.r.t.  $L_1$  of its endpoints, i.e., by  $(x_0, \alpha_0, r_0, x_1, \alpha_1, r_1)$ . Note that for every  $p$  on such a forward edge with  $x$ -coordinate  $x$ ,  $x_0 \leq x \leq x_1$ , we can calculate ( $\alpha$ ,  $r$ ) in constant time. The radial representation is depicted in figure 3-2; in figure 3-3 formulae are given that help for the calculation of ( $\alpha$ ,  $r$ ) - for simplification, the axis is assumed to be the  $x$ -axis.

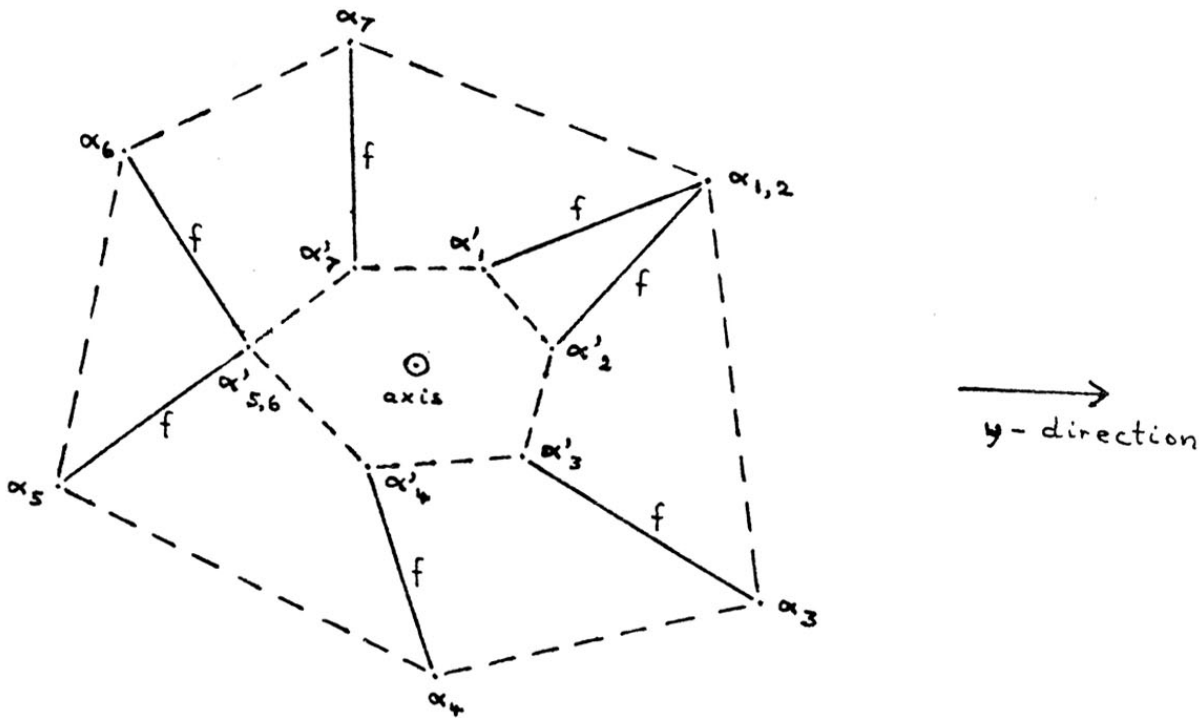


Figure 3-2 Radial representation of a crown, viewed parallel to the axis.  $\alpha_i$ 's are angles of left,  $\alpha'_i$ 's angles of right endpoints of forward edges.

Note that although alphas and radii change with  $x$ , the relative ordering of crown edges remains invariant between transitions.

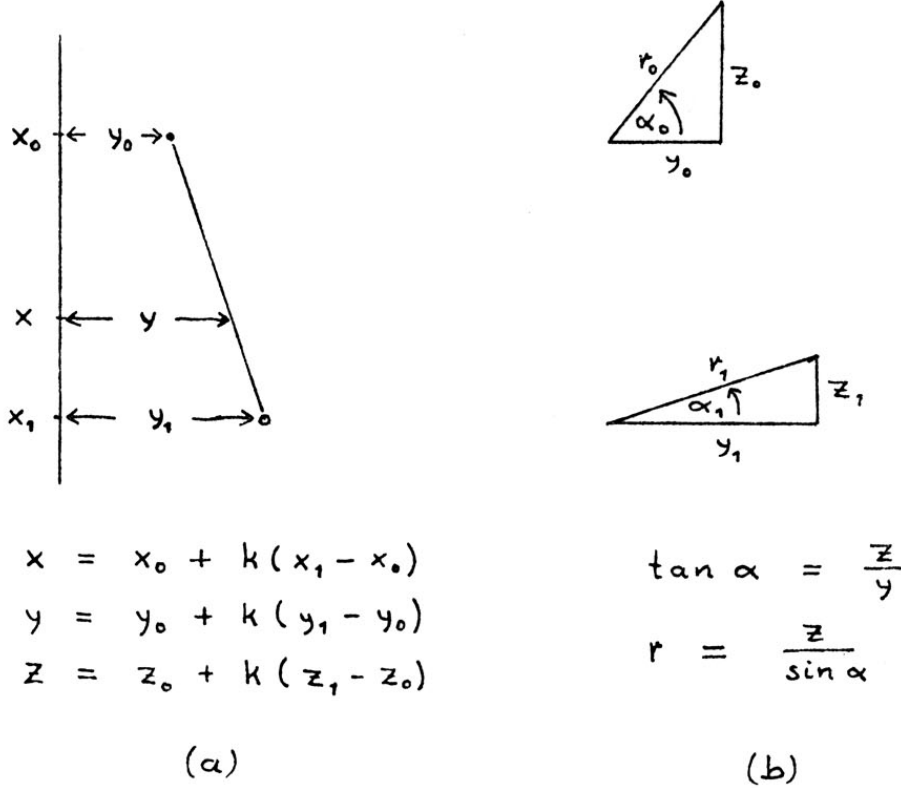


Figure 3-3 Coordinate transformation for a forward edge  $f = (p_0, p_1)$  if the axis is the x-axis.  
 (a) Projection into the xy-plane.  
 (b) Projections into the yz-plane at  $x = x_0$  and at  $x = x_1$ .

This allows us to store the crown as a balanced binary tree, organized with respect to alfas.

Before we can process (see next section) a new polyhedron vertex  $p = (x, y, z)$  (with cylindrical coordinates  $(x, \alpha, r)$ ), we have to know where it is located in the respective crown. Therefore we use angular binary search to find a crown edge the right endpoint of which is  $p$ . If the intersection of any forward edge of the crown with the yz-plane through  $p$  has cylindrical coordinates  $(x, \alpha^*, r^*)$ , we have to search until we find an edge such that  $\alpha^* = \alpha$ .

#### 4. THE ALGORITHM

Our algorithm will construct two crowns, one for each polyhedron, and will find their intersections (as specified in problem ICP', i.e., at least one per connected component of  $E_1$ ). Thereby, if edge  $e$  lies completely in a face of the other polyhedron, "intersection" is defined to be an end-point of  $e$ . Advancing in the  $x$ -queue (that initially contains all  $n$  corners) from vertex to vertex of the two polyhedra, we perform one transition per vertex. We will first present procedure TRANSITION, the core of our algorithm, and then show that the algorithm is correct and stays within the desired time limit of  $O(n \log n)$ .

Processing vertex  $p$  of polyhedron  $P_i$  we assume that at least one point was found already of every component of  $E_1$  completely left of  $p$  (and possibly including  $p$ ); we only look for intersections to the right of  $p$ . One execution of TRANSITION will deliver zero, one, or several intersection points of the two polyhedra on an edge or in a face of  $P_i$  starting at  $p$ . To achieve this, we first intersect all proper edges of  $P_i$  starting at  $p$  with the opposite crown  $C_{1-i}$ . If none of the two edges bounding a face  $F$  of  $P_i$  starting at  $p$  intersects the opposite crown, a part of  $P_{1-i}$  could nevertheless penetrate face  $F$ , as shown in figure 4-1.

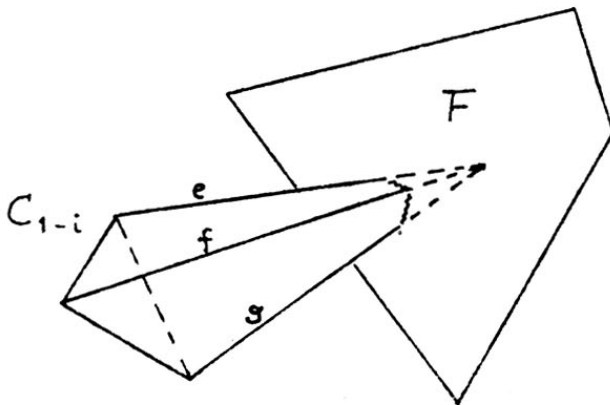


Figure 4-1 No bounding edge of face  $F$  of  $P_i$  intersects  $C_{1-i}$ ; however, forward edges of  $C_{1-i}$  penetrate  $F$ .

To detect some of these intersections, we intersect, in this case, an arbitrary proper forward edge of crown  $C_{1-i}$  with  $F$ . Lemma 2 below asserts that this is sufficient to find at least one point of each component of  $E_1$ .

Specifically, a transition is performed as follows ( $S$  is an initially empty set of intersections):

- (1) proc TRANSITION( $p, i$ ):
- (2)     update crown  $C_i$  of polyhedron  $P_i$ ;
- (3)     if exactly one edge  $e$  of  $P_i$  starts at  $p$
- (4)         then intersect  $e$  with the crown  $C_{1-i}$  of  $P_{1-i}$  and  
           add all intersections to set  $S$
- (5)     else for all faces  $F$  of  $P_i$  starting at  $p$   
           (possibly consisting of several prongs)
- (6)         do intersect the starting proper edges of  $F$   
           with the crown  $C_{1-i}$ , and add all inter-  
           sections to  $S$ ;
- (7)         if no intersection is found
- (8)             then choose a proper forward edge of crown  
                   $C_{1-i}$  and intersect it with  $F$ ; add  
                  intersection, if any, (in the pertain-  
                  ing bounding triangle) to  $S$
- (9)             fi
- (10)            od
- (11)     fi
- (12) corp.

Lemma 2: Performing TRANSITION once for each of the vertices of the two polyhedra sorted into a common queue correctly solves problem ICP'.

Proof: It is clear that a set of intersections is computed. Thus we only need to show that at least one point of each component of  $E_1$  will be reported.

Let  $K$  be a component of  $E_1$ , and let  $v$  be a vertex of  $K$  with minimal  $x$ -coordinate. Clearly  $v$  is the intersection of an

edge  $e$  of polyhedron  $P_i$  with a face  $F$  of polyhedron  $P_{1-i}$ . If  $v$  is not reported then  $e$  must start before  $F$  (in the sweep order). Consider the state of the space-sweep immediately after the start vertex  $p$  of  $F$  is encountered. At this point  $e$  is a forward edge of the crown  $C_i$  of  $P_i$ .

Conceptually trace  $K$  in face  $F$  and crown  $C_i$ , starting at  $v$ . Two cases may arise:

Case\_1: We are not able to trace  $K$  completely, i.e., we first hit either a bounding edge  $e'$  of  $F$  (subcase a)), or a prong edge  $e''$  of  $C_i$  (subcase b)). Because of the minimality of  $v$ 's  $x$ -coordinate we do not leave  $C_i$  by way of a base edge.

a)  $e'$  intersects a prong  $G$  of  $C_i$ . Then  $e' \cap G$  either was detected at processing  $p$ , or it will be detected when the starting point of  $e'$  is processed, since  $G$  is still a prong of  $C_i$  at that time. Compare figure 4-2.

b) Because of the triangulation chosen for faces,  $e''$  is a proper edge. Also,  $F$  still is part of crown  $C_{1-i}$  when the starting point of  $e''$  is processed. Therefore  $e'' \cap F$  is detected at that time.

Case\_2: We are able to trace  $K$  completely in  $F$  and  $C_i$ .

Since  $e$  starts before  $F$  and since it does not intersect a bounding edge of  $F$ ,  $e$  does not lie in face  $F$  but intersects the interior of  $F$ . The intersection with  $F$  of the two prongs of  $C_i$  neighboring  $e$  is completely contained in  $F$ , and the forward edges (other than  $e$ ) of these prongs again intersect the interior of  $F$ . The argument propagates around the crown  $C_i$ . Thus  $K$  is a closed curve in  $F$  running through all prongs of  $C_i$ , and intersecting all forward edges of  $C_i$ . Hence a point of  $F$  is found in line (8) of TRANSITION. Figure 4-1 helps for understanding this case; we may, in line (8) of TRANSITION, select forward edge  $f$  of  $C_i$  and intersect it with  $F$ . □

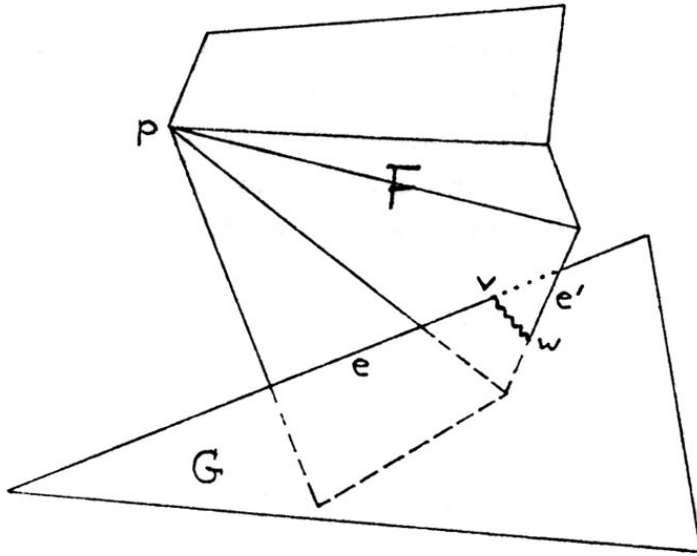


Figure 4-2 Case 1 a) of the proof of lemma 2. Shown are a part of the crown  $C_{1-i}$ , and prong  $G$  of crown  $C_i$ .  $w = e' \cap G$  is detected when the starting point of  $e'$  is processed.

Let us now examine the time required for the different actions of TRANSITION.

Lemma 3: The updating of crown  $C_0$  ( $C_1$ , resp.) can be done in total time  $O(n \log n)$ .

Proof: Consider updating crown  $C_i$  ( $i = 0,1$ ) at a transition vertex  $v = (x,y,z)$ . Let  $c$  be the maximum of the number of forward edges of  $C_i$  before the transition, and the number of such edges after the transition, and let  $d = d_1 + d_2$ , with  $d_1$  ( $d_2$ ) being the number of edges of  $P_i$  ending (starting) at  $v$ . We have to localize the edges ending at  $v$  in the balanced tree representing  $C_i$  for updating  $C_i$  subsequently. This can be done by angular binary search as described in the previous

section. Because of the cyclic ordering of the crown this yields a subtree  $T_v$ , as pictured in figure 4-3.

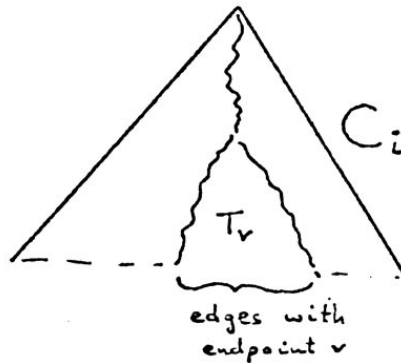


Figure 4-3 The two outermost search paths (bounding  $T_v$ ) for edges with endpoint  $v$ .

Now we split  $C_i$  along the two outermost search paths, drop  $T_v$ , and merge the tree  $T_v^{\text{new}}$  formed from the  $d_2$  edges starting in  $v$  (that are given in cyclic order) with the remainder of  $C_i$ . This can be done in time  $O(\log c + d_2)$ , namely time  $O(\log c)$  for the search with the subsequent splitting of the tree  $C_i$ , time  $O(d_2)$  for the construction of the tree  $T_v^{\text{new}}$ , and time  $O(\log c)$  for the merging of  $T_v^{\text{new}}$  with the remainder of  $C_i$ . The operations necessary are those of a "concatenable queue" ([AHU 74], p. 155ff.) - observe that the remainder of  $C_i$  is a forest of trees with known relative ordering, and with height differences between two neighbors summing up to  $O(\log c)$ . Since  $c < n$ , and since the sum of  $d$ 's over all polyhedron vertices is  $O(n)$ , the lemma follows.  $\square$

Lemma 4: Let  $g$  be a straight line, and let  $C$  be a crown with  $c$  forward edges.  $C \cap g$  can be computed in time  $O(\log c)$ .

Proof: Similarly to the idea of Dobkin and Kirkpatrick [DK 82,



DK 83] we let the balanced tree define a hierarchical representation of the crown. It will help us for finding an intersection if we do not use the crown directly but an extension of it to a convex polyhedron. Specifically, let  $v_1, \dots, v_c$  be the vertices of the base polygon, and let  $w_1, \dots, w_c$  be the right endpoints of the  $c$  forward edges. Neighboring  $v$ 's or neighboring  $w$ 's may coincide. Let  $V := \{v_1, \dots, v_c\}$ ,  $W := \{w_1, \dots, w_c\}$ , and let  $t$  be the right endpoint of the axis of  $C$ , i.e., the rightmost vertex of the respective polyhedron. Consider the convex hull of the points  $V \cup W \cup \{t\}$ . The bounding faces of the convex hull are the base polygon which we will not consider in the sequel (intersections with  $g$  there are not interesting), the prongs of crown  $C$  and the *terminal triangles* ( $tt$ 's) extending from two neighboring but not identical  $w$ 's each to  $t$ . Thus each  $tt$  has exactly one *partner prong* ( $pp$ ); the reverse relation does not hold necessarily. Figure 4-4 presents such a solid object and illustrates the new terms.

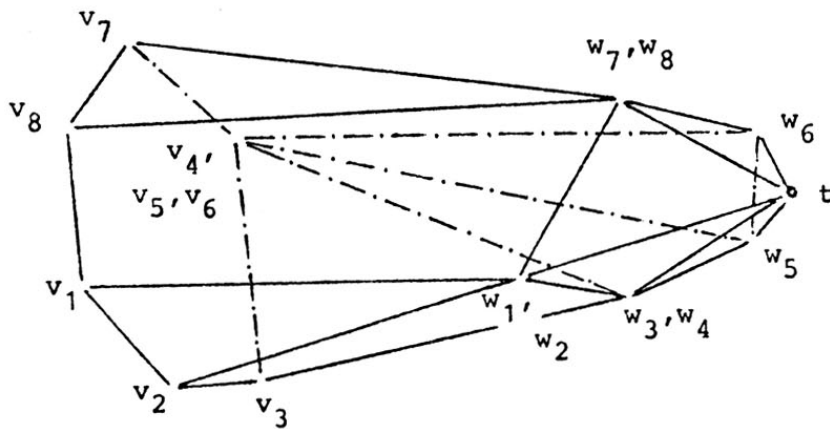


Figure 4-4 Convex hull of a crown with 8 forward edges joining in 5 different right endpoints, together with  $t$ , the rightmost polyhedron vertex. Partner of  $tt$   $w_7w_1t$  is prong  $v_1w_1w_8v_8$ ; prong  $v_7v_8w_8$  has no partner.

We will represent such a special polyhedron hierarchically by a balanced tree. To be specific, let us choose a (2,3)-tree (compare [AHU 74]). Observe that the crown resp. the special polyhedron is kind of a two-and-a-half-dimensional solid for which the two-dimensional hierarchical representation of [DK 82] is basically sufficient. Let  $C^{(i)}$ ,  $i = 1, \dots, k = O(\log c)$ , be the forward edges of the crown stored in depth  $i$  of the tree.  $C^{(k)}$  is  $C$ , and  $P^{(i)}$ , the convex hull of  $C^{(i)}$  and point  $t$ , is an inner approximation of the special polyhedron.  $C^{(i)}$  is shrunk to  $C^{(i-1)}$  by transforming two (or three) prongs of  $C^{(i)}$  each into one - this, in turn, is done by removing the forward edge(s) separating them. Compare figure 4-5.

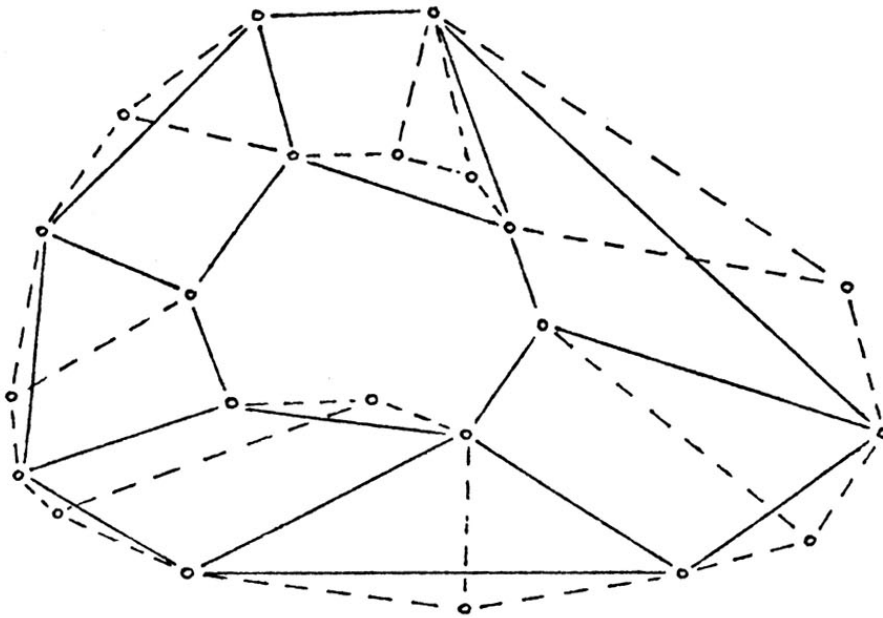


Figure 4-5 a Two subsequent approximations of the crown of a polyhedron, seen from the direction of the positive axis.  $C^{(i)}$  (consisting of prongs with at least one forward edge shown dashed) is made coarser and thus changed into  $C^{(i-1)}$  (solid edges).

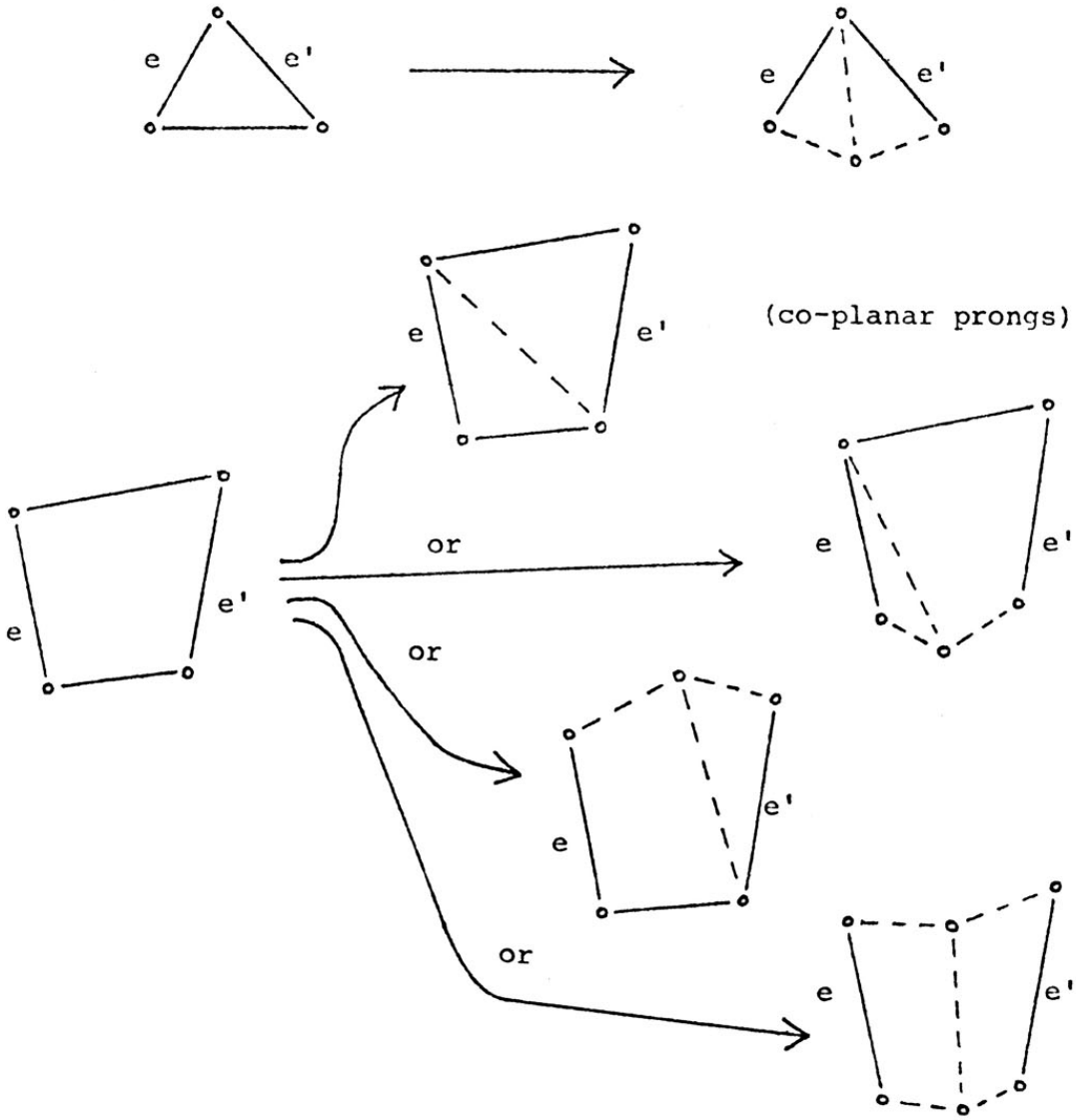


Figure 4-5 b Refinement of the representation of a crown; transition from  $C^{(i-1)}$  to  $C^{(i)}$ .

Forms of the "partitioning" of a prong into two by adding an additional forward edge (dashed; between two neighboring forward edges,  $e$  and  $e'$ , of  $C^{(i-1)}$ ). If two additional edges are inserted even more forms of refining a prong are possible.

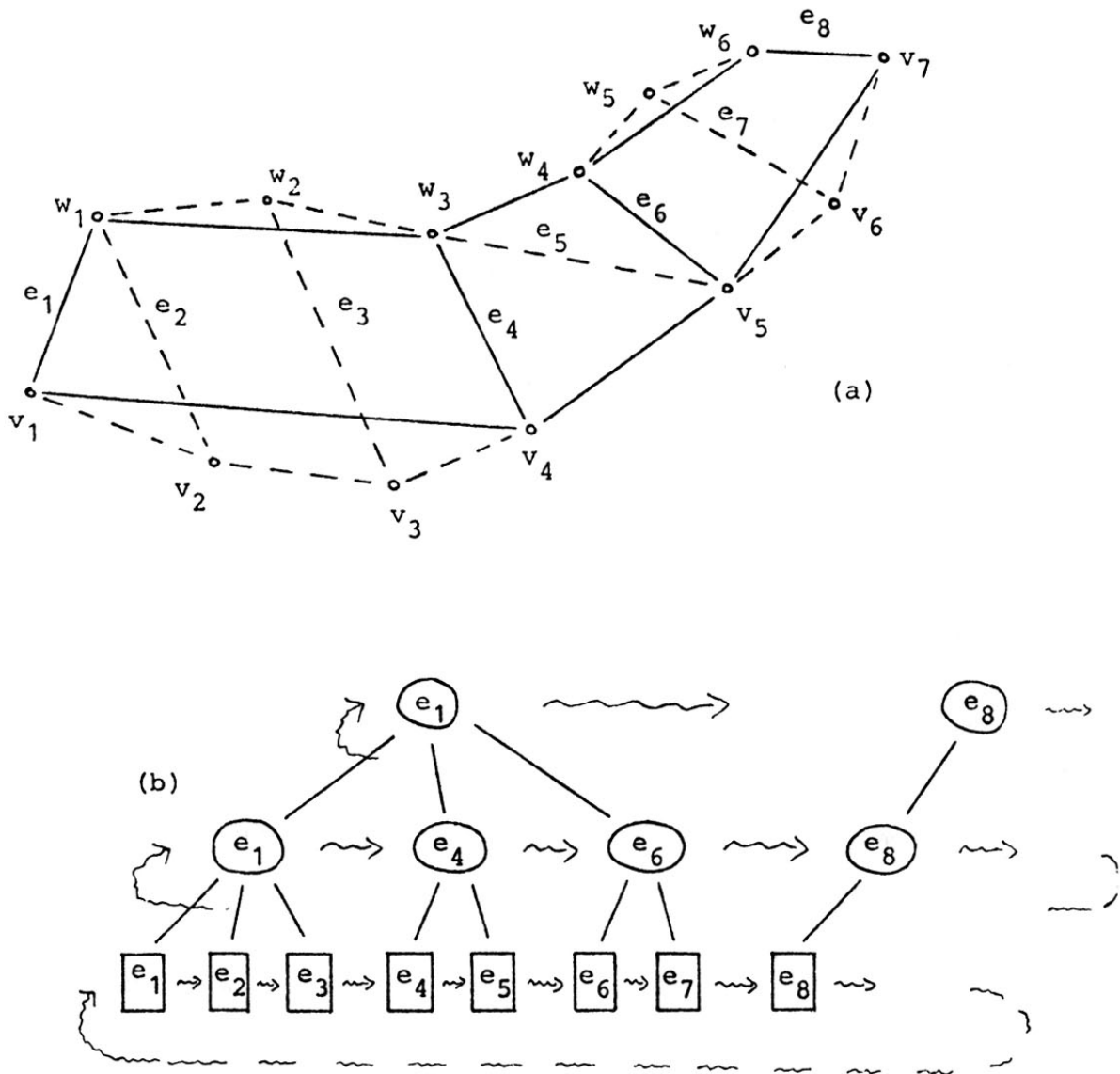


Figure 4-6 (a) Part of crown  $C^{(k)}$  with the corresponding part of  $C^{(k-1)}$  (solid lines).

(b) Hierarchical representation of (a) in the  $(2,3)$ -tree; shown is only the hierarchy information in the internal nodes. From the upper node marked by  $e_1$  one can find the corresponding prong  $w_1 w_6 v_7 v_6$  of  $C^{(k-2)}$  that is not explicitly drawn in (a).

Internal nodes of the balanced tree contain a hierarchy information, apart from the usual order information (for preserving the order during rebalancing). The former is chosen to insure that, in each depth  $i$ ,  $i = 1, \dots, k$ , two neighboring nodes each correspond to a prong of the crown  $C^{(i)}$ . To achieve this, it is sufficient to let leaves of the tree represent forward edges of the crown  $C = C^{(k)}$ , and every internal node the (cyclically) minimal edge of the subtree the root of which it is. Via neighbor pointers on every level one can find the prong corresponding to a node in constant time. Figure 4-6 illustrates the correspondence between a connected section of the crown  $C^{(k)}$  and the pertinent tree representation.

The refinement of the representation, i.e., the transition from  $P^{(i)}$  to  $P^{(i+1)}$ , may be understood as a convex extension of  $P^{(i)}$ . We will present related terms and investigations before we explain the algorithm for determining  $C \cap g$ .

Let  $P = (V, E)$  be an arbitrary convex polyhedron. Let  $Z = (v_1, \dots, v_s, v_{s+1} = v)$  be a closed simple path of edges on  $P$ , i.e.,  $v_i \in V$ ,  $(v_i, v_{i+1}) \in E$  for  $i = 1, \dots, s$ , and  $v_i \neq v_j$  for  $i \neq j$ ,  $i, j \in [1:s]$ .  $Z$  divides the surface of  $P$  into two *segments*. Let  $S$  be a segment. A *convex extension* of  $S$  is an expansion of  $P$  in the region of  $S$  (by adding new nodes and edges, and by possibly removing nodes in the interior of  $S$ , i.e., in  $S$  but not on the path of edges bounding  $S$ ) that preserves convexity. The *roof over  $S$*  is the maximal convex extension of  $S$ . Roofs may be closed or open (i.e., infinite); for instance, the faces of a tetrahedron have open roofs, those of a dodekahedron have closed roofs.

Relevant for our algorithm is the following fact (proved in [He 84], p. 85):

Fact: Let  $S = \{S_1, \dots, S_k\}$  be a partition of the surface of convex polyhedron  $P$  into segments. A line  $g$  that does not

intersect  $P$  can intersect the roofs of at most two of these segments.

Now we are going to compute  $C \cap g$ , guided by Dobkin and Kirkpatrick's algorithm for determining the intersection of a straight line with a convex polygon. In the following, segments will always be tt-pp-pairs, and prongs without partner; compare figure 4-4.

We start with  $P^{(1)}$ , a (possibly degenerated) convex polyhedron with few segments, and determine the part of  $P^{(1)}$  where an intersection could occur if it exists. There we locally proceed to  $P^{(2)}$  and determine a - smaller - section for a possible intersection. We locally expand the polyhedron up to  $P^{(k)}$  to find an intersection if any. If  $g$  intersects  $P^{(k)}$  in a prong of  $C^{(k)}$  this point is reported; intersections with tt's are neglected.

Depending on whether we already have found an intersection  $g \cap C$  or not, we have to distinguish two cases.

A proper intersection (i.e., not only touching point) found in depth  $i$  is processed by refining the respective segment to the next depth. One of the (at most six) new bounding faces must again have an intersection with  $g$ ; we proceed analogously at depth  $i + 1$ .

If no intersection was found so far, the following invariant is maintained before the start of the  $i$ -th iteration (i.e., computation at depth  $i$ ),  $i = 2, \dots, k$ :

(INV. 1):  $S_1, S_2$  are two segments of  $P^{(i-1)}$  with the following property: If  $g$  intersects  $P^{(j)}$  with  $j \geq i$ , then  $g$  intersects one or both of the roofs over  $S_1, S_2$ .  
 $S_1 = S_2$  is allowed.

For the second iteration the invariant is established as follows:

Intersect  $g$  with all bounding faces of  $P^{(1)}$ . If no such intersection is found intersect  $g$  with the roofs of all segments of  $P^{(1)}$ . Observe that such an intersection is computable in constant time. Get at most two segments  $S_1, S_2$  the roofs of which are intersected by  $g$ . Stop if no such segment exists.

The transition from the  $i$ -th to the  $(i+1)$ st iteration,  $i \geq 2$ , is performed as follows:

Refine segments  $S_1, S_2$  to depth  $i + 1$ . Get at most six new segments. If no intersection of  $g$  with one of the maximally 12 new bounding faces is found, intersect  $g$  with the roofs of the new segments. Get at most two new segments  $S_1, S_2$  the roofs of which are intersected by  $g$ . Observe that, if segment  $S$  is refined into two or three new segments, the roofs of the new segments are contained in the roof of  $S$ , and, that roofs of segments not considered never grow either. Stop if  $g$  does not intersect any roof.

Let us briefly mention two special cases that require some modification of our procedure in the  $i$ -th iteration. If  $g$  touches  $P^{(i)}$  in exactly one point on a forward or "terminal" edge, we refine the two incident segments; if a part of  $g$  lies in a prong of  $C^{(i)}$  we refine, per iteration, the outermost two segments concerned, in order to avoid reporting an intersection of  $g$  with an improper edge.

Details of such special cases are left to the reader. It suffices to realize that only constant work is necessary per iteration. Since, due to the convexity of  $P^{(i)}$ , not more than two search paths are followed through in the tree, the time bound follows.  $\square$

Corollary: Let  $C$  be a crown with  $c$  forward edges, and let  $e$  be an edge the left endpoint of which does not lie left of the base of  $C$ .  $C \cap e$  can be computed in time  $O(\log c)$ .

By combining lemmata 2 - 4 we obtain

Lemma 5: The space-sweep algorithm using procedure TRANSITION described above solves problem ICP' defined in section 2 in time  $O(n \log n)$ .

Proof: The total cost for line (2) of TRANSITION is  $O(n \log n)$  according to lemma 3, as is, according to lemma 4 including its corollary, the total cost for lines (4) and (6). Line (8) needs  $O(\deg_1(F))$  per execution, that is, total time  $O(n)$  since it is executed at most once per face  $F$ . For all other lines linear time suffices in total.  $\square$

Together with lemma 1 we finally get our main result

Theorem: The intersection of two convex polyhedra with a total of  $n$  corners can be computed by space-sweep in time  $O(n \log n)$ .



## 5. CONCLUSIONS

We have seen that the space-sweep approach yields an efficient algorithm for intersection of convex polyhedra that matches the performance of the best known algorithm to this problem. Moreover, the method is more universal than the previous solution by Muller and Preparata [MP 78]; we further believe it to be easier to understand.

Thus there are effective space-sweep algorithms for selected problems. However, space-sweep does not seem to offer as general an approach to solving geometric problems in 3 dimensions as plane-sweep does for 2 dimensions. It is an open question whether space-sweep can be effectively applied to more general problems, such as those involving non-convex solids.

## ACKNOWLEDGEMENT

We are grateful to Klaus Hinrichs, Herbert Edelsbrunner, and Athanasios Tsakalidis for helpful comments.

REFERENCES

- [AHU 74] A.V.Aho/J.E.Hopcroft/J.D.Ullman:  
"The Design and Analysis of Computer Algorithms"  
Addison-Wesley Publ. Comp., Reading, Mass., 1974
- [BN 82] H.Bieri and W.Nef:  
"A Recursive Sweep-Plane Algorithm, Determining  
All Cells of a Finite Division of  $\mathbb{R}^d$  "  
Computing 28 (1982), 189-198
- [BO 79] J.L.Bentley and T.A.Ottman:  
"Algorithms for Reporting and Counting Geometric  
Intersections"  
IEEE Trans. on Comp., Vol. C-28 (1979), 643-647
- [DK 82] D.P.Dobkin and D.G.Kirkpatrick:  
"Fast Detection of Polyhedral Intersections"  
Proc. 9th ICALP, Springer LNCS 140 (1982), 154-165
- [DK 83] D.P.Dobkin and D.G.Kirkpatrick:  
"A Linear Algorithm for Determining the Separation  
of Convex Polyhedra"  
Manuscript, 1983
- [Ha 55] H.Hadwiger:  
"Eulers Charakteristik und kombinatorische  
Geometrie"  
J. reine und angew. Math. 194 (1955), 101-110
- [Ha 68] H.Hadwiger:  
"Eine Schnittrekursion für die Eulersche Charakte-  
ristik euklidischer Polyeder mit Anwendungen  
innerhalb der kombinatorischen Geometrie"  
Elemente der Mathematik 23 (1968), 121-132
- [He 84] S.Hertel:  
"Sweep-Algorithmen für Polygone und Polyeder"  
Univ. des Saarlandes, Saarbrücken, Diss. 1984
- [MP 78] D.E.Muller and F.P.Preparata:  
"Finding the Intersection of Two Convex Polyhedra"  
Theoret. Comp. Sci. 7 (1978), 217-236
- [MS 83] Mairson and Stolfi, personal communication, 1983
- [NP 82] J.Nievergelt and F.P.Preparata:  
"Plane-Sweep Algorithms for Intersecting Geometric  
Figures"  
Comm. ACM 25, 10 (Oct. 1982), 739-747

- [Sch 81] A.Schmitt:  
"Time and Space Bounds for Hidden Line and  
Hidden Surface Algorithms"  
Proc. EUROGRAPHICS '81, North-Holland, Amsterdam,  
1981, pp. 43-56
- [Sh 75] M.I.Shamos:  
"Geometric Complexity"  
Proc. 7th ACM STOC (1975), 224-233
- [SH 76] M.I.Shamos and D.Hoey:  
"Geometric Intersection Problems"  
Proc. 17th IEEE FOCS Symp. (1976), 208-215