COMPUTER GRAPHICS AND IMAGE PROCESSING 19, 384–391 (1982)

NOTE

A New Linear Algorithm for Intersecting Convex Polygons

JOSEPH O'ROURKE, CHI-BIN CHIEN, THOMAS OLSON, AND DAVID NADDOR

Department of Electrical Engineering and Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218

Received July 23, 1981; revised October 7, 1981

An algorithm is presented that computes the intersection of two convex polygons in linear time. The algorithm is fundamentally different from the only known linear algorithms for this problem, due to Shamos and Hoey. These algorithms depend on a division of the plane into either angular sectors (Shamos) or parallel slabs (Hoey), and are mildly complex. Our algorithm searches for the intersection points of the polygons by advancing a single pointer around each polygon, and is very easy to program.

1. INTRODUCTION

It has been known for some time that the intersection of convex polygonal regions can be formed in linear time: $O(n + m)$ if the boundary polygons have n and m vertices [3, 4]. Partly because of this linearity, the intersection of convex polygons has been employed as a subcomponent of many other geometric algorithms. Shamos used convex region intersection in his $O(n \log n)$ half-plane intersection algorithm [3, 5], which itself can be used for computing the kernel of a polygon, for determining whether two sets of points in the plane are linearly separable, and for solving two-variable linear programming problems [3–5]. Muller and Preparata use linear convex polygon intersection in their algorithm for intersecting convex polyhedra [2], and Ahuja *et al.* use it in their interference detection algorithm [1].

Shamos was the first to discover a linear algorithm for intersecting convex polygons [3]. His algorithm depends on dividing one polygon into angular sectors, and locating each vertex of the other polygon either inside or outside of these sectors. Hoey later designed a somewhat simpler algorithm [4]. The plane is partitioned into parallel "slabs," with every polygon vertex lying on some slab edge. Within the slabs, the problem reduces to the intersection of two trapezoids, which can be performed in constant time. The resulting regions within the slabs can then be merged together in a single pass over all slabs to form the output polygon. Hoey's algorithm seems to be in wide use [1, 2, 4].

The algorithm presented here is fundamentally different from either Shamos's or Hoey's, and we feel it is considerably easier to program. The algorithm maintains two special pointers, distinguishing one edge on each polygon. These pointers are advanced around the polygons such that their edges "chase" one another, searching for the intersection points. All the intersection points can be found within two cycles around the polygons, and thus the algorithm achieves linear time complexity.

After establishing some notation, the algorithm is presented in the next section and intuitively justified in Section 3. Section 4 covers degenerate cases. A proof of correctness follows in the Appendix.

0146-664X/82/080384-08$02.00/0

2. THE ALGORITHM

Notation and Definitions

Let the two polygons P and Q be represented as circular lists of their vertices. Counterclockwise will be used as the positive sense, so that the inside of the polygon is always to the left during a positive traversal.

Let $\mathbf{p}$ and $\mathbf{q}$ be vertices of P and Q, respectively. The next and previous vertices with respect to $\mathbf{p}$ will be denoted by $\mathbf{p}_+$ and $\mathbf{p}_-$, respectively. The vector from $\mathbf{p}_-$ to $\mathbf{p}$ will be called $\dot{\mathbf{p}}$ (recalling the tangent vector of differential geometry), and similarly for $\dot{\mathbf{q}}$. Thus $\mathbf{p}$ is at the head of $\dot{\mathbf{p}}$.

The half-plane (including the half-plane edge) determined by $\dot{\mathbf{p}}$ will be represented by $hp(\dot{\mathbf{p}})$, and similarly for $\mathbf{q}$. In terms of vectors, hp can be defined as the set

$$hp(\dot{\mathbf{p}}) = \{\mathbf{x} : \dot{\mathbf{p}} \times (\mathbf{x} - \mathbf{p}_-) \geq 0\}$$

where $\mathbf{x}$ is an arbitrary point in the plane. Figure 1 illustrates these definitions.

Algorithm

The algorithm consists mainly of a set of "advance rules" embedded in a **repeat** loop. At each step, two edges are checked for intersection, and a vertex is output. When the loop finishes, all the vertices of the polygon $P \cap Q$ have been output, except when the boundaries of P and Q do not intersect. Special code handles this possibility.

The algorithm below assumes that all intersections are *nondegenerate*: whenever two edges intersect, they intersect in a single point that is not a vertex of either polygon. Modifications to handle degenerate cases are described in Section 4. The algorithm is presented in an informal style that should nevertheless be easy to directly implement. Comments are enclosed within brackets.

```
input: P and Q
output: P ∩ Q
begin {main}
Choose p and q arbitrarily.
repeat
    {Check to see if ṗ and q̇ intersect}
    if ṗ and q̇ intersect then
        if this intersection is the same as the first intersection then halt
        else output the point of intersection and
            {set inside} if p ∈ hp(q̇) then inside ← "P" else inside ← "Q";
    {Advance either p or q.}
    if q̇ × ṗ ≥ 0 then
        if p ∈ hp(q̇) then (advance q) else (advance p)
    else {q̇ × ṗ < 0}
        if q ∈ hp(ṗ) then (advance p) else (advance q);
until repeat has executed more than 2(|P| + |Q|) times;
{either P ∩ Q = ∅ or P ⊇ Q or P ⊆ Q}
Choose p and q arbitrarily.
if p ∈ Q then output P else if q ∈ P then output Q else output ∅;
end {main}
```

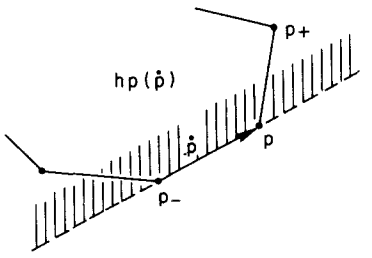

FIG. 1. Notational conventions used to label vertices and edges of a polygon, and the half-plane determined by an edge.

advance **p**:
    output **p** if *inside* = "*P*";
    p ← $\mathbf{p}_+$;
advance **q**:
    output **q** if *inside* = "*Q*";
    q ← $\mathbf{q}_+$;

The algorithm has been implemented, and a sample of its operation is detailed in Fig. 2.

Determining whether $\mathbf{p} \in Q$ (i.e., whether **p** is inside or on the boundary of the polygon Q) can easily be accomplished in linear time by summing the angles subtended at **p** by each edge of Q: if the sum is 360°, then **p** is inside, otherwise the sum is 0° and **p** is outside.

3. INTUITIVE JUSTIFICATION

The crux of the algorithm is in the advance rules, and these are somewhat obscure. To make their operation more transparent, define a predicate *aim* ($\dot{\mathbf{p}}$, $\dot{\mathbf{q}}$) as follows:

aim ($\dot{\mathbf{p}}$, $\dot{\mathbf{q}}$) = *true* iff

$$\mathbf{p} \in hp(\dot{\mathbf{q}}) \text{ and } \dot{\mathbf{q}} \times \dot{\mathbf{p}} < 0$$

or

$$\mathbf{p} \notin hp(\dot{\mathbf{q}}) \text{ and } \dot{\mathbf{q}} \times \dot{\mathbf{p}} \geq 0.$$

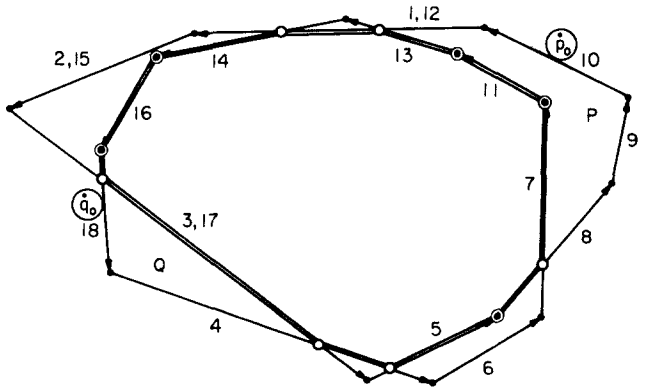

FIG. 2. Execution of the algorithm. The initial positions of $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ are labeled $\dot{\mathbf{p}}_0$ and $\dot{\mathbf{q}}_0$, and an edge of either polygon is labeled *i* if the *i*th iteration of the **repeat** loop caused an advance to that edge. The circled vertices are output by the algorithm.

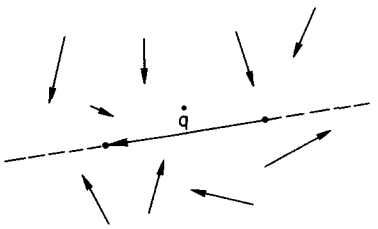

FIG. 3. All of the unlabeled vectors are "aiming towards" $\dot{\mathbf{q}}$'s half-plane edge.

Informally, *aim* $(\dot{\mathbf{p}}, \dot{\mathbf{q}})$ is *true* iff $\dot{\mathbf{p}}$ is "aiming toward" $\dot{\mathbf{q}}$'s half-plane edge (see Fig. 3). In terms of this predicate, the advance rules reduce to Table 1. In this table, the minus sign means *not*, and $\mathbf{p}$ is "outside" when $\mathbf{p} \notin hp(\dot{\mathbf{q}})$, and similarly for $\mathbf{q}$. It is easily established that the phrase "whichever is outside" is unambiguous in the table, and that the table is equivalent to the advance rules presented in the algorithm.

The justification for this table is that an intersection of $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ may be sought by trying to move $\dot{\mathbf{p}}$ so that it crosses $\dot{\mathbf{q}}$'s half-plane edge, and similarly for $\dot{\mathbf{q}}$. So whenever just one aims towards the other's half-plane edge, it should be advanced. In the cases where either both or neither aim towards one another, the outside one is advanced. The idea here is that, since all of the polygon Q is included in $hp(\dot{\mathbf{q}})$, when $\mathbf{p}$ is outside it must be advanced until it crosses into $hp(\dot{\mathbf{q}})$ before any intersection is possible.

Although the intuitive justification should now be clear, it is not immediately obvious that every intersection point will be found. A straightforward but tedious proof of the algorithm's correctness is offered in the Appendix.

4. DEGENERATE CASES

There are three types of "degenerate" intersections that may occur: a vertex of P may lie on an edge of Q, a vertex of P may coincide with a vertex of Q, and an edge of P may be collinear with and overlap an edge of Q. Our claim is that the previously presented algorithm will handle all three cases properly when (a) $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ are interpreted as having *no* intersection whenever they are collinear (i.e., whenever $\dot{\mathbf{q}} \times \dot{\mathbf{p}} = 0$), even if they overlap, and (b) the loop exit condition

if this intersection is the same as the first intersection **then** halt

is modified to read

if the first intersection was *not* found during the previous loop iteration
then if this intersection is the same as the first intersection **then** halt.

TABLE 1

The Advance Rules Expressed in Terms of the *aim* Predicate

	$aim(\dot{\mathbf{p}}, \dot{\mathbf{q}})$	$-aim(\dot{\mathbf{p}}, \dot{\mathbf{q}})$
$aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$	advance whichever is outside	advance $\mathbf{q}$
$-aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$	advance $\mathbf{p}$	advance whichever is outside

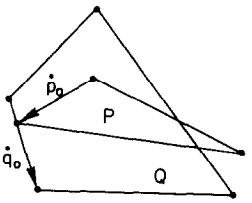

FIG. 4. A simple degenerate intersection that necessitates suspending the usual loop exit condition.

This last change is required to exclude the case illustrated in Fig. 4.

These claims concerning degenerate cases are justified in the Appendix.

5. FINAL REMARKS

By merely flipping the sense of the variable *inside*, the algorithm will output the union rather than the intersection. (Minor modifications must also be made to the code after the end of the repeat loop.)

Although it seems superficially feasible, we have been unable to extend the algorithm to compute the intersection of non-convex polygons, or to compute the intersection of convex polyhedra.

APPENDIX

Assume that none of the cases $P \cap Q = \varnothing$, $P \supset Q$, or $P \subset Q$ holds, as these are obviously handled correctly by the algorithm as special cases. Further assume that all intersections between the polygons are *nondegenerate*: when two edges intersect, they intersect at a single point which is not a vertex of either polygon. Degenerate intersections will be discussed at the end of this section.

The proof will proceed by first establishing that at least one intersection point will be discovered, and then showing that once one intersection is found, the next one is guaranteed to be found also.

Because we have excluded nonintersecting and degenerate cases, the two polygon boundaries must intersect at two or more points. It is clear that these intersection points can be ordered into a counterclockwise circular list just as the polygon vertices are. Thus we can unambiguously speak of the "next intersection point," and since there are at least two intersection points, the next one will always be distinct from the current one. Also note that if at one intersection point P enters Q, then at the next intersection point P exits Q, and similarly for Q: the direction of passage alternates along the sequence of intersection points.

LEMMA 1. *If the boundaries of P and Q intersect nondegenerately, the algorithm will find at least one intersection point.*

Proof. Assume the contrary: all intersection points are missed and the loop iterates $2(|P| + |Q|)$ times before exiting. Since each iteration advances either $\mathbf{p}$ or $\mathbf{q}$, after $|P| + |Q|$ iterations either $\mathbf{p}$ or $\mathbf{q}$ (or perhaps both) must have made a complete circuit of its polygon.

Suppose $\mathbf{q}$ is the one that makes a complete circuit. Then at some point $\mathbf{q}$ must be situated such that $\dot{\mathbf{q}}$ is at an intersection point where the polygon P passes from the outside of Q to its inside. We can be assured of such a situation because there are at least two intersection points and they alternate in direction of passage.

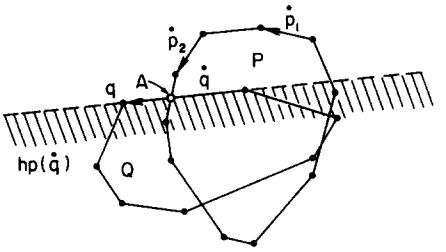

FIG. 5. When an intersection point A lies on $\dot{\mathbf{q}}$ and $\mathbf{p}$ is outside of $hp(\dot{\mathbf{q}})$ (as are $\mathbf{p}_1$ and $\mathbf{p}_2$), then $\dot{\mathbf{q}}$ will remain stationary and $\dot{\mathbf{p}}$ will meet it at A.

It will now be shown that once $\dot{\mathbf{q}}$ is situated as described above, the remaining $|P|+|Q|$ iterations of the loop will necessarily result in an intersection being found, contradicting our assumption that all are missed.

When $\dot{\mathbf{q}}$ first arrives at the intersection point, $\mathbf{p}$ is either inside or outside $hp(\dot{\mathbf{q}})$. These two cases will be considered separately.

Case 1: $\mathbf{p} \notin hp(\dot{\mathbf{q}})$ (see Fig. 5).

Our claim is that $\mathbf{q}$ remains stationary while $\mathbf{p}$ advances until $\dot{\mathbf{p}}$ intersects $\dot{\mathbf{q}}$. Consider the initial step, when $\mathbf{p}$ is outside.

If $\dot{\mathbf{p}}$ does not aim towards $hp(\dot{\mathbf{q}})$ (e.g., $\mathbf{p}_1$ in Fig. 5), then the "$-aim\ (\dot{\mathbf{p}}, \dot{\mathbf{q}})$" column of Table 1 is applicable. It is clear from the figure that in this case the "$-aim\ (\dot{\mathbf{q}}, \dot{\mathbf{p}})$" row is the appropriate one, so $\mathbf{p}$ will be advanced and $\mathbf{p}$ will remain outside.

If $\dot{\mathbf{p}}$ does aim towards $hp(\dot{\mathbf{q}})$ (e.g., $\dot{\mathbf{p}}_2$ in Fig. 5), then the "$aim(\dot{\mathbf{p}}, \dot{\mathbf{q}})$" column is applicable and again $\mathbf{p}$ will be advanced. Clearly, $\mathbf{p}$ cannot change its status from outside to inside without crossing $\dot{\mathbf{q}}$.

Thus the initial step advances $\mathbf{p}$ and leaves it outside if it does not cross $\dot{\mathbf{q}}$. But this leaves us in the same situation as that which held before the initial step. Thus, until $\dot{\mathbf{p}}$ crosses $\dot{\mathbf{q}}$, $\dot{\mathbf{p}}$ will remain outside and only $\mathbf{p}$ will be advanced. This establishes the claim.

Case 2: $\mathbf{p} \in hp(\dot{\mathbf{q}})$ (see Fig. 6).

Here there are two distinct possibilities. If $\dot{\mathbf{p}}$ aims towards $hp(\dot{\mathbf{q}})$ (e.g., $\dot{\mathbf{p}}_1$ in Fig. 6), then because it is *not* the case that $aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$,Table 1 shows that $\mathbf{p}$ will be advanced. In fact, $\mathbf{p}$ will be advanced until it emerges from $hp(\dot{\mathbf{q}})$, which gives *Case 1*. Therefore an intersection will be found.

If $\dot{\mathbf{p}}$ aims away from $hp(\dot{\mathbf{q}})$ (e.g., $\dot{\mathbf{p}}_2$ in Fig. 6), we claim that $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ will meet at the next intersection point. This will be established by proving that it is not possible for either $\mathbf{p}$ or $\mathbf{q}$ to pass through the next intersection point before the other.

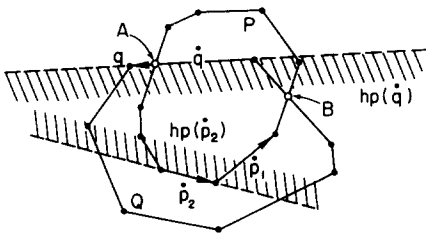

FIG. 6. When an intersection point A lies on $\dot{\mathbf{q}}$ and $\mathbf{p}$ is inside of $hp(\dot{\mathbf{q}})$, then if $\dot{\mathbf{p}}$ aims toward $hp(\dot{\mathbf{q}})$ (as $\dot{\mathbf{p}}_1$ does), then $\dot{\mathbf{q}}$ will remain stationary and $\dot{\mathbf{p}}$ will meet it at A. If $\dot{\mathbf{p}}$ aims away from $hp(\dot{\mathbf{q}})$ (as $\dot{\mathbf{p}}_2$ does), then they will meet at B, the next intersection point.

First, if $\mathbf{q}$ is inside $hp(\dot{\mathbf{p}})$ (as it is with $\dot{\mathbf{p}}_2$ in Fig. 6), then $aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$ is true, but our hypothesis is that $aim(\dot{\mathbf{p}}, \dot{\mathbf{q}})$ is false, so $\mathbf{q}$ would be advanced. Thus eventually it will be the case that $\mathbf{q} \notin hp(\dot{\mathbf{p}})$.

Once this latter condition holds, the advancement of $\mathbf{p}$ and $\mathbf{q}$ follows very simple rules up until the next intersection point: $\mathbf{p}$ advances only if $\mathbf{q} \in hp(\dot{\mathbf{p}})$, and therefore $\mathbf{q}$ advances only if $\mathbf{q} \notin hp(\dot{\mathbf{p}})$. To see this, suppose that $\mathbf{q} \notin hp(\dot{\mathbf{p}})$ but that $\mathbf{p}$ advances. Since $\mathbf{q}$ is outside, only one cell of Table 1 could account for $\mathbf{p}$'s advancement. But that cell requires both $aim(\dot{\mathbf{p}}, \dot{\mathbf{q}},)$ and $-aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$, conditions that cannot simultaneously hold under the described situation.

Now since $\mathbf{p}$ can advance only when $\mathbf{q}$ enters $hp(\dot{\mathbf{p}})$, it is impossible for $\mathbf{p}$ to advance past the next intersection point while $\mathbf{q}$ remains outside. So the only way for the intersection point to be missed is for $\mathbf{q}$ to pass through while $\mathbf{p}$ hangs back. But $\mathbf{q}$ clearly cannot do this without crossing $\mathbf{p}$'s half-plane edge. Since neither $\mathbf{p}$ nor $\mathbf{q}$ can pass through next intersection point before the other, and since the $|P| + |Q|$ available iterations will force at least one of them past the intersection point, $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ must meet there.

Thus in both of the *Cases* (which exhaust all possibilities), an intersection will be found, contradicting our denial of the Lemma. □

LEMMA 2. *If the boundaries of P and Q intersect nondegenerately, and if* $\dot{\mathbf{p}}$ *and* $\dot{\mathbf{q}}$ *are situated at an intersection point, then the next intersection point will be properly found, and it will in fact be the* next *one to be found (i.e., no intersection points will be skipped).*

Proof. If $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ intersect, then it must be the case that either $\mathbf{q} \notin hp(\dot{\mathbf{p}})$ and $\mathbf{p} \in hp(\dot{\mathbf{q}})$, or vice versa (interchanging $\mathbf{p}$ and $\mathbf{q}$). Assume the former without loss of generality. This corresponds precisely to the second possibility in *Case 2* of Lemma 1, which established that $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ would meet at the next intersection point. □

THEOREM. *If the boundaries of P and Q intersect nondegenerately, then the algorithm will find all of the intersection points in order.*

Proof. Follows immediately from Lemmas 1 and 2. □

It should be mentioned that the convexity of the two polygons was used implicitly at a number of places throughout the above proofs. For example, in the analysis of *Case 1* in Lemma 1, when $-aim(\dot{\mathbf{p}}, \dot{\mathbf{q}})$ holds, the condition $aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$ can be shown to require P to be nonconvex, which justifies our choice of the "$-aim(\dot{\mathbf{q}}, \dot{\mathbf{p}})$" row of the table.

Finally, we consider degenerate intersections. The claim of Section 4 will be argued informally, as a detailed proof is even more tedious than the foregoing.

There are only two ways in which the algorithm could go awry in degenerate cases:

(1) The variable *inside* might be improperly set (due to a vertex falling on the edge of a half-plane), thus causing a vertex to be output by the "advance" function which in fact should not be output.

(2) The advance rules may select the wrong pointer (due to either a zero cross product of $\dot{\mathbf{q}}$ and $\dot{\mathbf{p}}$, or a vertex falling on the edge of a half-plane), allowing one pointer to "slip through" an intersection point when it should not.

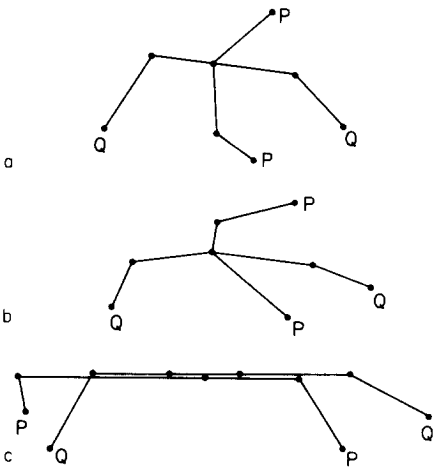

FIG. 7. The three classes of degenerate intersection. (a) A vertex of P falls on an edge of Q. (b) P and Q share a common vertex. (c) An edge of P overlaps an edge of Q.

Consider the three classes of degenerate intersections mentioned in Section 4, instances of which are illustrated in Fig. 7. In Figs. 7a and b, it does not matter if *inside* is set "improperly," since the only vertex that might be output is actually on the boundary of $P \cap Q$. This vertex may be output twice in succession, but we assume this is no problem. Because of the convention that collinear vectors do not intersect, the situation depicted in Fig. 7c cannot itself cause *inside* to be set. If, however, *inside* was first set by a condition like that illustrated in Figs. 7a or b, it might happen that the wrong one of two collinear vectors is advanced, causing an incorrect output. This never occurs for a fairly subtle reason: *inside* is set to "P" whenever there is a borderline case, but the Q pointer is the one advanced in the collinear degenerate cases. Thus these advances are always "safe" in that the value of *inside* will prevent any output. Thus the first way that the algorithm might go wrong does not occur.

It is easily established that the degeneracies in Figs. 7a and b do not modify the arguments presented in *Case 2* of Lemma 1: these types of degenerate intersection points prevent passage of one pointer before the other in the same manner as normal intersection points. In the type of degeneracy illustrate in Fig. 7c, it does not matter which of the two pointers is advanced, since (a) any vertex in the overlap portion is on the boundary of $P \cap Q$ and can be output without error, and (b) eventually, one of the cases shown if Figs. 7a or b will hold. Thus the algorithm will not make any errors of type (2) above.

REFERENCES

1. N. Ahuja, R. T. Chien, R. Yen, and N. Birdwell, Interference detection and collision avoidance among three dimensional objects, Proceedings of First Annual National Conference on Artificial Intelligence, pp. 44–48, Stanford, California, 1980.
2. D. E. Muller and F. P. Preparata, Finding the intersection of two convex polyhedra, *Theoret. Comput. Sci.* 7, 1978, 217–236.
3. M. I. Shamos, Geometric Complexity, Proceedings of Seventh Annual ACM Symposium on Theory of Computing, pp. 224–233, 1975.
4. M. I. Shamos, Computational Geometry, Ph.D. Dissertation, Yale University, 1978.
5. M. I. Shamos and D. Hoey, Geometric Intersection Problems, Proceedings of Seventeenth Annual Symposium on Foundations of Computer Science, pp. 208–215, Houston, Texas, 1976.