Theoretical Computer Science 27 (1983) 241-253
North-Holland

FAST DETECTION OF POLYHEDRAL INTERSECTION

David P. DOBKIN*
Electrical Engineering and Computer Science Department, Princeton University, Princeton, NJ 08544, U.S.A.

David G. KIRKPATRICK
Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada

Abstract. Methods are given for unifying and extending previous work on detecting intersections of suitably preprocessed polyhedra. 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 [2], hidden surface elimination [9, 10] and computer vision [11].

In a previous paper [1] the detection problem for suitably preprocessed polyhedral intersections was distinguished from the computation problem and shown to be of lower complexity. Given two polyhedra, the detection problem asks only for a witness to the intersection or non-intersection rather than a computation of the entire intersection. Furthermore, it is allowed to preprocess separately each of the polyhedra, as long as the preprocessing is independent of any underlying coordinate system. The witness might consist of a common point in the case of an intersection or a separating line or plane in the case of no intersection. Because of the preprocessing and reduced output size, solutions of complexity $c \log^k n$ (for fixed constants c and k) are possible for the detection problem while linear lower bounds are known on the computation problems [8, 5].

The results of Chazelle and Dobkin [1] are unified and extended here. This is done by combining the method of dynamically defining convex polyhedra from [4] and the method of shadowing from [7]. 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

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

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.

The results of this paper are derived by building from two-dimensional polygons to three-dimensional polyhedra. As an intermediate step, drums are defined. A drum is a three-dimensional object of restrict form. In particular, all vertices of a drum are required to be the vertices of two convex polygons having parallel planes of support. In our applications, drums are derived from consecutive cross-sections of a polyhedron taken in a fixed direction. Our use of these properties of drums suggests that they are in fact $2\frac{1}{2}$-dimensional objects and we use them as such. All objects are defined to consist of boundary and interior. Were this not the case, linear lower bounds would exist (see [1]).

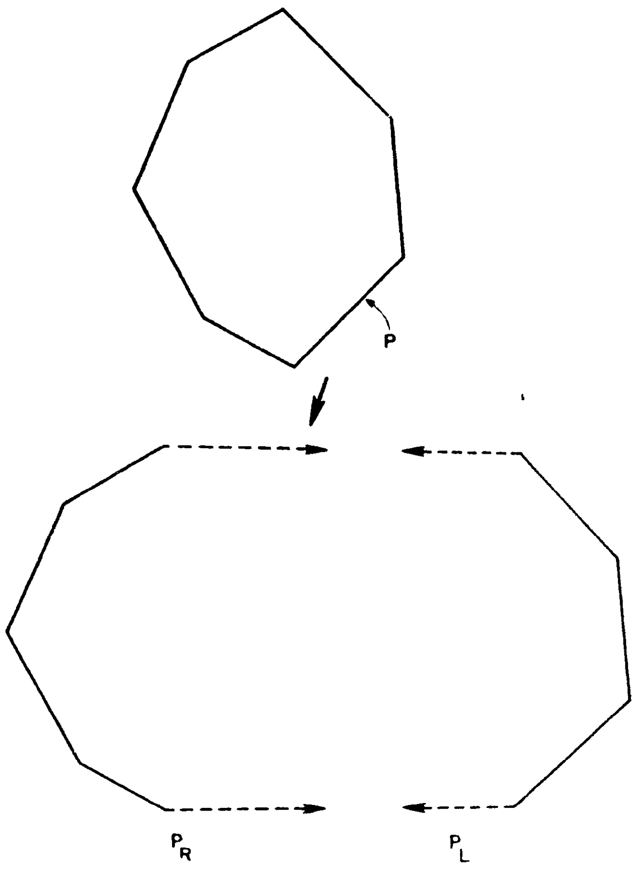
The efficiency of our algorithms is achieved by balancing the complexity of the intersection calculation against the number of iterations which must be performed to be certain of completely considering the relevant objects. In the case of two-dimensional polygons and $2\frac{1}{2}$-dimensional drums, $O(\log n)$ iterations are performed each requiring $O(1)$ operations. Three-dimensional polyhedra require $O(\log n)$ iterations of the $O(\log n)$ operation drum intersection algorithm.

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

2. Detecting two-dimensional intersections

Intersection problems involving two hierarchically described objects are solved by dynamically growing the objects. The presentation of the two-dimensional case simplifies that of [1] and sets ideas for the three-dimensional case. A polygonal boundary is decomposed into two monotone polygonal chains of edges by cutting at its highest and lowest y coordinates (see Fig. 1).[1] This yields two sequences of vertices and edges in order of increasing y-coordinate. By convexity, such chains will either be left- or right-oriented. Semi-infinite rays are attached to the beginning and end of each chain and the interior is included to form two *monotone polygonal sectors* (MPS). These edges run parallel to the x-axis towards $+\infty$ if right-oriented or $-\infty$ if left-oriented and define the area contained by the MPS. $O(\log n)$ operations suffice to decompose a convex polygon P into MPS 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 without preprocessing by finding the vertices of maximum and minimum y-coordinate in that system. The decomposition into MPS has the property that $P = P_L \cap P_R$ and $P \subseteq P_L$, P_R. In higher dimensions, extensions of this decomposition method simplify algorithm presentations via the following lemma.

[1] Ties resulting from horizontal edges may be broken arbitrarily since there can be no more than 2 collinear vertices.



Fi· 1. Decomposing a polygon into left and right MPS.

Lemma 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_R intersect and P_R and Q_L intersect.

If P and Q do not intersect, then the supporting line of an edge of one polygon (say P) is separating. If the supported edge belongs to P_L, it also separates P_L from Q and P_L from Q_R.

Given this reduction, it remains to present an algorithm for intersecting vertex 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 MPS) 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 MPS 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) MPS 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 $i = \lceil \frac{1}{2}m \rceil$ and $j = \lceil \frac{1}{2}n \rceil$, 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 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 Fig. 2. New MPS R' (resp. R'') are defined to be R with the area defined by edges above (resp. below) r_i sliced 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.

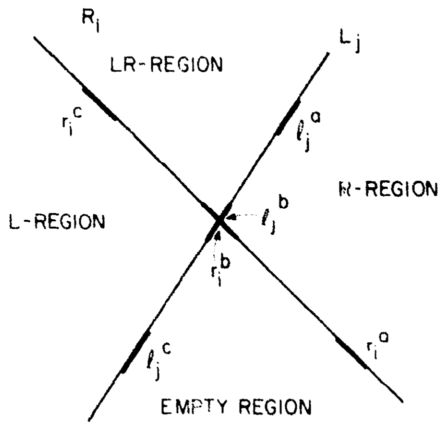


Fig. 2. Regions involved in tests for polygonal intersection—r_i (resp. l_j) can exist in one of the three positions r_i^a, r_i^b, r_i^c (resp. l_j^a, l_j^b, l_j^c) on r_i (resp. l_j).

Lemma 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 Fig. 3). 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

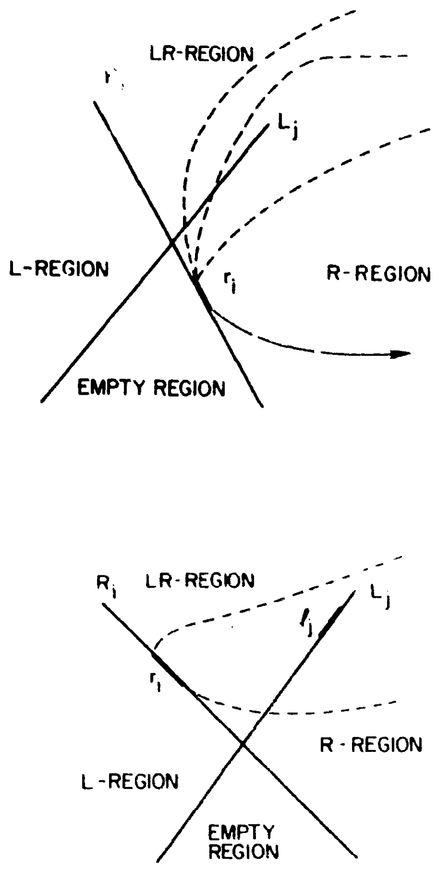


Fig. 3. Cases (i) and (ii) in the proof of Lemma 2. (a) Upper endpoint of r_i lies in the R-region, dashed paths represent possible R'' trajectories. (b) Upper endpoints of both r_i and l_j lie in the LR-region and lower endpoint of r_i lies below lower endpoints of l_j. If L and R intersect, then L and R'' must intersect.

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 the following theorem.

Theorem 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 are parallel, 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 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 [1] to find a point of intersection or separating edge.

Note that the two intersection tests need not report the same point of intersection. However, if both tests report an intersection and return a point, an intersection point can be found. If neither of the reported points belongs to both of the polygons, it must be the case that one belongs to each.

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.

3. Detecting three-dimensional intersections

3.1. Methods of preprocessing polyhedra

The discussion of two-dimensional objects ignored representational issues since any representation of a convex polygon which allowed for binary searching was suitable. And, no constraints were put on the coordinate system. This was true because polygons are essentially one-dimensional manifolds and edge-chains can be represented as (piecewise) one-dimensional objects. Similarly, three-dimensional polyhedra can be represented as two-dimensional manifolds or as planar subdivisions. Unfortunately, no known techniques reduce this subdivision to a one-dimensional manifold to which simple ordering properties might be applied. A three-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 (see Fig. 4).

The drum representation of a polyhedron has some useful properties. Even though a drum represents a three-dimensional piece of a three-dimensional object, there is no freedom of motion in passing from the bottom to the top of a drum. This motion consists 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

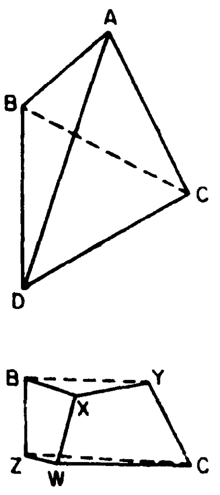


Fig. 4. One of the drums in the decomposition of a tetrahedron. Note that the vertices X, Y, Z, W are created when forming cross-sections.

from its bottom face to its top face along the connecting edges. Thus, in a sense, drums are two-dimensional objects, lying between polygons and polyhedra. This representation allows algorithms which work on 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 [3] 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.[2] 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 [3] 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.

[2] Actually, Overmars [6] has improved upon the bounds given here.

3.2. *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 three 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 the following theorem.

Theorem 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 3 yields the result. ∎

For the problem of detecting drum–drum intersections two-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*.[3] 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) halfdrum. All faces lit by this light (consisting of those having positive component of their normals in this direction) belong to the relevant halfdrum. 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 $D \subseteq D_L, D_R$. Using these results, it is easy to verify the following lemma.

Lemma 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 $D \subseteq D_L, D_R$ and $E \subseteq E_L, E_R$, 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.

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 $r_1, r_2, \ldots, r_m$ and $l_1, l_2, \ldots, l_n$ 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 $i = \lceil \frac{1}{2}m \rceil$ and $j = \lceil \frac{1}{2}n \rceil$, 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 exist in only two of these regions. L and R can only coexist in one of the four regions. The regions

[3] 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 contrast 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.

are labeled as the R-region, the L-region, the LR-region analogous to the planar regions shown in Fig. 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 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 Fig. 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 immediately suggests 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_j and r_i 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 Fig. 3 or a similar configuration with the empty region above the LR-region. In the former case, the results of Lemma 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 6 gives the same result. This leads to the following theorem.

Theorem 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.*

3.3. *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.*[4] If these drums intersect, it is reported and the algorithm halts. If not, the result of Theorem 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 depending on the intersection of this plane and the polyhedron. This intersection need not be computed. It is sufficient to know the maximal and minimal vertex values in the direction normal to this plane. These values may be computed in $O(\log n)$ operations from the preprocessed polyhedron. If the plane 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 (see Fig. 5). 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 to their supporting planes and computing this intersection of half-spaces. The cone, which may or may not be closed, is the largest convex polyhedra which might enclose the given 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 following result.

Theorem 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

[4] 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).

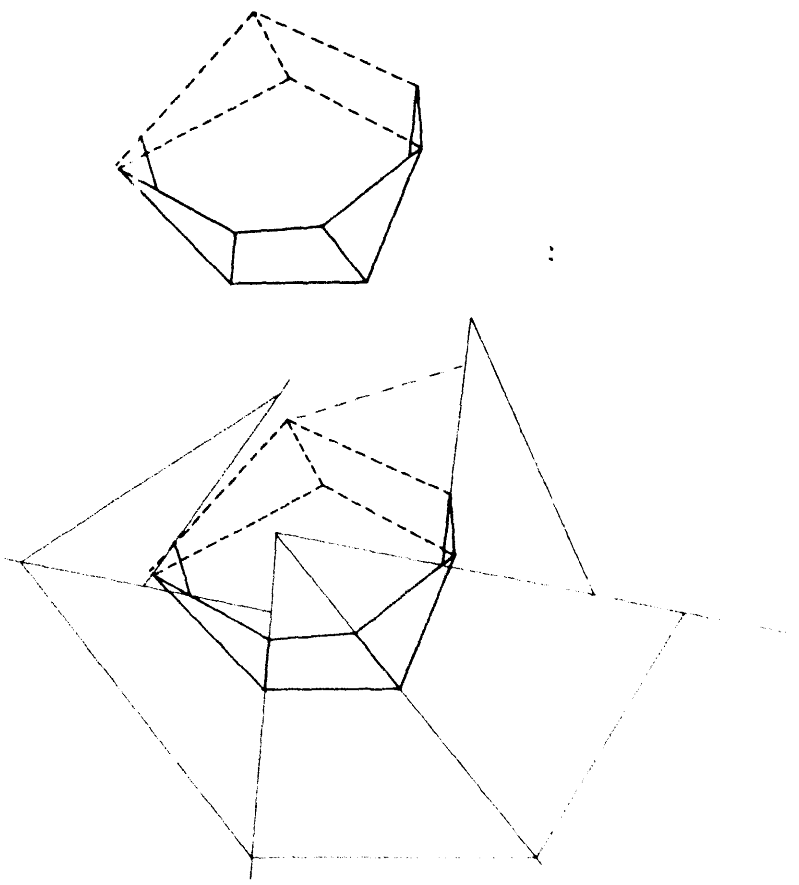


Fig. 5. A drum and the faces constituting the front of its cone.

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 Fig. 4. The algorithm of Theorem 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 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.

4. 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 $O((d \log n)$ 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.

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

Acknowledgment

We would like to thank Dan Field whose comments helped in making the final presentation of this paper more coherent. We also acknowledge his help in identifying a bug in the original presentation of Lemma 1. We thank an anonymous referee for comments leading to a more coherent final presentation.

References

[1] B. Chazelle and D. Dobkin, Detection is easier than computation, *ACM Symp. on Theory of Computing*, Los Angeles, CA (1980) 146–153.
[2] G.B. Dantzig, *Linear Programming and Its Extensions* (Princeton University Press, Princeton, NJ, 1963).
[3] D.P. Dobkin and J.I. Munro, Efficient uses of the past, *21st Ann. Symp. on Foundations of Computer Science*, Syracuse, NY (1980) 200–206.
[4] D.G. Kirkpatrick, Optimal search in planar subdivisions (detailed abstract), Univ. of British Columbia, Vancouver, BC, Canada, 1980.
[5] D. Muller and F. Preparata, Finding the intersection of 2 convex polyhedra, Tech. Rept., University of Illinois, 1977.
[6] M.H. Overmars, Searching in the past II: General Transforms, Tech. Rept., RUU-CS-81-9, Department of Computer Science, University of Utrecht, 1981.
[7] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *JCSS* **23** (2) (1981) 166–204.
[8] M. Shamos, Computational geometry, Ph.D. Thesis, Yale University, 1978.
[9] J.E. Warnock, A hidden-surface algorithm for computer generated half-tone pictures, Tech. Rept., University of Utah, Computer Science Department, 1969.
[10] G.S. Watkins, A real-time visible surface algorithm, UTEC-CSc-70-101, University of Utah, Computer Science Department, 1970.
[11] P.H. Winston, *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).