Computing the Intersection-Depth of Polyhedra

David Dobkin* John Hershberger† David Kirkpatrick‡
Subhash Suri§

Abstract

Given two intersecting polyhedra P, Q and a direction d, find the smallest translation of Q along d that renders the interiors of P and Q disjoint. The same problem can also be posed without specifying the direction, in which case the minimum translation over all directions is sought. These are fundamental problems that arise in robotics and computer vision. We develop techniques for implicitly building and searching convolutions and apply them to derive efficient algorithms for these problems.

1 Introduction

The computation of spatial relationships among geometric objects is a fundamental problem in such areas as robotics, computer-aided design, VLSI layout, and computer graphics. In a dynamic environment where objects are mobile, intersection or proximity among objects has obvious applications. Consider, for instance, the problem of collision detection in robot motion planning. The Euclidean distance is a commonly used measure in these areas. Numerous efficient algorithms are known for computing the minimum distance between two polyhedra in two and three dimensions (see [10, 12, 14]). Whenever two objects intersect, this distance measure is zero. Thus, it fails to provide any information about the *extent of penetration*. The notion of *negative distance* has been proposed by Buckley and Leifer [6] and Cameron and Culley [7] to rectify this discrepancy. We follow Keerthi and Sridharan [25] and define the following measure of negative distance.

*Department of Computer Science, Princeton University, NJ 08544, U.S.A., and Bell Communications Research, Morristown, NJ 07960, U.S.A. The work of this author was partially supported by National Science Foundation Grant CCR90-02352.

†DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, U.S.A.

‡Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1W5, Canada. The work of this author was partially supported by grant A3583 from the Natural Sciences and Engineering Research Council of Canada.

§Bell Communications Research, 445 South Street, Morristown, NJ 07960, U.S.A.

Given two intersecting polyhedra P and Q and a direction d, let $\sigma_d(P,Q)$ denote the minimum distance by which Q must be translated along d to make the interiors of P and Q disjoint. We call $\sigma_d(P,Q)$ the *intersection-depth* between P and Q in direction d. The *minimum intersection-depth* between P and Q, denoted $\sigma(P,Q)$, is defined as the minimum of $\sigma_d(P,Q)$ over all directions d. The definition of $\sigma(P,Q)$ agrees with the measure proposed by Cameron and Culley [7].

The fixed-direction intersection-depth problem can be reduced to a separation problem: move one of the polyhedra sufficiently far in direction d to ensure that the two polyhedra are disjoint, and then find their directional separation. The intersection-depth can be inferred by subtracting the separation from the initial displacement. This gives an alternative way to consider and attack the problem; a safe estimate of the initial displacement can often be made efficiently, for example, by finding the extremal vertices of the two polytopes in directions $\pm d$.

The problem of computing the intersection-depth between two intersecting objects is addressed by Cameron and Culley [7]; however, the complexity of their algorithm is not analyzed. Keerthi and Sridharan [25] consider the problem for two convex polygons. Given two intersecting convex polygons P and Q of n and m vertices, respectively, they propose an $O(\log n \log m)$ time algorithm for computing the directional intersection-depth, $\sigma_d(P,Q)$. For the minimum intersection-depth, $\sigma(P,Q)$, they achieve $O(n+m)$ time. In this paper, we derive the following improvements and extensions of these results.

Given two intersecting convex polygons P and Q, where $|P| = n$ and $|Q| = m$, we can determine $\sigma_d(P,Q)$ for any direction d in optimal time $O(\log n + \log m)$. In addition, after linear-time preprocessing, we can in $O(\log(n+m))$ time compute the minimum intersection-depth, $\sigma(P,Q)$, for any placement of Q.

Next, we relax the convexity assumption on one of the polygons. If P is a simple nonconvex polygon and Q is a convex polygon, we can build a data structure in $O(n \log^2(n+m) + m)$ time such that directional intersection-depth queries and minimum intersection-depth queries can both be answered for any placement of Q in time $O((n+m)\log(n+m))$. The data structure can be extended to support directional intersection-depth queries in $O(n^{2/3+\delta})$ time. The data structure requires $O(n+m)$ space. The query time can be reduced to $O(\sqrt{n}\log^2 n)$ by increasing the preprocessing time and space.

We then consider the directional intersection-depth problem for two convex polyhedra in three dimensions. For suitably preprocessed polyhedra P and Q, we can determine $\sigma_d(P,Q)$ in time $O(\log n \log m)$, for any direction d. The representation assumed for the polyhedra is the hierarchical representation, originally proposed by Dobkin and Kirkpatrick [11, 12, 13], which can be built in linear time.

The unifying idea in all our algorithms is implicitly searching the convolution of two polytopes. It is well known that if two polytopes P and Q intersect, then the difference of their reference vectors lies in their convolution [21]. In fact, the problem of computing the intersection-depth reduces to calculating the distance between the boundary of a convolution and a point inside it. In all interesting cases, however, an explicit construction of the convolution can be rather expensive; in two dimensions, the convolution can have quadratic size if one of the polygons is nonconvex, and in three dimensions, the size can be quadratic even for two convex polyhedra. We develop methods for building and searching the convolutions implicitly, and apply them to solve the intersection-depth problems in time sublinear in the size of the convolution.

2 Minkowski Sum

Given two sets A and B of vectors, their *Minkowski sum* or *convolution* is the set

$$A \oplus B = \{a + b \mid a \in A \text{ and } b \in B\}.$$

In all our applications, the ambient space is two or three-dimensional Euclidean space, the sets are polyhedra, and the elements are points taken as vectors. We will need to distinguish between various placements of the same shape (or solid) in the space, and so each polyhedron is endowed with a *reference point*. Since only the translates of a polyhedron are of interest, the placement of a polyhedron is completely specified by the coordinates of its reference point. The polyhedron P translated by a vector x is denoted P^x. Symbols without the superscript denote the "default" placement of the associated polyhedron, where the reference point is at the origin. Finally, the polyhedron P reflected through the origin is denoted $(-P)$:

$$(-P) = \{-a \; : \; a \in P\}.$$

The following lemma establishes a connection between the intersection and the convolution.

Lemma 2.1 ([21]) *Let P and Q be two polyhedra, and let x and y be vectors. Then $P^x \cap Q^y \neq \emptyset$ if and only if $y - x \in P \oplus (-Q)$.*

Let P and Q be two intersecting polyhedra. Without loss of generality, assume that the reference point of P coincides with the origin of our coordinate system, and that the reference point of Q is given by the vector q. Their initial placements satisfy $P \cap Q^q \neq \emptyset$, which by the preceding lemma implies that $q \in P \oplus (-Q)$. A displacement of Q in direction d is given by Q^{q+td}, for some $t \geq 0$. Thus, the problem of computing $\sigma_d(P, Q^q)$ is equivalent to finding the *minimum* t such that $P \cap Q^{q+td} = \emptyset$. By Lemma 2.1, this is equivalent to finding the *minimum* t such that

$$q + td \notin P \oplus (-Q). \tag{1}$$

Since $q + td$ is the parametric equation of a ray, whose origin is q and direction is d, the minimum t for which (1) holds is determined by the (first) intersection of the ray $q + td$ with the boundary of $P \oplus (-Q)$. Thus finding the intersection-depth in a fixed direction reduces to the problem of intersecting a ray with a Minkowski-sum polyhedron.

The problem of finding $\sigma(P, Q)$, the smallest translation in *any* direction that would separate P and Q, is equivalent to finding the minimum distance between q and the boundary of $P \oplus (-Q)$:

$$\sigma(P, Q^q) \;=\; \min(t \geq 0 \mid \exists d \text{ with } |d| = 1 \text{ and } q + td \notin P \oplus (-Q)). \qquad (2)$$

3 Convex Polygons

It is well known that the Minkowski sum of two convex polyhedra is a convex polyhedron [18]. In two dimensions, if P and Q are convex polygons of n and m vertices, respectively, then $P \oplus Q$ is a convex polygon of $n+m$ vertices; an exception occurs if P and Q have parallel edges, in which case the convolved polygon has fewer than $n + m$ vertices. In the plane, $P \oplus Q$ has a particularly simple characterization: its edges are those of P and Q, merged in slope order [20]. This fact allows one to compute $P \oplus Q$ in linear time.

Theorem 3.1 ([20]) *Given two convex polygons P and Q, with n and m vertices, respectively, we can compute their convolution $P \oplus Q$ in time $O(n + m)$.*

The directional intersection-depth between P and Q can be computed by implicitly building that portion of $P \oplus (-Q)$ that intersects the ray from q in direction d. Guibas and Stolfi [21] use a similar method to detect whether two given convex polygons intersect. Observe that testing whether P intersects Q^q is equivalent to checking if $q \in P \oplus (-Q)$. Guibas and Stolfi achieve $O(\log n)$ time by determining the position of those edges of the convolution that intersect the horizontal line through q. The containment of q in the convolution can be determined by testing against these edges. The method of Guibas and Stolfi can be easily modified to determine, in $O(\log n)$ time, the point of intersection between a ray originating from an interior point q and the convolution $P \oplus (-Q)$.

We can also perform a binary search directly in the *object space*, rather than the convolution (i.e., the *configuration space*) to compute $\sigma_d(P, Q)$, using a method similar to Chazelle and Dobkin's method for detecting the intersection between two convex polygons [9]. The procedure repeatedly selects the median edges on the two polygonal chains and then, based on some local computations involving the median edges, discards half of one of the chains. For further details, we refer the reader to [9].

Next, to solve the problem of finding the minimum intersection-depth over all directions, we first compute the convolution $P \oplus (-Q)$, in $O(n)$ time; we assume that each of the polygons has at most n vertices. The minimum depth is found by determining the point closest to q on the boundary of $P \oplus (-Q)$. The latter task can be accomplished easily in linear time by checking all the edges of the convolution.

Once $P \oplus (-Q)$ is computed, we can also answer the following query efficiently: given Q^y, find the minimum translation needed in any direction to separate Q from P. To handle these queries, we compute the medial axis of the polygon $P \oplus (-Q)$ and build a point-location structure for it, using $O(n)$ time altogether [2, 15, 26]. (The *medial axis* of a polygon P is the locus of all points in P that are equidistant from at least two points on the boundary of P. The medial axis of a convex polygon of n vertices can be computed in $O(n)$ time [2].) Given a query placement Q^y, we can find the closest point to y on the boundary of $P \oplus (-Q)$ in $O(\log n)$ time by point location in the medial-axis diagram. This establishes the following theorem.

Theorem 3.2 *Let P and Q be two convex polygons, with a total of n vertices. After $O(n)$ time preprocessing, given a placement Q^y, we can compute $\sigma(P, Q^y)$ in time $O(\log n)$.*

In the following sections, we extend these ideas to solve similar problems for non-convex polygons and three-dimensional convex polyhedra. The main difficulty in these generalizations is that the Minkowski sum of non-convex polygons or 3D polyhedra may have quadratic complexity, which makes their explicit construction too costly. Instead, we develop methods for implicitly constructing and searching these structures.

4 One Convex and One Nonconvex Polygon

Let P be a simple nonconvex polygon of n vertices and let Q be a convex polygon of m vertices. Given a placement of Q, we want to compute the intersection-depth, either in a fixed direction or the minimum over all directions. The Minkowski sum of P and Q is again a polygon; however, in contrast to the convex case, $P \oplus Q$ is in general multiply connected, and may have $\Omega(nm)$ vertices. Therefore, an explicit construction of the convolution may be too expensive. Our main result in this section is a method for constructing and searching an implicit representation of $P \oplus Q$. The representation can be constructed in $O(n \log^2(n+m) + m)$ time, stored in $O(n+m)$ space, and searched efficiently for the depth queries. At the top-level, our method consists of the following steps.

Construct Convolution

Step 1. Triangulate P. Let $T_1, T_2, \ldots, T_s$ be the triangles in this triangulation ($s = n - 2$).

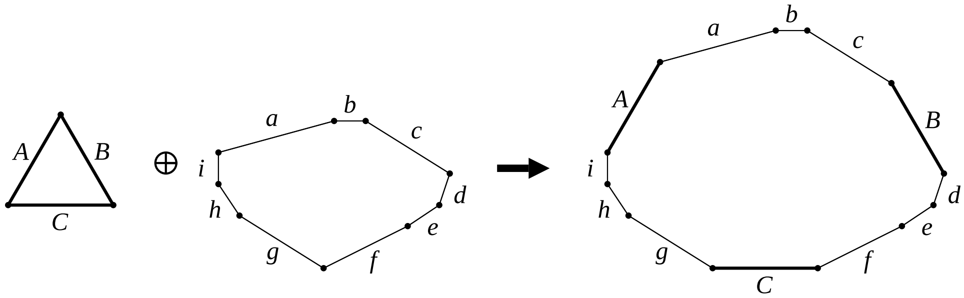

Figure 1: A convolved triangle

Step 2. Compute an implicit representation of the convolutions $R_i = T_i \oplus (-Q)$, for $i = 1, 2, \ldots, s$.

Step 3. Compute an implicit representation of the boundary of $R = \cup R_i$.

4.1 Details of the Construction

We fix a Cartesian coordinate system in the plane and, without loss of generality, assume that the origin lies in Q; we take the origin as the reference point of Q.

Step 1 is easily accomplished in $O(n \log n)$ time, using any of the well-known triangulation algorithms [17, 23]; in fact the time complexity can be reduced to $O(n)$ using a recent algorithm of Chazelle [8]. Pick a reference point r_i for each of the triangles T_i; for instance, we may pick the vertex closest to the origin.

Step 2 computes an implicit representation of the convolutions of these triangles with the convex polygon $(-Q)$. The key to this step is the observation that the convolution of two convex polygons has a particularly simple form, which in the case of a triangle T_i and a convex polygon Q can be determined implicitly in $O(\log m)$ time. Recall that if A and B are two convex polygons, then the boundary of $A \oplus B$ consists of edges of A and B merged in slope order [20]. To compute $T_i \oplus (-Q)$, therefore, we need only to locate the slopes of each of the three edges of T_i among the slopes of the edges of $(-Q)$, which takes three binary searches and $O(\log m)$ time. To represent $R_i = T_i \oplus (-Q)$ implicitly, we just record the three places in the array storing $(-Q)$ where the edges of T_i are inserted. We call the convolution R_i a *convolved triangle*; see Figure 1 for an illustration.

Implicit Representation of the Convolved Triangles

In our description of the convolution so far we have concentrated on the "form" and ignored the "placement" issue. A uniform way to resolve this discrepancy is to convolve $(-Q)$ with a canonical copy of T_i, placed at the origin, and then translate the result by the vector r_i. More precisely, let $T_i^o = \triangle Oab$ be the canonical copy of

T_i, where O is the origin and a and b are vectors denoting the remaining two vertices of the triangle. Let $q_1, q_2, \ldots, q_m$ be the ordered list of vertices of $(-Q)$. Suppose that the slope of Oa lies between the slopes of $q_{i-1}q_i$ and $q_i q_{i+1}$. Similarly, suppose ab (resp. Ob) lies between $q_{j-1}q_j$ and $q_j q_{j+1}$ (resp. $q_{k-1}q_k$ and $q_k q_{k+1}$). Then the ordered list of vertices forming the boundary of $T_i^o \oplus (-Q)$ is given by

$$q_{i-1}, q_i, (q_i + a), (q_{i+1} + a), \ldots, (q_j + a), (q_j + b), (q_{j+1} + b), \ldots, (q_k + b), q_k, q_{k+1}, \ldots, q_{i-1}.$$

Finally, to obtain the convolution $T_i \oplus (-Q)$, we add the vector r_i to each vertex of $T_i^o \oplus (-Q)$. It is clear that the indices i, j, k and vectors a, b and r_i completely, though implicitly, encode a representation of $T_i \oplus (-Q)$. This is precisely the implicit representation of R_i stored by our algorithm.

Next, we discuss the details of Step 3, in which we compute the boundary of the union of these convolved triangles.

Union of Convolved Triangles

We want to compute $R = \cup_{i=1}^{s} R_i$, where each of the R_i is a convex polygon with $m+3$ vertices. An explicit description of R will require $\Omega(nm)$ space in the worst case. We therefore need to exploit the fact that each R_i is (implicitly) representable as a six-sided figure: three straight lines and three "arcs" of a convex polygon. This alone still does not suffice since, in general, even the boundary of the union of s triangles can have quadratic size. Fortunately, a result of Kedem et al. [24] shows that a collection of convolved triangles is "well-behaved"; we use that result to construct a linear-space implicit representation of the boundary of R. Our approach is similar in spirit to that of Hershberger and Guibas [22]. We use the following two facts proved in Kedem et al. [24].

Fact 4.1 ([24], p. 66) *Let A_1 and A_2 be two convex polygons with disjoint interiors, and let B be another convex polygon. Then the boundaries of $A_1 \oplus B$ and $A_2 \oplus B$ intersect in at most two points.*

Fact 4.2 ([24], p. 61) *Let A_i, $i = 1, 2, \ldots, s$, be a collection of convex polygons with disjoint interiors, and let B be another convex set. Then the boundary of $R = \cup_{i=1}^{s}(A_i \oplus B)$ has at most $6s - 12$ points of local nonconvexity.*

(Given a set S, a point $p \in S$ is called a point of *local nonconvexity* if each open disk centered on p contains two points $x, y \in S$ such that the segments $px, py \in S$ but the segment xy is not contained in S.)

A Sweep-line Algorithm

We use these facts to get an upper bound on the running time of a sweep-line algorithm for computing the boundary of R. The algorithm we use is due to Ottman, Widmeyer and Wood [28], which in turn is based on a technique of Bentley and Ottman [5] for counting and reporting the intersections in a collection of planar line segments. These algorithms run in $O((n+t)\log(n+t))$ time, where n and t, respectively, are the numbers of line segments and segment intersections. (Ottman, Widmeyer and Wood extend the basic plane sweep algorithm of Bentley and Ottman to perform boolean mask operations. Computing the boundary of the union of two sets of polygons, each set free of self-intersections, is the boolean mask operation we use. Readers unfamiliar with the paradigm of plane sweep are encouraged to refer to Preparata and Shamos [29].)

In our case, the collection consists of straight-line segments as well as *implicitly represented arcs of a convex polygon.* The algorithm of Ottman et al. works just as efficiently in this setting too, with one minor modification. In addition to intersecting two line segments, the algorithm must also be able to intersect two convex polygonal arcs. Intersecting two arbitrary convex arcs can be difficult in general; however, there are lucky breaks that we are able to exploit. First, by Fact 4.1, any two arcs intersect in at most two points. Second, we intersect the arcs in the order the sweep line encounters them. Thus, when our algorithm tries to compute the intersections of two arcs, their leftmost points lie on the sweep line. Since each of the arcs is given as an ordered list of vertices in an array, we can intersect them in time $O(\log m)$ by a binary search. (The details of this search are quite straightforward and can easily be worked out by the reader.) The worst-case running time of the sweep line algorithm is still $O((n+t)\log(n+t))$ since the overhead cost per intersection point is $O(\log n)$. The following is a high-level description of our algorithm.

Algorithm to Compute $R = \cup R_i$.

1. Divide the set of input convolved triangles into two halves: $R_1, \ldots, R_{\lfloor s/2 \rfloor}$ and $R_{\lfloor s/2 \rfloor + 1}, \ldots, R_s$.
2. Recursively compute the union of the two halves, $U = \bigcup_{i=1}^{\lfloor s/2 \rfloor} R_i$, and $V = \bigcup_{i=\lfloor s/2 \rfloor + 1}^{s} R_i$.
3. Merge the two contours U and V using the (modified) Ottman-Widmeyer-Wood algorithm.

Fact 4.2 implies that the total number of intersections t is at most $6s - 12$, which is $O(n)$, and hence the merge phase of our algorithm (step 3) takes $O(n\log(n+m))$. The entire algorithm takes $O(n\log^2(n+m) + m)$ time.

The boundary of R has a total of $O(n)$ vertices, and each pair of adjacent vertices is joined either by a straight line segment (corresponding to an edge of P) or an implicitly described convex arc (corresponding to some portion of the boundary of $(-Q)$). We can store this boundary in $O(n+m)$ space.

4.2 Computing the Intersection-Depth using R

Having computed an implicit representation of $R = \cup R_i$, we now discuss how to use it to solve the intersection-depth problem.

In the case of directional intersection-depth, we are given a placement of Q, say Q^q, and want to compute $\sigma_d(P, Q^q)$. Recalling the relation (1), this distance is equal to the distance between q and its closest point on the boundary of R in direction d. Equivalently, we need to find the first intersection of R with the ray from q in direction d. We intersect the ray with each of $O(n)$ bounding edges and arcs of R, and choose the point closest to q. We can intersect the ray with a line segment in $O(1)$ time, and find its first intersection with a convex arc of m vertices in time $O(\log m)$. Thus, the directional intersection-depth can be determined from an implicit representation of R in $O(n \log m)$ time.

If the minimum intersection-depth over all directions, namely $\sigma(P, Q^q)$, is desired, we need to do a little more work. We start by computing the minimum distance from q to the *explicitly stored boundary of* R; that is, we ignore the convex arcs on the boundary and find the minimum distance to the remaining edges and vertices of R. If this distance is Δ, let D be the disk of radius Δ centered on q. Let $V(q, \Delta) \subset R$ be the set of edges and vertices visible from q whose distance from q is no more than Δ; alternatively, $V(q, \Delta)$ is the intersection of the visibility polygon of q with the disk D.

Lemma 4.3 *$V(q, \Delta)$ has size $O(n+m)$, and it can be computed from the implicit representation of R in time $O(n \log(n+m) + m)$.*

Proof: We give an algorithm that computes $V(q, \Delta)$ in time $O(n \log(n+m) + m)$ using $O(n+m)$ space, which proves the lemma. We start by computing an implicit representation of the visibility polygon V of q with respect to the boundary of R. This takes $O(n \log(n+m))$ time and $O(n+m)$ space using a variant of a standard visibility polygon algorithm [3, 30], which we now describe. On each arc of Q in the implicit representation of R, there are at most two points x such that the ray $\overrightarrow{qx}$ is tangent to the arc. We find these points in $O(n \log m)$ time altogether. If we cut the boundary of R at these tangent points and at all the explicitly represented vertices, we get a collection of $O(n)$ explicit segments and implicit arcs, no two intersecting. A ray from q intersects a segment or an arc in the collection in at most one point. We use an angular-sweep method to find the visibility polygon of q. We first find the intersections of all arcs and segments with the vertical ray extending up from

q. The closest intersection to q belongs to V. Now we rotate the ray through 360° while maintaining the segment or arc closest to q as the minimum element of a priority queue. Because the segments and arcs are disjoint, the closest segment or arc changes only when the sweeping ray passes over an explicit vertex or a tangent point. The result of the sweep is a partition of the directions around q into $O(n)$ angular ranges such that within each range, exactly one segment or arc is visible from q. The algorithm takes $O(n \log(n + m))$ time altogether for tangent-finding, intersection-finding, and priority queue operations.

Once we have found V implicitly, we must refine it to get $V(q, \Delta)$. We compute Δ in $O(n)$ time. We wish to find any implicit arcs that cross D, but do not want to examine all the $O(nm)$ edges of V. Suppose that some implicit convex chain c is visible from q in an angular range (α, β). Because c bulges away from q, any edge of c that intersects the interior of D must have its normal in the range (α, β), that is, must have the same slope as a tangent to D between α and β. Thus to find any portion of c inside D, we find the extreme points of c in directions α and β, then examine the edges between to see if any intersects D. This takes $O(\log m)$ time, plus a cost proportional to the number of edges examined. There is no overlap between the edges of Q examined in any two angular ranges, and so computing $V(q, \Delta)$ takes $O(n \log(n + m) + m)$ time and $O(n + m)$ space. ▮

Once $V(q, \Delta)$ is known, the minimum distance between q and $V(q, \Delta)$ can be computed in $O(n + m)$ time. By equation (2), this distance equals the minimum intersection-depth $\sigma(P, Q^q)$. We summarize the results of this section:

Theorem 4.4 *Let P be a simple nonconvex polygon and Q a convex polygon, where $|P| = n$ and $|Q| = m$. In $O(n \log^2(n + m) + m)$ time, we can build a linear-space data structure that can answer the following queries: (1) given a placement Q^q and a direction d, determine the directional intersection-depth $\sigma_d(P, Q^q)$, and (2) given a placement Q^q, determine the minimum intersection-depth $\sigma(P, Q^q)$. The query times are $O(n \log m)$ and $O(n \log(n + m) + m)$, respectively.*

Remark. Theorem 4.4 also holds for a polygon P with holes. The construction is exactly the same: we triangulate P, compute convolved triangles, and then find the boundary of their union. Since we only require that the triangles in the triangulation of P have pairwise disjoint interiors, the time and space bounds are the same whether the polygon has holes or not.

4.3 Query Answering in Sublinear Time

In this section, we show how to preprocess R so that fixed-direction intersection-depth queries can be answered in sublinear time, for an arbitrary placement of Q. Finding $\sigma_d(P, Q^q)$ is a ray-shooting problem: we want to find the first point where the ray

$q+td$ hits the boundary of R. After some preprocessing, we can apply the ray-shooting techniques of Guibas, Overmars and Sharir [19] and Agarwal [1]. The preprocessing takes $O(n \log n + m)$ time and $O(n+m)$ space; queries take $O(n^{2/3+\delta} + \log m)$ time, for $\delta > 0$ and arbitrarily small.

For any implicit arc c, we know that the segment connecting the arc endpoints lies inside R (it lies inside the placement of Q that produced c). Furthermore, the segments for any two distinct arcs are disjoint. Each arc and its associated segment form an "ear" of the region R (see Figure 2). These ears play a critical rôle in ray-shooting: any ray that enters an ear by crossing its segment will intersect the ear's arc before hitting any other boundary point of R. The point of intersection can be found in $O(\log m)$ time.

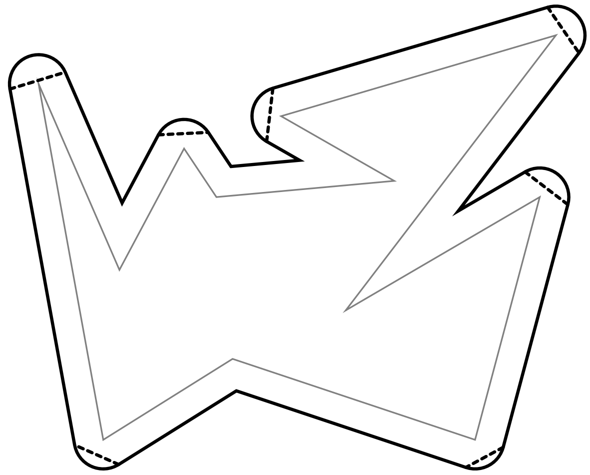

Figure 2: Ears produced by convolving a polygon with a disk

To prepare R for ray-shooting, we cut off all its ears. This leaves a polygonal region bounded by $O(n)$ line segments. We preprocess this region for ray-shooting [1, 19], which takes $O(n \log n)$ time and $O(n)$ space. We also preprocess the ears for point location; the preprocessing takes $O(n \log(n+m))$, after which a point location query can be answered in $O(\log(n+m))$ time [15].

At query time, we try to locate q in the ears. If q lies in an ear, we check whether $q+td$ hits the arc or the segment of the ear. If it hits the arc, we are done. If it hits the segment, we proceed as if q lay outside the ear at the intersection of the ray and the ear's segment. In this case, or if q lies outside all ears, we find the first explicit segment or ear segment that the ray hits, which takes $O(n^{2/3+\delta})$ time. If the ray hits an ear segment, we find the intersection of the ray with the ear's arc in $O(\log m)$ additional time. If we are willing to spend $O(n^{3/2} \log^5 n)$ time for preprocessing and $O(n \log^3 n)$ storage space, then the query time can be reduced to $O(\sqrt{n} \log^2 n)$ [1]. This solves the directional intersection-depth problem.

Theorem 4.5 *Let P be a simple polygon and let Q be a convex polygon with a total of n vertices. In $O(n \log^2 n)$ time we can build an $O(n)$ space data structure that can compute $\sigma_d(P, Q^q)$, for an arbitrary direction d and an arbitrary placement of Q, in time $O(n^{2/3+\delta})$, where δ is an arbitrarily small but positive constant. The query time can be reduced to $O(\sqrt{n} \log^2 n)$ at the expense of $O(n^{3/2} \log^5 n)$ preprocessing time and $O(n \log^3 n)$ space.*

5 Convex Polyhedra in Three Dimensions

5.1 Representation of Polytopes

Let P and Q be two convex polyhedra with a total of n vertices. We show that if P and Q are suitably preprocessed, then their directional intersection-depth can be determined in $O(\log^2 n)$ time. The representation of each polyhedron consists of an inner polyhedral hierarchy and an outer polyhedral hierarchy. These hierarchical representations were introduced by Dobkin and Kirkpatrick [11, 12, 13], who used them for determining the separation of two convex polyhedra. To describe these data structures, consider a polytope P with vertex set $V(P)$.[1] A sequence of polytopes $P_1, P_2, \ldots, P_k$ is called an *inner polyhedral hierarchy* of P if, for $1 \le i < k$,

(i) $P_1 = P$, and P_k is a simplex,

(ii) $P_{i+1} \subset P_i$ and $V(P_{i+1}) \subset V(P_i)$,

(iii) $V(P_i) \setminus V(P_{i+1})$ forms an independent set in P_i.

Thus, an inner hierarchy is a sequence of increasingly smaller polyhedral approximations of P, where each approximation is contained in its predecessor.

The *outer polyhedral hierarchy* is similar, except at each iteration we remove an independent set of faces. In particular, let $H(P)$ be the set of planes bounding the faces of P. Then the outer hierarchy of P is a sequence of polytopes $P_1, P_2, \ldots, P_k$ such that

(i) $P_1 = P$, and P_k is a simplex,

(ii) $P_{i+1} \supset P_i$ and $H(P_{i+1}) \subset H(P_i)$,

(iii) $H(P_i) \setminus H(P_{i+1})$ bounds an independent set of faces in P_i.

The *degree* of an inner polyhedral hierarchy is the maximum degree, over all i, of a vertex in $V(P_i) \setminus V(P_{i+1})$. The degree of an outer polyhedral hierarchy is analogously

[1] We limit our discussion to bounded polytopes, although all our results hold for unbounded polyhedra as well, at the expense of some minor technical complications.

defined, with faces in place of vertices. The following fact is based on the property that the facial graph of a convex polytope is a planar graph, and planar graphs admit a large independent set of small degree.

Fact 5.1 ([12]) *There is an inner (resp. outer) polyhedral hierarchy of P with height $O(\log |P|)$, size $O(|P|)$, and degree $O(1)$. Furthermore, given a standard representation of the facial graph of P (cyclic orderings of edges around the vertices and the faces), one can construct these hierarchies in linear time.*

In our discussion, we assume that each polyhedron is represented by its twin polyhedral hierarchies satisfying properties stated in Fact 5.1. In particular, a convex polytope P is represented by its inner polyhedral hierarchy $P_1, P_2, \ldots, P_k$, where each P_i is represented by an outer polyhedral hierarchy. We endow each hierarchy with some additional data to facilitate searching, as follows.

1. In an inner hierarchy, for each face F of P_{i+1} that is not a face of P_i, we store a pointer to the unique vertex v of P_i whose removal created F.

2. In an outer hierarchy, for each vertex v of P_{i+1} that is not a vertex of P_i, we store a pointer to the unique face F of P_i whose removal created v.

Using the hierarchical representation of a polytope, we can answer various extremal queries involving points, lines or planes in logarithmic time. These queries are later used by our intersection-depth algorithm. The reader may refer to the papers of Dobkin and Kirkpatrick [11, 13] or the book by Mehlhorn [27] for proofs of the following lemmas.

Lemma 5.2 *Given a directed line l, we can find its first intersection with P_i, for any $1 \leq i \leq k$, in $O(\log n)$ time.*

Lemma 5.3 *Suppose l is a line translating from infinity in direction d. We can find the first point of contact between l and P_i, for any $1 \leq i \leq k$, in $O(\log n)$ time.*

Lemma 5.4 *Suppose H is a plane translating from infinity in direction d. We can find the first point of contact between H and P_i, for any $1 \leq i \leq k$, in time $O(\log n)$.*

5.2 Computing the Intersection-Depth

We begin with the observation that if A is a convex polytope of constant complexity, then the intersection-depth $\sigma_d(A, P)$ can be computed in time $O(\log n)$. This follows because the intersection-depth $\sigma_d(x, P)$ for each element (vertex, edge or face) $x \in A$ can be computed in time $O(\log n)$ using Lemmas 5.2, 5.3, and 5.4. We therefore have the following corollary of the results of the previous subsection.

Corollary 5.5 *Let A be a convex polytope of bounded complexity and let P be a convex polytope of n vertices, given by its twin hierarchical representation. Then for any direction d, we can compute the depth of collision $\sigma_d(A, P)$ in time $O(\log n)$.*

We now describe our algorithm for computing the fixed-direction intersection-depth between P and Q. For ease of description, let us assume that the inner hierarchies of both P and Q have k levels, with $k > 1$.

Algorithm Compute-Depth-3D

Initialization. Compute the intersection between the simplices P_k and Q_k. If $P_k \cap Q_k = \emptyset$, find a separating plane H_k and go to Phase 1.

Otherwise (i.e., $P_k \cap Q_k \neq \emptyset$), compute $\sigma_d(P_k, Q_k)$; translate Q_k by $\sigma_d(P_k, Q_k)$ along direction d so that P_k and Q_k are in contact; find a plane H_k that passes through the common tangency of P_k and Q_k and separates their interiors; and go to Phase 2.

Phase 1. *(This phase maintains the invariant that the polytopes P_i and Q_i are non-intersecting and we know a witness plane H_i such that $P_i \subset H_i^+$ and $Q_i \subset H_i^-$.)*

(1A) Compute the polytopes $P'_{i-1} = P_{i-1} \cap H_i^-$ and $Q'_{i-1} = Q_{i-1} \cap H_i^+$.

(1B) Determine whether the following intersections are nonempty: $P_{i-1} \cap Q'_{i-1}$ and $P'_{i-1} \cap Q_{i-1}$. If both intersections are empty, go to Step 1C; otherwise go to Step 1D.

(1C) Find a plane H_{i-1} such that $P_{i-1} \subset H_{i-1}^+$ and $Q_{i-1} \subset H_{i-1}^-$. If $i > 2$, decrement i by 1 and go to Step 1A; otherwise, output "$\sigma_d(P, Q) = 0$" and stop.

(1D) Compute $\sigma_d(P_{i-1}, Q_{i-1}) = \max \left\{ \sigma_d(P'_{i-1}, Q_{i-1}),\ \sigma_d(P_{i-1}, Q'_{i-1}) \right\}$.

Translate Q_{i-1} by $\sigma_d(P_{i-1}, Q_{i-1})$. Find a plane H_{i-1} such that $P_{i-1} \subset H_{i-1}^+$ and $Q_{i-1} \subset H_{i-1}^-$. If $i > 2$, decrement i by 1 and go to Phase 2; otherwise, output $\sigma_d(P_1, Q_1)$ and stop.

Phase 2. *(This phase maintains the invariant that the polytopes P_i and Q_i are tangent to each other and we know a plane H_i such that $P_i \subset H_i^+$ and $Q_i \subset H_i^-$.)*

(2A) Compute the polytopes $P'_{i-1} = P_{i-1} \cap H_i^-$ and $Q'_{i-1} = Q_{i-1} \cap H_i^+$.

(2B) Compute $\Delta\sigma_d(P_{i-1}, Q_{i-1}) = \max \left\{ \sigma_d(P'_{i-1}, Q_{i-1}),\ \sigma_d(P_{i-1}, Q'_{i-1}) \right\}$.

(2C) Translate Q_{i-1} by $\Delta\sigma_d(P_{i-1}, Q_{i-1})$. Update

$$\sigma_d(P_{i-1}, Q_{i-1}) = \sigma_d(P_i, Q_i) + \Delta\sigma_d(P_{i-1}, Q_{i-1}).$$

Find a plane H_{i-1} satisfying $P_{i-1} \subset H^+_{i-1}$ and $Q_{i-1} \subset H^-_{i-1}$.
If $i > 2$, decrement i by 1 and go to Step 2A; otherwise, output $\sigma_d(P_1, Q_1)$ and stop.

Theorem 5.6 *Let P and Q be two convex polyhedra, with n and m vertices, respectively, each given by its hierarchical representation. For any direction d, the algorithm described above computes the directional intersection-depth $\sigma_d(P, Q)$ in time $O(\log n \log m)$.*

Proof: We first establish the correctness of the algorithm. The algorithm ensures that $i \geq 2$ each time step 1A or step 2A is executed, so P_{i-1} and Q_{i-1} are well defined. Phase 1 finishes when $P_i \cap Q_i \neq \emptyset$, for some i, $1 \leq i \leq k$. One easily verifies that if $P_i \cap Q_i = \emptyset$, then $P_{i-1} \cap Q_{i-1} \neq \emptyset$ if and only if one of the two pairs of polytopes in Step 1B is found to intersect. If neither of the pairs intersects, then one can also find a plane H_{i-1} that separates P_{i-1} from Q_{i-1}. (There are several ways to find such a separating plane. One possibility is to use the perpendicular bisector of the closest pair of P_{i-1} and Q_{i-1}.) This proves the correctness of Phase 1.

In Phase 2, we maintain the invariant that P_i and Q_i are in contact, and a plane H_i separating their interiors is known. Suppose that we enter an iteration of this phase with P_i and Q_i. It is easy to see that only the polytopes P'_{i-1} and Q'_{i-1} are of interest for determining whether P_{i-1} and Q_{i-1} intersect. Furthermore, $\Delta\sigma_d(P_{i-1}, Q_{i-1})$ correctly reflects the amount by which Q_{i-1} must be translated to make its interior disjoint from P_{i-1}. We find a plane H_{i-1}, which can be determined easily using the contact point of the two polytopes, and start the next iteration. The invariant is maintained and correctness of Phase 2 has been established.

Finally, we show that each iteration of Phase 1 or Phase 2 can be completed in $O(\log n)$ time. The basic step of computing the polytopes P'_{i-1} and Q'_{i-1} in Steps 1A and 2A takes $O(\log n)$ time, as follows. By the convexity of P_{i-1}, at most one of its vertices lies in H^-. If there is a vertex $v \in V(P_{i-1})$ lying in H^-, then v also forms the first point of contact between P_{i-1} and a plane moving parallel to H_i from infinity. By Lemma 5.4, we can find v, and thus P'_{i-1}, in time $O(\log n)$. The remaining steps take $O(\log n)$ time by Corollary 5.5, since both P'_{i-1} and Q'_{i-1} have bounded complexity. (The logarithmic time bound per step also suffices for maintaining the closest pair (and hence the separating plane in Phase 1) of P_i and Q_i.) This completes the proof of the theorem. ∎

6 Extensions, Related Problems and Conclusions

The intersection-depth is a reasonable measure of negative distance between intersecting objects. In many realistic situations, however, some additional factors might come into play. Practical constraints might require that all translations be along a fixed set of directions. This restriction simplifies the query version of the intersection-depth problem. Given a non-convex polygon P and a convex polygon Q, for each direction d, we perform an (implicit) trapezoidal decomposition of the convolution-space R, and preprocess the resulting subdivision for point location. Given a placement of Q^q and direction d, we locate q in the subdivision of R corresponding to d and compute $\sigma_d(P, Q^y)$ by intersecting the ray $q + td$ with the sides of the trapezoid containing q. The procedure takes $O(\log n)$ time.

In many robotic situations, one may prefer a *complete separation* of the two polygons. Given two intersecting polygons P and Q and a direction d, translating Q by $\sigma_d(P, Q)$ ensures that P and Q no longer intersect; however, Q may still be stuck in one of the "pockets" of P and hence cannot be moved arbitrarily far. By the *complete separation* of P and Q in direction d we mean the smallest translation that moves Q to a position from where it can be translated arbitrarily far along d without intersecting P. In terms of the convolution it requires finding a point in the infinite face of the complement of $P \oplus (-Q)$. This problem also appears to be easier than computing σ_d and σ. We maintain an implicit representation of only the outermost boundary of $R = P \oplus (-Q)$. After linear-time preprocessing, the complete separation in a fixed direction can be computed in $O(\log n)$ time by performing a ray-shooting query in a simple polygon.

If P and Q are both non-convex, then we can find their complete separation in a given direction in linear time. We first translate Q sufficiently far along the given direction to ensure that it is disjoint from P; this can be accomplished by determining the extreme vertices of P and Q in the given direction. Next we compute the "horizontal" visibility profile of Q with respect to P and vice versa (by choosing the fixed direction to be the horizontal direction); the horizontal visibility profile computes for each vertex $q \in Q$ the edge of P that is hit by the horizontal ray issuing from q. The horizontal separation can be inferred from the visibility profile, and by subtracting it from the initial displacement we arrive at the complete separation. The procedure runs in linear time since we can compute the horizontal visibility profile in $O(n)$ time using a standard visibility-polygon algorithm [16]. (The key observation here is that since P and Q are separated by a vertical line, it suffices to consider the horizontal visibility profile of only the left and right envelopes of P and Q; the vertices not on the envelopes don't matter. To compute the visibility profile, we merge the two envelopes, which takes linear time. Computing the envelopes also takes linear time, since an envelope of a polygon is the same as its (external) visibility polygon from the point $x = \infty$.) In contrast, we know of no $o(n^2)$ time algorithm for determining the fixed-direction intersection-depth of two non-convex polygons.

Finally, the minimum intersection-depth between two three-dimensional convex polyhedra P and Q^q can be computed in $O(n^2)$ time by computing their convolution explicitly and then finding the boundary point closest to q.

Our work suggests several open problems. Can the fixed-direction intersection-depth of two non-convex polygons be computed in substantially better than $O(n^2)$ time? How fast can the minimum intersection-depth be computed for two nonconvex polygons? An $O(n^4)$ time bound can be obtained by explicitly computing the convolution of the two polygons. (The convolution of two n-vertex polygons can have $\Omega(n^4)$ size in the worst case [4], and so a sub-quartic algorithm must avoid building the entire convolution.) Can some nontrivial lower bounds be proved for these problems?

For two convex polyhedra P and Q in three dimensions, we gave an $O(\log^2 n)$ time procedure for determining the directional depth of intersection. Can the time bound be improved to $O(\log n)$? We can compute the minimum intersection-depth of two convex polyhedra by computing their convolution explicitly in $O(n^2)$ time. Is it possible to do substantially better? Extensions involving nonconvex polyhedra are also worth investigating.

References

[1] P. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *Proc. of 5th ACM Symposium on Computational Geometry,* 315–325, 1989.

[2] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989.

[3] T. Asano. An efficient algorithm for finding the visibility polygons for a polygonal region with holes. *Transactions of IECE of Japan*, E-68:557–559, 1985.

[4] B. S. Baker, S. F. Fortune and S. R. Mahaney. Polygon containment under translation. *Journal of Algorithms,* 532–548, 1986.

[5] J. Bentley and T. A. Ottman. Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers,* C-28:643–647, 1979.

[6] C. E. Buckley and L. J. Leifer. A proximity metric for continuum path planning. *Proc. 9th International Joint Conference on Artificial Intelligence,* 1096–1102, 1985.

[7] S. A. Cameron and R. K. Culley. Determining the minimum translation distance between two convex polyhedra. *Proc. IEEE International Conference on Robotics and Automation,* 591–596, 1986.

[8] B. Chazelle. Efficient polygon triangulation. *Proc. of IEEE Symp. on Foundations of Computer Science*, 220–230, 1990.

[9] B. Chazelle and D. Dobkin. Intersection of convex objects in two and three dimensions. *J. of ACM*, 34:1–27, 1987.

[10] F. Chin and C. A. Wang. Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE Transactions on Computers*, C-32:1203–1207, 1983.

[11] D. Dobkin and D. Kirkpatrick. Fast detection of polyhedral intersections. *Proc. of ICALP '82*, 154–165, 1982. Lecture Notes in Computer Science 140.

[12] D. Dobkin and D. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. of Algorithms*, 6:381–392, 1985.

[13] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach. *Proc. of ICALP '90*, 400–413, 1990. Lecture Notes in Computer Science 443.

[14] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. of Algorithms*, 6:213–224, 1985.

[15] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[16] H. El Gindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *J. of Algorithms*, 2:186–197, 1981.

[17] M. Garey, D. S. Johnson, F. P. Preparata and R. E. Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7:175–179, 1978.

[18] B. Grünbaum. *Convex Polytopes.* John Wiley and Sons, Ltd. 1967.

[19] L. Guibas, M. Overmars, and M. Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. of the First Scandinavian Workshop on Algorithm Theory*, pages 64–73. Springer-Verlag, 1988. Lecture Notes in Computer Science 318.

[20] L. Guibas, L. Ramshaw and J. Stolfi. A kinetic framework for computational geometry. *Proc. 24th Foundations of Computer Science*, 100–111, Nov. 1983.

[21] L. J. Guibas and J. Stolfi. Ruler, compass, and computer: The design and analysis of geometric algorithms. Research Report 37, DEC Systems Research Center, 1989. Also appeared in *Theoretical Foundations of Computer Graphics and CAD*, Springer-Verlag.

[22] J. Hershberger and L. Guibas. An $O(n^2)$ shortest path algorithm for a non-rotating convex body. *Journal of Algorithms*, 9:18–46, 1988.

[23] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. *Lecture Notes in Computer Science,* 158, 207–218, 1983.

[24] K. Kedem, R. Livne, J. Pach and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete and Computational Geometry*, 1:59–71, 1986.

[25] S. S. Keerthi and K. Sridharan. Efficient algorithms for computing two measures of depth of collision between convex polygons. Technical Report, Department of Computer Science and Automation, IIS, Banglore, India, 1989.

[26] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.

[27] K. Mehlhorn. *Multi-dimensional Searching and Computational Geometry*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1984.

[28] T. A. Ottman, P. Widmeyer and D. Wood. A fast algorithm for Boolean mask operations. Inst. f. Angewandte Mathematik und Formale Beschreibungsverfahren, D-7500 Karlsruhe, Report no. 112, 1982.

[29] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[30] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. *Proc. of 2nd ACM Symposium on Computational Geometry,* 14–23, 1986.