FAST DETECTION OF POLYHEDRAL INTERSECTIONS

David P. Dobkin+

Electrical Engineering and Computer Science Department
Princeton University
Princeton, NJ 08540

David G. Kirkpatrick

Department of Computer Science
University of British Columbia
Vancouver, British Columbia

ABSTRACT

Methods are given for unifying and extending previous work on detecting polyhedral intersections. The technique of dynamic (vs. static) description is introduced and used to extend previous results. New upper bounds of O(log n) and O($\log^2$n) are given on plane-polyhedron and polyhedron-polyhedron intersection problems.

1. Introduction

A fundamental problem in geometric computing is that of detecting polyhedral intersections. Versions of this problem lie at the core of such problems as linear programming[Da], hidden surface elimination[War,Wat] and computer vision[Wi]. In a previous paper [CD], the detection problem for polyhedra intersection problems was shown to be of lower complexity than the computation problem. Solutions of complexity $c \log^k n$ (for fixed constants c and k) were given for instances of the former problem (of input size n). And, linear lower bounds are known on the computation problems[SH,MP].

The results of [CD] are unified and extended here. This is done by extending the method of dynamically defining convex polyhedra from [K]. Using this method, convex polygons and polyhedra are defined through a hierarchy of descriptions each refining previous definitions. A coarse description of the object is given. Then, at each stage, more detail is given about a smaller part of the object. In moving from step to step of the detection algorithm (and level to level of the hierarchy), finer descriptions of smaller portions of the object are given. These portions are those which are shown to be relevant to possible intersections if the two objects intersect. Details of the hierarchical method used for dynamic description are given in Section 2.

The efficiency of our algorithms is achieved by balancing the complexity of the algorithm. These results are presented in Section 4.

Static-dynamic intersection methods are considered in Section 3. Applications to detecting intersections between polyhedra, which are dynamically described, and lines and planes

+This research supported in part by the National Science Foundation under Grant MCS81-14207.

are given. Since lines and planes lack structure, they are described in a static manner. O(log n) operation algorithms are given for these problems.

The conclusions include a presentation of some open problems involving higher dimensional extensions and some applications of the algorithms to relevant problems.

2. Hierarchical Representations of Convex Polyhedra

As stated above, the algorithms are based on hierarchical descriptions of objects. These descriptions give two representations - an inner one and an outer one. In the inner representation, the polyhedron is "grown" from descriptions of increasing detail. Each new description gives a more accurate description of a polyhedron interior to the original object. If at any stage in the computation an intersection with an inner representation is detected, this is guaranteed to also be an intersection with the object itself. The outer representation "shrinks" the polyhedron from a superset of its points by adding detail locally as necessary. If at any point in this description a non-intersection is detected, no intersection of the original objects can occur. Details of this method are given below for polygons and polyhedra. In the next section, these methods are used to detect static-dynamic intersections. A variant of this method is used in section 4 to detect dynamic-dynamic intersections. Further unification of the techniques of this paper will allow the results of section 4 to be presented in a hierarchical form.

2.1. The two dimensional case

Let P be a polygon with vertices V(P) and edges E(P). Inner and outer representations of P are defined by:

Definition: An *inner polygonal representation* of P is an ascending chain P_i i=1,...,k of polygons where P_{i+1} is derived from P_i as follows: Each edge (u,v) in E(P_i) which is not in E(P) is replaced by edges (u,w) and (w,v) and the vertex w is added to V(P_i) in forming V(P_{i+1}). P_k is P.

Definition: An *outer polygonal representation* of P is a descending chain P_i i=1,...,k of polygons where P_{i+1} is derived from P_i as follows: If (u,v) and (v,w) are adjacent edges of P_i and v is not a vertex of P, then k and l are chosen on (u,v) and (v,w) such that (k,l) is the extension of an edge of P and {(u,v),(v,w)} is replaced by {(u,k),(k,l),(l,w)}. k and l are selected to balance the number of unspecified edges in (u,k) and (l,w).

An inner representation is a "growing out" of a polygon by adding new vertices to extend its perimeter. And, an outer representation if a "growing in" by adding new edges to limit its area. For an inner representation, the vertices considered at each stage are a subset of those in V(P). For an outer representation, the halfplanes defining the polygon at each stage are a subset of those defining P. However, this may lead to edges of P_i which contain the corresponding edges of P. Examples of these representations are given in Figure 1. Representations requiring space O(n) and having height k = O(log n) are formed by applying the process mentioned in each definition. For inner representations, P_i can be formed from P_{i+1} by deleting every other vertex. For outer representations, P_i can be formed from P_{i+1} by deleting every second bounding half-plane.

lines determine the segment of the line which intersects the polygon (or polyhedron) and thus are easily adapted to algorithms for segment or point intersections. Plane-polyhedron algorithms are based on projection techniques and are of necessity detection algorithms.

3.1. The two dimensional case

In deriving line-polygon intersection algorithms, it is sufficient to find an intersection point at any level of the hierarchy of an inner representation or separation information at any level for an outer representation. Lemma 2.1 is applied to actually determine the intersection.. Using an inner representation, the initial test for intersection with P_1 requires only a constant number of operations to determine the vertex of P_1 closest to the line. P_1 is now grown towards P_2 by including the neighbors of this nearest vertex. Next, the relevant sections of P_2 are tested against the line and the process is repeated. At each level, the polygon grows by only a constant number of vertices (the nearest neighbors of the previous closest vertex) and the iteration requires a constant number of operations yielding:

Theorem 3.1: Given a polygon P of n vertices and a line L, O(log n) operations suffice to compute the intersection of L and P.

Corollary: Given a polygon P of n vertices and a line segment S, O(log n) operations suffice to compute the intersection of S and P.

Corollary: Given a polygon P of n vertices and a point R, O(log n) operations suffice to compute the intersection of R and P.

A dual of the above argument could have been applied to an outer polygonal representation of P yielding the same result.

3.2. The three dimensional case

The line-polyhedron intersection problem may be solved by a technique similar to the line-polygon intersection algorithm. This method easily extends to both inner and outer polyhedral representations through the use of Lemma 2.1. An alternative approach is to consider the projections of the line and polyhedron onto a plane normal to the line. The resulting point lies in the resulting polygon if and only if the line and polyhedron intersect. Having detected an intersection, O(log n) operations suffice to determine the points on the boundary of the polyhedron which lie on the line. Lemma 2.2 shows that the hierarchical description of the polyhedron also gives a hierarchical description of the projection of the polyhedron onto a plane. These techniques yield:

Theorem 3.2: Given a polyhedron P of n vertices and a line L, O(log n) operations suffice to compute the intersection of L and P.

Corollary: Given a polyhedron P of n vertices and a line segment S, O(log n) operations suffice to compute the intersection of S and P.

Corollary: Given a polyhedron P of n vertices and a point R, O(log n) operations suffice to compute the intersection of R and P.

A variant of the last technique solves the plane-polyhedron intersection problem. Here, only an intersection detector is possible since the description of the intersection may require O(n) operations. Projecting the plane and polyhedron onto a plane normal to the plane yields a line and polyhedron which intersect if and only if the original plane and polyhedron intersect. Once again, Lemma 2.2 gives a method for finding the hierarchical description of the

2.2. The three dimensional case

In three dimensions, the intuition is the same as that presented above. Inner representations now involve growing faces out of each existing face and outer representations involve decreasing size by adding intermediate bounding halfspaces. Letting V(P) and F(P) represent the vertices and faces of polyhedron P, the details are as follows:

Definition: An *inner polyhedral representation* of P is an ascending chain P_i ,i=1,..,k of polyhedra where P_{i+1} is formed from P_i as follows: For each face (u,v,w) $\in F(P_i)$, either (u,v,w) $\in F(P_{i+1})$ or there is a vertex x $\in V(P_{i+1})-V(P_i)$ with (u,v,x), (v,w,x), and (w,u,x) all faces of P_{i+1}.

Definition: An *outer polyhedral representation* of P is a descending chain P_i i=1,...,k of polyhedra where P_{i+1} is formed from P_i as follows: If $P_i = \bigcap_{j=1}^{s_i} H_j$ then $P_{i+1} = \bigcap_{j=1}^{s_i} H'_j$ where either $H'_j = H_j$ or H'_j is all of three space. Further, if H_{j_1} and H_{j_2} are adjacent in P_i, then not both of H'_{j_1} and H'_{j_2} are all of three space. And, if H'_j is all of three space, then H_j is adjacent to at most some constant number of half-spaces in P_i.

To form P_i from P_{i+1} in an inner representation remove an independent set of low degree and form the convex hull of the remaining vertices. This computation requires linear time and gives a representation of linear space and O(log\ n) height. Dually, in an outer representation, form P_i from P_{i+1} by removing an independent set of bounding half-spaces of low degree. Again, a representation requiring space O(n) and having height k = O(log n) results after O(n) computation.

2.3. Basic properties of inner and outer representations

Inner and outer representations are useful to intersection problems because of their shallow (O(log n)) depth, ease of creation (linear time) and local nature. When the area of a potential intersection has been identified, it is possible to use either of the representations to grow the polygon or polyhedron locally within that region in a constant number of operations per iteration. This property are captured as follows:

Lemma 2.1: Let $p_i(d)$ be the maximal vertex of P_i in the direction d where P_i is the ith member of a hierarchy for an inner or outer representation for a polygon or polyhedron. Then, either $p_{i+1}(d) = p_i(d)$ or $p_{i+1}(d)$ is one of the new neighbors of $p_i(d)$ in P_{i+1}.

Proof: The result in all cases follows from the convexity of each P_i. In the case of an inner polygonal representation, observe that a tangent line in direction d passing through $p_i(d)$ divides the plane into two halfplanes with P_i lying strictly within one of the halfplanes. If added vertices lie within the other halfplane and are not adjacent to $p_i(d)$, the resulting polygon cannot be convex. similar contradictions yield the same result in all other cases. ▪

Lemma 2.2: If Q is any plane and if $P_1,...,P_k$ is an inner (resp. outer) polyhedral representation of P, then $P_1 \cap Q,...,P_k \cap Q$ is an inner (resp. outer) polygonal representation of $P \cap Q$.

Proof: The convexity of P and P_i for all i, shows that the $P_1 \cap Q,...,P_k \cap Q$ and $P \cap Q$ are all convex. Convexity also guarantees that the $P_i \cap Q$ grow (or shrink) appropriately. ▪

3. Static-Dynamic Intersection Methods

Hierarchical representations are used to derive O(log n) algorithms for intersecting polyhedra with linear subspaces of various dimensions. The point-in-polygon and point-in-polyhedron results were previously known but all others improve previous results[CD].

Algorithms are given for the line-polygon intersection problem in the plane and the line-polyhedron and plane-polyhedron problems in 3 dimensions. Intersection problems involving projected polygon and Theorem 3.1 yields:

Theorem 3.3: Given a polyhedron P of n vertices and a plane R, O(log n) operations suffice to compute the intersection of R and P.

4. Dynamic-Dynamic Intersection Methods

4.1. The two dimensional case

Intersection problems involving two hierarchically described objects are solved by dynamic-dynamic methods. The presentation of the two dimensional case simplifies that of [CD] and sets idea for the 3 dimensional case. A *monotone polygonal chain* **(MPC)** is defined to be a sequence of vertices and edges of a convex polygon given in order of increasing y-coordinate. By convexity, an MPC will either be left-oriented or right-oriented. Semi-infinite rays called *endedges* are attached to the beginning and end of the MPC. These edges run parallel to the x-axis towards *+inf* if right-oriented or *−inf* if left-oriented. O(log n) operations suffice to decompose a convex polygon P into MPC P_L and P_R with the vertices of P_L (resp. P_R) given in clockwise (resp. counter-clockwise) order. This decomposition can be done in any coordinate system. and has the property that P = $.P_L \cap P_R$ and $P \sqsubseteq P_L, P_R$. In higher dimensions, extensions of this decomposition method simplify algorithm presentations via the following:

Lemma 4.1: Convex polygons P and Q intersect if and only if P_L and Q_R intersect and P_R and Q_L intersect.

Proof: If P and Q intersect, then since $P \sqsubseteq P_L, P_R$ and $Q \sqsubseteq Q_L, Q_R$, it is obvious that P_L and Q_R intersect and P_R and Q_L intersect.

If P and Q do not intersect, then P_L must be strictly to the right of Q_R or P_R must be strictly to the left of Q_L. Since the finite parts of each of these polygonal chains do not intersect and the semi-infinite parts grow away from each other, no intersection can take place. ▪

Given this reduction, it remains to present an algorithm for intersecting chains. The algorithm involves a generalization of binary search. At each iteration, an edge of each chain is selected and extended infinitely in each direction. The intersection of these supporting lines gives information (based on the properties of MPC) which allows half of the edges of one (or both) polygons to be ignored without missing the detection of an intersection. Edges are not eliminated, but the structural information they provide is discarded and a new endedge is introduced preserving the MPC properties. A simple case analysis shows that the newly formed chains intersect if and only if the original chains did.

Let R (resp. L) be a right (resp left) MPC with edges $r_1, r_2, \ldots r_m$ (resp. $l_1, l_2, \ldots l_n$). The edges r_1, r_m, l_1 and l_n are now rays and all other edges are finite. Let i = m/2 and j = n/2, and consider the four regions formed by the intersection of the lines R_i and L_j supporting the edges r_i and l_j. R and L can each exist in only two of these regions. Further, L and R can only coexist in one of the four regions. Label the regions as the R-region, the L-region, the LR-region and the empty region as shown in Figure 2. New MPCs R' (resp. R'') are defined to be R with the edges above (resp. below) r_i replaced by the semi-infinite ray parallel to the x-axis and intersecting r_i at its vertex. L' and L'' are defined from L in an analogous manner. The algorithm relies on the following:

Lemma 4.2: If the lines R_i and L_j intersect and the segments r_i and l_j do not, then if the LR-region is above the empty region (i.e. seeks $+\infty$ in the y-direction)

i) If the upper endpoint of r_i does not lie in the LR-region, then R intersects L if and only if R'' intersects L.

ii) If the upper endpoint of l_j does not lie in the LR-region, then R intersects L if and only if R intersects L''.

iii) If both endpoints of r_i and l_j lie in the LR-region and the lower endpoint of r_i has smaller (resp. larger) y-coordinate than the lower endpoint of l_j, then R intersects L if and only if R'' intersects L.

Proof: (See Figure 2) In case i), since the upper endpoint of r_i does not lie in the LR region, all points of R above r_i lie in that region by convexity. A similar argument handles case ii).

In case iii), if the lower endpoint of r_i has smaller y-coordinate than the lower endpoint of l_j, then the lower part of R cannot intersect the upper part of L. The lower part of R can never intersect the lower part of L twice and the upper part of L can never intersect the upper part of R twice. Therefore, either the intersection is exactly a vertex or edge or the upper part of R must be involved. If the intersection is restricted to the boundary, it must involve the upper part of R, hence R'' must intersect L. ▪

The extension to the case where the LR-region lies below the empty region yields:

Theorem 4.3: Given two polygons, O(log n) operations suffice to generate either

a) A point common to both polygons
or

b) A line supporting an edge of one polygon which separates the two polygons

Proof: In a constant number of operations, half of one of the two chains, L or R can be eliminated without changing the intersection status of the reduced problem. To achieve this, the algorithm first considers the middle edges r_i and l_j and their supporting lines R_i and L_j. If R_i and L_j do not intersect, two cases arise depending on whether L_j is to the left or right of R_i. In the first case, there is no intersection and R_i and L_j are separating lines. In the second, replacing i by i+1 yields a situation in which R_i and L_j cannot be parallel, so the algorithm proceeds. If R_i and L_j intersect and r_i and l_j also intersect, then a point of intersection has been found. Finally, the two remaining cases handling different orientations of intersecting lines R_i and L_j are considered in Lemma 4.2.

The algorithm will eventually reduce one of the chains to a wedge of two edges. At this point, it is sufficient to apply an extension of the segment-polygon intersection detector given in [CD] to find a point of intersection or separating edge. The two intersection tests need not report the same point of intersection. If neither of the reported points belongs to both of the polygons, it must be the case that one belongs to each. In this case, a point belonging to both can be easily found. ▪

This theorem guarantees a separating line which is an extension of an edge of one of the polygons. While this is unnecessary here, it proves crucial in the three dimensional case.

4.2. The three dimensional case

4.2.1. Methods of preprocessing polyhedra

The discussion of 2-dimensional objects ignored representational issues since any representation of a convex polygon in any coordinate system was suitable. This was true because polygons are essentially 1-dimensional manifolds and chains can be represented as (piecewise) 1-dimensional objects. Similarly, 3 dimensional polyhedra can be represented as 2-dimensional manifolds as planar subdivisions. Unfortunately, no known techniques reduce this subdivision to a 1-dimensional manifold to which simple ordering properties might be applied. A 3-dimensional polyhedron will be viewed as a sequence of cross-sections each of which is a polygon. Appropriate choices of cross-sections allow convexity to play a key role in the algorithms given here. For any representation of a polyhedron in an xyz coordinate system, consider x,y cross-sections corresponding to the z-values of all its vertices. These cross sections together with the edges joining adjacent cross-sections then give a characterization of the complete polyhedron. A *drum* is defined as 2 adjacent cross-sections along with all of their connecting edges. In this representation, a polyhedron of n vertices, might be decomposed into as many as n-1 drums.

The drum representation of a polyhedron has some useful properties. Even though a drum represents a 3-dimensional piece of a 3-dimensional object, there is no freedom of motion in passing from the bottom to the top of a drum. This motions consist of travel along single edges on which no vertices lie. The simplicity of this motion allows the view of a drum as a continuous transformation from its bottom face to its top face along the connecting edges. Thus in a sense, drums are $2\frac{1}{2}$ dimensional objects, lying between polygons and polyhedra. This representation allows algorithms which work for polygons to be modified to work on drums.

The space and time requirements of the drum representation are unfortunate. A polyhedron might be decomposed into O(n) drums each requiring O(n) space for its description. So, $O(n^2)$ space and time might be necessary for generating and storing this representation. These bounds are unsatisfying in light of other representations requiring only linear space from which intersections may be computed in O(n log n) time. Recent work has provided a first step towards circumventing this difficulty. In [DM], a method is given which requires O(n log n) preprocessing time and O(n log n) storage for representing the drum decomposition of a polyhedron. Since this method might represent as much as $O(n^2)$ information, it is not possible to store information in a random access fashion. Rather, $O(\log^2 n)$ operations are required to retrieve specific information about particular aspects (e.g. edges, vertices or faces) of particular drums. O(log n) operations at each iteration are sufficient to give the information necessary to the detection algorithms given here.

In the algorithms given below, preprocessing is assumed which makes available in a random-access fashion, all the necessary information about a polyhedron. Any time bounds which take advantage of this storage scheme must be multiplied by O(log n) if the O(n log n) space and time preprocessing of [DM] is used. When considering 2 polyhedra, it is *not* assumed that each has been preprocessed in the same xyz coordinate system. Thus, the representation is robust being invariant under the translation, rotation and scaling of objects.

4.2.2. Detecting drum-drum intersections

A drum-drum intersection detector forms the core of the polyhedron-polyhedron intersection detector. Separation information for 2 non-intersecting drums is used to remove half of one polyhedron from consideration in the polyhedron-polyhedron intersection algorithm. Thus, polyhedron-polyhedron intersection problems are reduced to O(log n) drum-drum intersection problems.

Drum-drum intersections are detected by generalizations of the techniques used to detect polygon-polygon intersections. The structure of a drum as the continuous transformation of its bottom into its top is crucial. However, the change to 3 dimensions adds complexity to the analysis which resolved the polygon-polygon intersection problem. To set ideas, consider first the problem of detecting polygon-drum intersections.

Let P be a polygon and Q a drum. If R is the intersection of the plane of P with Q, then P and Q intersect if and only if P and R intersect. Determining the vertices and edges of R explicitly requires a linear number of operations. Therefore, R is considered as an implicitly specified object. The polygon-polygon intersection algorithm is used to detect the intersection of P and R. Additional computation is done each time an edge of R is needed. R is described as a clockwise sequence of vertices consisting of 2 (or possibly 1 or 0) vertices from the intersection of the plane and the top of the drum, followed by vertices derived from intersections of the plane and consecutive edges connecting the top and bottom faces of the drum, followed by 2 (or 1 or 0) vertices from the intersection of the plane and the bottom of the drum and finally consisting of vertices derived from intersections of the plane and consecutive edges connecting the bottom and top faces of the drum. Since the representation is presented in no more than four components, the needed edges of R can be found in a constant number of operations. Thus, intersecting a drum and a polygon is as easy (after O(log n) operations) as intersecting two polygons leading to

Theorem 4.4: Given a drum and a polygon, O(log n) operations suffice to compute either

a) A point common to both
or

b) A line supporting an edge of the polygon or a plane supporting a face (or top or bottom) of the drum (or both) which separates the two objects.

Proof: To begin, an implicit representation for R is found in O(log n) operations. From this representation, desired vertices of R can be found in a constant number of operations. Since, R and P are coplanar, by construction, the algorithm of Theorem 4.3 yields the result. ▪

For the problem of detecting drum-drum intersections 2½ dimensional analogs of polygon-polygon intersection detectors are used. Each drum is decomposed into left and right halves relative to the plane formed by the normals to the tops of the two *drums*[1]. Conceptually this division is done by shining a beam of light in the direction of the normal to this plane starting at $+\infty$ (resp. $-\infty$) to define the right (resp. left) half drum. All faces lit by this light (consisting of those having positive component of their normals in this direction) belong to the relevant half drum. These halfdrums are then made semi-infinite by adding endfaces perpendicular to the drum top and extending towards $+\infty$ *or* $-\infty$. For a drum D, this decomposition into left and right halfdrums D_L and D_R satisfies again the properties that $D = D_R \cap D_L$ and $D \sqsubseteq D_L, D_R$. Using these results, it is easy to verify that

Lemma 4.5: If D and E are drums which have been decomposed into left and right halves as described above, then $D \cap E$ iff $D_L \cap E_R$ and $D_R \cap E_L$.

Proof: If $D \cap E$, then since $D \sqsubseteq D_L$, D_R and $E \sqsubseteq E_L$, E_R, it is obvious that $D_L \cap E_R$ and $D_R \cap E_L$.

If D and E do not intersect, assume without loss of generality that there is a face of D which forms a separating plane between D and E. Assume that this face belongs to D_L (the case of D_R following in an obvious manner). Then, D must lie to the left of this face and E to its right (with left and right defined relative to the decomposition of the drums into halfdrums. So, any extension of E to the right cannot intersect this plane and hence cannot intersect D_L. Therefore, D_L and E_R cannot intersect. ▪

1 In the case where the two drum tops are parallel any plane including the normal to the drum tops will suffice. In this case, the problem is first reduced (in constant time) to one in which drum tops and bottoms are (pairwise) coplanar. This will have no effect on running times and will make the algorithms avoid unnecessary work.

Given this reduction, it remains to generalize the polygon algorithm to the case of halfdrums. The middle face of each halfdrum is selected and extended infinitely in all directions. The intersection of these supporting planes then gives information (based on the properties of halfdrums) which allows the identification of that half of the faces of one drum which can be ignored without missing the detection of an intersection. Faces are not eliminated, but the structural information they provide is discarded and an endface is created as a semi-infinite slab preserving the halfdrum properties. A simple case analysis shows that the newly formed halfdrums intersect if and only if the original drums did.

To set notation, consider a right halfdrum R and a left halfdrum L with faces $r_1,r_2,...r_m$ and $l_1,l_2,...l_n$ respectively. Recall that in these representations, the endfaces r_1,r_m,l_1 and l_n are semi-infinite and all other faces are finite. Let i = m/2 and j = n/2, and consider the four regions formed by the intersection of the planes R_i and L_j supporting the faces r_i and l_j. Again, R and L can each exist in only two of these regions. L and R can only coexist in one of the four regions. The regions are labeled as the R-region, the L-region, the LR-region analogous to the planar regions shown in Figure 2. The halfdrums R' (resp. R") are defined as R with the faces beyond (resp. before) r_i replaced by the semi-infinite endface of extension of r_i. L' and L" are defined from L in an analogous fashion.

Lemma 4.6: If the planes R_i and L_j intersect and the faces r_i and l_j do not and the LR-region is above the empty region (i.e. seeks $+\infty$) then

i) If the upper edge of r_i does not lie in the LR-region, then R intersects L if and only if R" intersects L.

ii) If the upper edge of l_j does not lie in the LR-region, then R intersects L if and only if R intersects L".

iii) If all edges of r_i and l_j lie in the LR-region and the lower edge of r_i has a smaller (resp. larger) normal than the lower edge of l_j, then R intersects L if and only if R" intersects L.

Proof: (Shown in projection in Figure 2) In case i), since the upper edge of r_i does not lie in the LR region, all points of R above r_i lie in that region by convexity. A similar argument handles case ii).

In case iii), if the lower edge of r_i has smaller normal than the lower edge of l_j, then the lower part of R cannot intersect the upper part of L. As always, the lower part of R cannot intersect the lower part of L twice and the upper part of L cannot intersect the upper part of R twice. Since an intersection must involve two "punctures" or be restricted to the boundary (in which case it must involve R"), the problem reduces to detecting the intersection of R" and L. ▪

This theorem suggests immediately an algorithm for detecting drum-drum intersections in O(log n) operations. r_i and l_j are considered and R_i and L_j are formed yielding the four regions L,R, LR and empty. If l_i and r_j intersect, the algorithm reports an intersection and halts. If L_i and R_j are parallel, one of two situations results. If there can be no intersection (i.e. L_i and R_j are separating planes), the algorithm reports so and halts. Otherwise, i is set to i+1 and the algorithm continues. If none of these cases result, it must be the case that the four regions exist in a configuration like those shown in projection in Figure 2 or a similar configuration with the empty region above the LR-region. In the former case, the results of Lemma 4.6 give us a method of removing half of one drum from consideration in O(log n) operations. In the latter case, an obvious analog of Lemma 4.6 gives the same result. This leads to:

Theorem 4.7: Given two preprocessed drums, O(log n) operations suffice to determine either

a) A point common to both
 or

b) A plane supporting a face or edge of one of the drums which separates the two drums.

4.2.3. Detecting polyhedral intersections

Finally, there remains the extension to polyhedral-polyhedral intersection problems. The algorithm of the previous section could be easily extended to the problem of detecting drum-polyhedron intersections. In that case, the drum is first compared to the middle drum of the *polyhedron*[2]. If these drums intersect, it is reported and the algorithm halts. If not, the result of Theorem 4.7 gives a separating plane supporting one of the drums. If it supports the drum belonging to the polyhedron, then it also separates the polyhedron from the drum. If it supports the separate drum, then one of three cases results. If it does not intersect the polyhedron, it acts as a separating plane and there can be no intersection. If it intersects the polyhedron above its middle drum, then the bottom part (lower half of its drums) of the polyhedron can be eliminated from further consideration of intersections. Similarly, if it intersects the polyhedron below its middle drum, the upper half of the polyhedron is eliminated from further consideration. Convexity guarantees that a plane cannot intersect the polyhedron both above and below its middle drum without intersecting the middle drum. This fact forms the basis of the algorithm which follows.

In considering polyhedron-polyhedron intersection problems, it is worthwhile to set some notation. The *waist* of a polyhedron is its middle drum. The *cone* of a drum of a polyhedron is formed by extending all its faces infinitely in both directions and computing their intersection. The cone, which may or may not be closed, is the largest convex polyhedra for the given drum. It is the polyhedron formed as the intersection of the halfspaces defined by the infinite extensions of the faces of the drum. Therefore, any polyhedron having this drum as its waist must be contained in its cone. However, the waist of the cone is exactly the drum which generated the cone. Therefore, if two drums do not intersect, their cones cannot intersect both above and below the drums. This fact is used to eliminate half of a polyhedron from consideration in intersection detection problems. leading to the result:

Theorem 4.8: Given two preprocessed polyhedra P and Q of p and q vertices respectively, $O(\log^2(p+q))$ operations suffice to determine either

a) A point common to both
 or

b) A plane supporting a face or edge of one of the polyhedra and separating them.

Proof: The proof follows from a method of dividing the number of drums of one of the polyhedra in half in O(log (p+q)) operations. The resultant problem is shown to have the same form. Let E be the waist of P, F be the waist of Q, A be the cone of E and B the cone of F as shown in Figure 3. The algorithm of Theorem 4.7 is used to detect whether E and F intersect. If they do, the algorithm exits in case a of this theorem. If not, a plane T is found which is an extension of a face or edge of E (without loss of generality) and hence P and separates E from F. Two cases now result. If T is an extension of a face or face-edge of E, T must also separate P

[2] If the preprocessing direction of the polyhedron is parallel to the top of the drum some difficulties result. This is resolved by doing (in O(log n) operations) a binary search to eliminate all drums of the polyhedron except those which could possibly intersect the drum (i.e. occur in the range of values between the drum top and bottom).

from F. In this case, the ideas from the drum-polyhedron intersection detector eliminate half of F from further consideration. The case where T is an extension of the top or bottom of E (or of an edge defining the top or bottom) is more complex.

Assume without loss of generality that T is an extension of the top of E (all other cases being similar). Now, since T separates E from F, F must lie "above" E. A and F intersect because otherwise a separating plane which was an extension of a face or face-edge of E would have been found. Therefore, F must intersect A above E. Now since F and A intersect above E, A and B also intersect above E. Observe that faces of A and B cannot intersect below E by convexity. Therefore, the bottom of A (and hence the bottom of P) can be eliminated from further intersections. ▪

5 Conclusions and possible extensions

A methodology for studying polyhedral intersection detection algorithms has been presented. The benefits of the methodology are twofold, providing a cleaner presentation of intersection algorithms and improving known results for these problems. There remain many open problems.

The techniques used to state and prove these results in three dimensions differ very little from those used in two dimensions. This suggests the possibility of extending these algorithms to arbitrary dimensions and achieving $O((d \log n)^2)$ as a time bound for intersection detection in d dimensions. There also remains open the problem of determining whether three (or more) polyhedra have a point in common. Were it possible to achieve both of these extensions, it might be possible to produce a sub-exponential algorithm for linear programming having a form different. from the ellipsoid algorithm.

There also remain the practical issues of implementing the algorithms presented here with the goal of achieving improved methods for hidden surface elimination.

Acknowledgement

We would like to thank Dan Field whose comments helped make the final presentation of this paper more coherent. We also acknowledge his help in identifying a bug in the original presentation of Lemma 4.1

6. References

[CD] B. Chazelle and D. Dobkin, Detection is easier than computation, ACM Symposium on Theory of Computing, Los Angeles, Ca, May, 1980,146-153.

[Da] G. B. Dantzig, *Linear Programming and its Extensions*, Princeton University Press, Princeton, NJ, 1963.

[DM] D. P. Dobkin and J. I. Munro, Efficient uses of the past, 21st Annual Symposium on Foundations of Computer Science, Syracuse, NY, October, 1980, 200-206.

[K] D. G. Kirkpatrick, Optimal search in planar subdivisions, detailed abstract, Univ. of British Columbia, Vancouver, B.C., Canada, 1980.

[MP] D. Muller and F. Preparata, Finding the intersection of 2 convex polyhedra, Technical Report, University of Illinois, Oct., 1977.

[Sh] M. Shamos, *Computational Geometry*, PhD Thesis, Yale U., May, 1978.

[War] J. E. Warnock, A hidden-surface algorithm for computer generated half-tone pictures, University of Utah Computer Science Department, TR 4-15, 1969.

[Wat] G. S. Watkins, A real-time visible surface algorithm, University of Utah Computer Science Department, UTEC-CSc-70-101, June, 1970.

[Wi] P. H. Winston, *The Psychology of Computer Vision*, McGraw Hill, New York, 1975.

Figure 1: The polygon P = ABCDEF has inner representation $\{P_1, P_2\}$ where $P_1 = ACE, P_2 = P$ and outer representation $\{Q_1, Q_2\}$ where $Q_1 = XYZ, Q_2 = P$

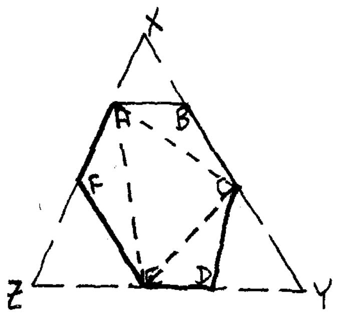


Figure 2: Regions involved in testing for polygonal intersections.

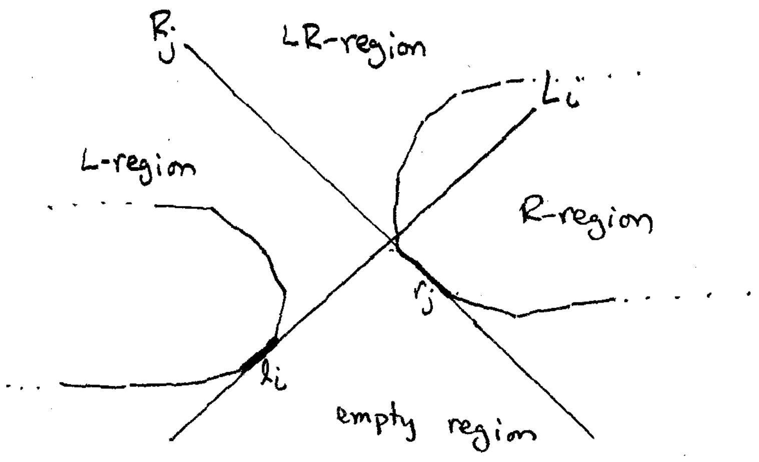
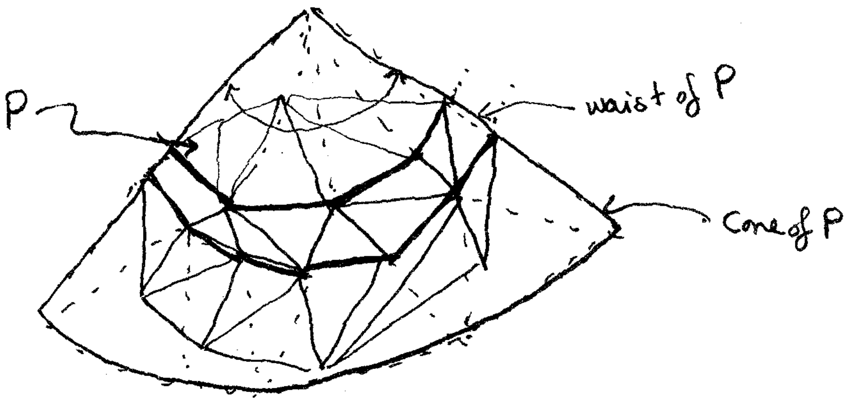


Figure 3: A polyhedron P with its waist and cone.


P
waist of P
Cone of P