MOST OF THESE RESULTS ALSO APPEAR IN: MITCHELL, MOUNT, & PAPADIMITRIOU, "THE DISCRETE GEODESIC PROBLEM" SIAM J. COMPUT. 6 (1987) 647-668.

CAR-TR-121
CS-TR-1496

F-49620-83-C-0082
May 1985

# VORONOI DIAGRAMS ON THE SURFACE OF A POLYHEDRON

David M. Mount

Department of Computer Science
University of Maryland
College Park, MD 20742

## ABSTRACT

We present an algorithm that computes the Voronoi diagram of a set of points lying on the surface of a possibly nonconvex polyhedron. Distances are measured in the Euclidean metric along the surface of the polyhedron. The algorithm runs in $O(n^2 \log n)$ time and requires $O(n^2)$ space to store the final data structure, where n is the maximum of the number of edges and source points on the polyhedron. This algorithm generalizes or improves the running times of a number of shortest path problems both on polyhedra and in the plane amidst polygonal obstacles. By applying standard algorithms for point location, we can determine the distance from a query point to the nearest source in $O(\log n)$ time and can list the shortest path in $O(k + \log n)$ time, where k is the number of faces traversed by the path. The algorithm achieves its efficiency by a novel method of searching the polyhedron's surface.

---

The support of the Air Force Office of Scientific Research under Contract F-49620-83-C-0082 is gratefully acknowledged.

## 1. Introduction

Because of its application to robotics and terrain navigation, there is interest in efficient algorithms for finding shortest paths amidst obstacles in both 2 and 3 dimensions. An $O(n^2 \log n)$ algorithm is known for finding shortest paths between points on a plane amidst polygonal obstacles by constructing the *visibility graph* [LW79, Le78, SS84], and $O(n \log n)$ is attainable in special cases when shortest paths possess certain monotonicity properties [LP84, SS84]. Sharir and Schorr present an $O(n^3 \log n)$ algorithm for finding shortest paths on convex polyhedra [SS84], which was subsequently improved to $O(n^2 \log n)$ [Mo84]. O'Rourke, Suri and Booth give an $O(n^5)$ algorithm for finding shortest path between a pair of points on the surface of a nonconvex polyhedron where the path lies on the surface of the polyhedron [OS84].

We present an algorithm which when given a possibly nonconvex polyhedron and a finite number of source points on the surface of the polyhedron, partitions each face of the polyhedron into Voronoi cells. A point x is a member of a Voronoi cell for a given source s if the Euclidean distance from s to x measured along the surface of the polyhedron is not greater than the distance from any other source. In fact, our algorithm produces a refinement of the Voronoi diagram from which the actual shortest path can easily be determined. This generalizes and improves on the results of O'Rourke, Suri and Booth. Notice this measure of distance may not be the shortest distance in 3-space, since we are constrained to travel along the surface of the polyhedron. However, this form of the shortest path problem is of interest in terrain navigation, where a moving vehicle is bound move along a surface that could be modeled by a polyhedron. An $O(n^2 \log n)$ algorithm for the single source shortest path problem appears in a companion paper [MM85].

The running time of our algorithm is $O(n^2 \log n)$ where n is the maximum of the number of polyhedron edges and source points. (Typically the number of source points will be small relative to the size of the polyhedron.) The output of the program can be stored in $O(n^2)$ space. With the aid of standard algorithms for point location [Ki83, Pr81, Co83], distance queries can be answered in $O(\log n)$ time, and the shortest path can be listed in $O(k + \log n)$ time, where k is the number of faces traversed by the path.

A polyhedron is considered to be a set of simple polygonal faces and an adjacency relation that connects a pair of faces through a common edge. Each edge is incident on exactly two faces. Note that the angle formed between adjacent faces is of no relevance to surface distances, and hence need not be specified. The generalization of the algorithm to unbounded faces is straightforward, but not discussed here. We do not require that the polyhedron is realizable as a rigid object in 3-space, or even that the polyhedron can be embedded on an orientable surface. For the expected domain of the problem, for

example CAD representations of 3-dimensional objects, the genus of the polyhedron is typically much less than linear in the number of its vertices. In this case, it follows form Euler's formula that the number of edges in the polyhedron is linear in the number of vertices.

The problem of finding shortest paths on the plane while avoiding polygonal obstacles can be reduced to the problem of finding shortest paths on a polyhedron by partitioning the plane into polygons and replacing obstacle boundaries by sufficiently high vertical faces. Hence, our algorithm generalizes the results of [Le78, SS84] for planar shortest paths. However, our algorithm is sufficiently complex that it would not be recommended as a practical solution to this problem.

Our algorithm is a generalization of the techniques used by Sharir and Schorr [SS84] called slices, which we generalize in Section 3 by defining *wedges*. Sharir and Schorr's method in turn is a generalization of Dijkstra's algorithm for finding shortest paths in graphs [AH74]. The faces of the polyhedron are analogous to the vertices of the graph in Dijkstra's algorithm, but unlike Dijkstra's algorithm, the distance to a given face is not a unique scalar. The purpose of a wedge is to define a region of a face within which the shortest paths have the same discrete properties, that is, they originate from the same source and traverse the same sequence of edges and vertices.

A straightforward generalization of Sharir and Schorr's or Mount's algorithm for convex polyhedra leads to an $O(n^3)$ algorithm in the nonconvex case. In the convex case wedges are bounded by straight line segments and are convex. The wedges arising for nonconvex polyhedra are bounded by hyperbolic segments and are nonconvex. The absence of convexity considerably complicates the procedure. However, an analogy can be drawn with a similar problem in 2-space. Lee and Preparata observed that the shortest path problem can be considerably simplified if the paths are shown to be monotonic, that is, they travel in essentially one direction [LP84]. We say that paths on a polyhedron are monotonic if each edge is traversed in only one direction. Shortest paths on a polyhedron may not be monotonic, but we can mimic monotonicity by imagining that each edge is split into two copies, one copy is associated with the paths crossing the edge from the left, and the other with paths crossing from the right. This technique, which we called *structured monotonicity*, increases the number of wedges somewhat but significantly decreases the processing time for each wedge.

After explaining notation and representational preliminaries in Section 2, we describe structured monotonicity in Section 3. In Section 4 we describe the algorithm for the monotonic case, and in Section 5 describes how to convert the monotonic solution to a general solution.

## 2. Preliminaries

The polyhedron consists of a set of vertices V, edges E and faces F. S denotes the set of source points. We assume that the polyhedron is represented so that standard enumeration tasks can be performed efficiently, such as cyclically listing the vertices and edges incident on a face [PM83]. For the purpose of simplifying our exposition we make two additional assumptions. First, we assume that the faces of the polyhedron have been triangulated. This can be performed in O(m log m) time per face where m is the degree of the face [GJ78]. Second, we assume that the source points lie on (possibly trivial) vertices of the polyhedron. This can be accomplished by adding new edges that connect each source with the vertices of its enclosing face. Thus we have $S \subseteq V$, and all the sets V, E and F are O(n) in size.

Each face of the polyhedron is associated with a 2-dimensional coordinate system. If a face $f_2$ is adjacent to a face $f_1$ at an edge e, then the length of a path passing from $f_1$ to $f_2$ is unaffected by the angle formed between the faces at e. It is easiest to view a path between two faces by unfolding the faces about e so that they lie on a common plane. This operation of unfolding is called the *planar unfolding* of $f_1$ and $f_2$, and can be generalized to a chain of faces. To facilitate distance computation, shortest paths are typically represented by their planar unfolding with respect to their destination. The conversion from the planar unfolding to a path on the surface of the polyhedron is straightforward.

As mentioned earlier, when considering the monotonic form of the shortest path problem, an edge e incident on two faces $f_1$ and $f_2$ is treated as two separate objects. In this case, we distinguish between an edge e on $f_1$, which carries paths from $f_1$ to $f_2$ and the complement edge e on $f_2$ that carries paths from $f_2$ to $f_1$. Paths are considered to be directed outwards from the source points.

The line segment between two points on the plane x and y is denoted xy and the Euclidean length of this path is denoted |xy|. We will be considering *weighted points*, that is, a point x and an associated nonnegative real number d. The *weighted distance* is the distance to such a point plus the associated weight.

## 3. Overview of the Algorithm

A *geodesic* is a path on the polyhedron that is locally a shortest path. Shortest paths are all geodesics, but the converse need not hold. The following properties of shortest paths and geodesics of finite length are easy to verify.

(1) The restriction of a shortest path to a face is a single line segment. The restriction of a geodesic to a face is a set of line segments.

(2) A geodesic passes through the interior of an edge so that the planar unfolding of the shortest path is a line.

To see how geodesics traverse vertices, consider a vertex v on the polyhedron and let $f_1, f_2, \ldots, f_k$ be the faces incident on v. Let $\theta_i$ be the angle formed by the edges incident on v in $f_i$, and let $\theta$ be the sum of these angles. The angle $\alpha$ formed by a path passing through v can be defined to be the minimum of the sums of angles clockwise about v and counterclockwise about v (see Fig. 3.1a).

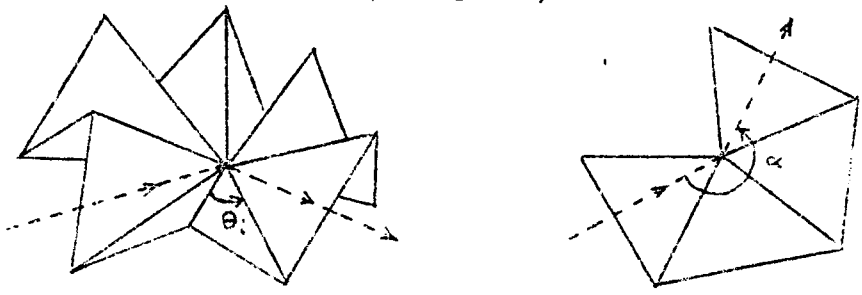


Fig. 3.1

This angle is at most $\theta/2$. If $\alpha$ is less than $\pi$, then by unfolding the faces on this side of v, we can easily find a shorter path in some neighborhood about v. From this observation we have

(3) A geodesic traversing a vertex forms an angle about the vertex that is greater than or equal to $\pi$.

These three properties fully characterize the geodesics, but for our purposes, it will simplify the presentation to weaken the definition. We define a geodesic as a path satisfying properties (1) and (2), but allow them to bend at arbitrary angles as they traverse vertices. Note that in the case of convex polyhedra, the sum of angles about any (non-trivial) vertex is strictly less than $2\pi$, and hence (3) reduces to the statement that geodesics do not pass through vertices [cf. SS84].

The structure of the algorithm is the same as Dijkstra's algorithm—a priority queue is maintained and events are processed in order of increasing distance, until no events remain. To represent the distance from the nearest source to some region of a face, notice that the shortest path to a point is determined by the sequence of edges and vertices through which the path travels. Since source points are themselves vertices, this sequence determines the starting source point. We may partition each face of the polyhedron into regions within which shortest paths traverse the same sequence of discrete components. We will show that there are $O(n^2)$ regions. The final output of our algorithm is this partition. The Voronoi cell for each source is the disjoint union of of the regions originating from that source.

The first observation needed in developing an efficient algorithm is that, for the purposes of Dijkstra's algorithm, we only need to maintain the distance to each vertex and edge, and not to each point on the interior of the face. This reduces a 2-dimensional problem to a simpler 1-dimensional problem.

To see how to succinctly represent the distance to an edge, consider a subset of points b on an edge e within which the shortest paths traverse the same sequence of edges and vertices. Consider the planar unfolding of this sequence with respect to e. Let o be the last vertex on the on this sequence before arriving at e (see Fig. 3.2a). Note that the rays from o to b pass through the interior of the subsequent edges. By observations (1) and (2), the planar unfolding of the shortest path to a point x on b is the shortest path from the source to o followed by the straight line joining o to x. Let d be the length of the shortest path from the source to the o. The distance from the source to x is just $|ox| + d$, which is the distance from x to the point o weighted by d. For the purpose of defining the distance from the source to x, we only need to record the position of the *origin* o relative to e, the *base* b, and the weight d. We define a *wedge* w to be the quadruple $<o,b,d,e>$. A wedge is *connected* if its base is connected. A wedge is *linear* if passes through a vertex of e. Wedges are related to the notion of funnels appearing in [LP84].

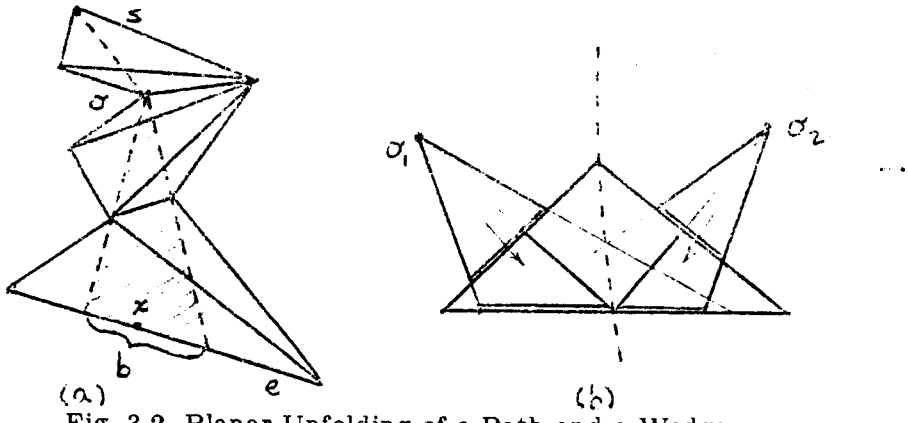


Fig. 3.2 Planar Unfolding of a Path and a Wedge

The algorithm extends wedges from face to face by extending the rays bounding the wedge at its origin. When wedges overlap the algorithm trims their bases by their proximity to the nearer origin (see Fig 3.2b). The boundary between two wedges is defined by the bisector between the wedge origins (where distances are weighted). Thus this bisector is a hyperbolic segment that may degenerate to a line segment (when origin weights are equal) or may be empty (when the difference in origin weights exceeds the distance between origins).

The fact that the bisector is a hyperbola, implies that the resulting regions are not convex, and hence, the base of a given wedge may not be connected. Cases can be

constructed in which wedges have bases with O(n) components. This significantly complicates any algorithm that explicitly constructs the bases. To overcome this difficulty we solve the following modification of the shortest path problem:

Monotonic Shortest Path Problem:

> For each point x on an edge e on a face f, determine the shortest geodesic from a source to x that approaches x from within f.

In this definition we allow the geodesic to pass through a vertex of e. For example, in Fig. 3.3, the paths from $s_1$ and $s_2$ to x are considered in the monotonic problem for e on $f_1$. The paths from $s_2$ and $s_3$ would be considered in the monotonic problem for e on $f_2$.

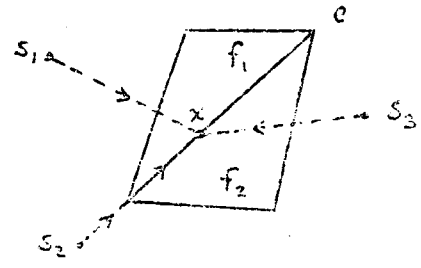


Fig. 3.3 Monotonic Shortest Paths

Since each edge lies on two faces, once the monotonic problem has been solved, we can determine true shortest path to x by taking the shorter of the two monotonic paths.

One way to visualize the monotonic paths is to imagine that a barrier is placed along the interior portion of each edge. The monotonic shortest path to a point x on edge e is the shortest path that is allowed to pass through every barrier except the barrier on e. It is easy to prove that this path will be a geodesic under our definition. The monotonic shortest path to a vertex is the true shortest path. If x lies on e at distance $d_1$ from a vertex of e, and e is distance $d_2$ from the nearest source, then with our weaker definition of a geodesic there is a geodesic to x of length at most $d_1 + d_2$. Thus the monotonic distance to every point on an edge is finite, and the distance function is continuous. The motivation behind defining the monotonic problem is given in the next lemma.

**Lemma 3.1** The bases of wedges arising in the monotonic problem are connected.

**Proof**

The proof is by induction on the number of edges traversed by the wedge. A degenerate wedge whose base equals its origin is trivially connected. Consider a wedge whose base lies on an edge e on face f. The wedge enters the face through one of f's other edges or vertices. The intersection of the wedge with this region of entry is a subset of a connected base by induction. At most one wedge on e can enter through this region since wedge bases do not overlap.

Suppose to the contrary, that the wedge is not connected. From the above comments and the fact that wedges cover e, there must be two wedges whose planar unfoldings intersect each other (see Fig. 3.4).

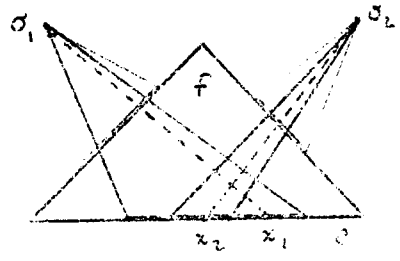


Fig. 3.4

Since f is convex, this intersection occurs within f. There exist points $x_1$ and $x_2$ lying on the bases of the intersecting wedges having origins $o_1$ and $o_2$ respectively whose shortest geodesics intersect within f at some nonzero angle. However, this leads to an immediate contradiction by constructing shorter paths through f from $x_1$ to $o_2$ and vice versa. This implies that there exists a shorter geodesic that approaches x from within f.

□

By introducing structured monotonicity we have simplified the structure of the wedges, but at what cost to the number of wedges that need to be processed? Our next lemma establishes that the total number of wedges is essentially quadratic in the size of the polyhedron, which is true for the general case also.

**Lemma 3.2** The number of wedges incident on a given edge in the monotonic problem is O(n), and hence the total number of wedges is $O(n^2)$.

**Proof**

Consider the set of wedges incident on one side of an edge e. Each wedge origin is a vertex. By arbitrarily selecting one of possibly many shortest paths to a vertex (as our algorithm does) we may consider the path to be unique. Thus, if two distinct wedges

share a common origin, then somewhere between the common origin and e the wedges pass on different sides of a vertex v after passing through a face f incident on v. No other pairs of wedges sharing a common origin can be first separated by v while passing through f, since this would imply that two shortest geodesics cross each other at some nonzero angle within f. In Lemma 3.1 we showed that this cannot happen.

It follows that the number of wedges incident on e is bounded by the number of origins plus the number of pairs (v,f) where v is incident on f. The number of origins is bounded by the number of vertices, and the number of pairs (v,f) is bounded by twice the number of edges. Thus there are O(n) wedges on e.

□

## 4. Monotonic Shortest Path Algorithm

In this section we describe the algorithm for the monotonic shortest path problem. The algorithm is based on Dijkstra's algorithm, where wedges are used to store the distance function.

The algorithm is controlled by a priority queue with entries ordered by distance. We assume that the operations insert, extract-min, delete and change-priority can be performed in O(log n) time each [AH74]. We refer to queue entries as *events*. We distinguish two types events: *edge-events* correspond to the extension of a wedge across one edge to another and *vertex-events* correspond to the arrival of a wedge at a vertex. The associated distance of an edge-event is the least weighted distance from the wedge origin to its base, and the distance of a vertex event is the weighted distance from the origin to the vertex.

Initially the priority queue contains a vertex-event at distance 0 for each of the source points. For each of the remaining vertices a vertex-event is enqueued at an infinite distance. As our knowledge of the distance to the vertex improves, the priority of this entry is updated.

Consider the case when a vertex-event is removed from the queue. For each face incident on the vertex we create a new wedge for each of the edges of this face. The base of each of these wedges is the edge itself. For edges incident on v this is a linear wedge. We proceed to the trimming procedure described below. (In the case we have arrived at the vertex from a particular direction, in contrast to the case that the vertex is a source point, then we could limit the extension to the angles given in observation (3). This improvement does not affect the asymptotic running time of the algorithm, but simplifies the processing of convex vertices.)

When an edge-event is removed from the queue a new wedge extending the current wedge is generated. If the originating wedge is incident on an edge e on a face $f_1$, and $f_2$ is the opposing face on e then new wedges are generated for the other edges on $f_2$ (see Fig. 4.1). The original wedge is the *parent* of the generated wedge. If the bases of the new wedges overlap with existing wedges, the wedge is *trimmed.*

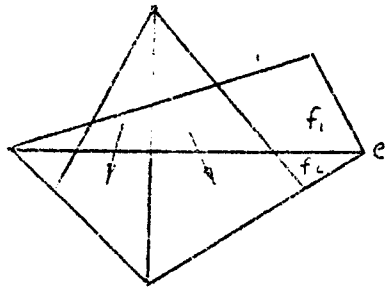


Fig. 4.1 Wedge Extension

Let e be the edge on which the trimming take place. The trimming process begins by determining the relative position of the newly created wedge w with respect to the other wedges on e. Because wedges incident on e do not cross over each other within f, these wedges are naturally ordered as their parent wedges are ordered around f. This ordering allows us to find the relative position of the new wedge in O(log n) time by bisection. Let $w_L$ and $w_R$ be the wedges to the left and right of w, and let z be a point between the bases of $w_L$ and $w_R$. By working outward to the left and right along e, we trim back neighboring wedges.

For concreteness consider the case of trimming in the leftward direction (see Fig. 4.2). The extent of the trimming is limited *a priori* to the base of the newly extended wedge. We compute the set of points on e that are closer to one wedge origin than another (weighted distance) by computing the intersection of e and the bisecting hyperbola defined by the origins. If every point on the base of $w_L$ is closer to the origin of w than to the origin of $w_L$ then $w_L$ is deleted. We let $w_L$ be the next wedge to the left and repeat the process. Otherwise, find the point x on the base of $w_L$ such that every point to the right of x is closer to w's origin than to $w_L$'s origin. The neighboring base is trimmed so that it contains only the points that lie to the left of x. The left trimming process terminates at this point and the region between x and z is made part of w's base. Notice that the point x is equidistant from the two wedge origins between which it lies, and wedge bases remain connected. Finally, if priority queue entries depended on the points of a wedge that were deleted or trimmed, then these entries are reevaluated.

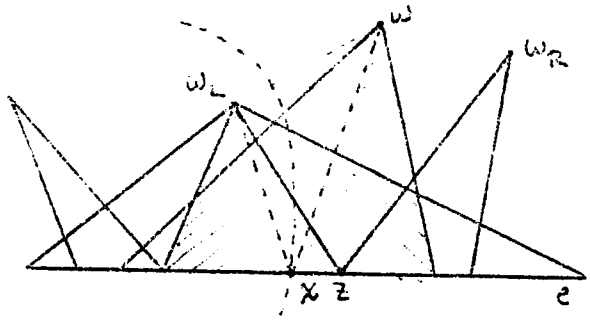


Fig. 4.2 Wedge Trimming

After the new wedge has been established and assuming that it is nonempty, the following priority queue entries are created. The closest point in the wedge base to the origin is enqueued with its weighted distance to the origin. This event is an edge-event. If the wedge base contains any vertices and the weighted distance from the vertex to the wedge origin is less than the best known distance to the vertex, then the distance to the vertex is updated, and the priority queue entry associated with the vertex is updated with this smaller distance. Note that there is only one priority queue entry for each vertex and its distance never increases.

Once the relative position of the new wedge has been determined, the cost of trimming wedges is proportional to the number of wedges deleted. This is turn is bounded by the number of wedges created, which we will show to be $O(n^2)$ overall. Modifying priority queue entries may require $O(n^2 \log n)$ time overall. It follows that the overall cost of trimming is $O(n^2 \log n)$.

The complexity and correctness of the algorithm are derived from the following invariant:

**Lemma 4.1** Let d be the distance of the next event in the priority queue. Then

(1) (wedges correspond to geodesics) if a wedge w is created by the algorithm, and a point x lies on w's base, then there is a geodesic to x from some source whose distance is the weighted distance from x to the origin of w,

(2) (wedges cover all points up to distance d) for a point x which is at distance $\hat{d} < d$ from the nearest source and which is on the interior of an edge e, there is a wedge w whose base contains x such that the weighted distance from x to the origin of w is $\hat{d}$, and w gives the planar unfolding of the shortest path to x,

(3) (only true wedges are extended) if a wedge w has been extended through an edge e, and x is the closest point on the base of w to its origin at the time of extension, then w is the planar unfolding of the shorest geodesic to x from this side of e, and

(4) (shortest paths to vertices are correct up to distance d) if a vertex v has been labeled with distance $\hat{d} \leq d$ then the distance from the nearest source to v is $\hat{d}$.

**Proof**

The proof is by induction on the number of entries extracted from the priority queue and noting that priority queue entries are extracted in nondecreasing order of distance. Item (1) follows simply from the fact that wedges are extended by unfolding faces, just as geodesics are.

To show (2) consider such a point x. The point y at which the shortest path from x to the nearest source intersects another edge or vertex is at a strictly closer distance to the nearest source. By induction, some wedge contains the point y. Since the distance to y is less than d this wedge has already been extended. Thus, some wedge contains x. By (1), this wedge must correspond to a true geodesic, and since the path through y is hypothesized to be the shortest geodesic, the wedge extended through y must contain x.

To show (3), suppose that x is the closest point on edge e to the wedge origin at the time of extension. By (1) there exists a path from x to some source whose distance is the weighted distance from x to this origin. By (2) there is no shorter path to any source. Thus there is at least one point of the wedge that will survive trimming, and so the wedge is permanent.

Item (4) follows from (1) and (2) and continuity of distances.

□

The correctness of the algorithm follows from (2) and (4). The complexity of the algorithm follows from the remarks made earlier on the total number of wedges, together with (3) that shows that no effort is wasted extending wedges that are eventually deleted.

## 5. Converting the Monotonic Solution to a Voronoi Diagram

In this section we extend the monotonic shortest path information for the edges into a Voronoi diagram on each face. Since source points are located at vertices, the shortest path to a point within a face f must pass through a vertex or an edge of f. Thus each Voronoi cell within f intersects the boundary of f. Because distances are weighted, this problem is closely related to the problem of computing generalized Voronoi diagrams [LD81, Ki79] and in particular computing Voronoi diagrams for circles [Sh85]. Our problem is not directly reducible to any existing Voronoi diagram problems because

(1) the restricting nature of a wedge implies that the shortest path to a wedge origin is not necessarily a straight line, and

(2) the distances already computed for each edge provide significant additional information.

However, our techniques are essentially the same as existing methods for computing Voronoi diagrams for circles. Our primary task is to show that the essential properties of circle Voronoi diagrams hold in our case. The presentation will largely follow that of Sharir [Sh85]. Unlike Sharir's algorithm which runs in $O(n \log^2 n)$ time, ours requires only in $O(n \log n)$ time due to observation (2). After applying our procedure to each of $O(n)$ faces we have $O(n^2 \log n)$ total complexity.

Consider face f surrounded by the edges $e_1$, $e_2$ and $e_3$. All shortest paths to points in f are conveyed by the wedges arising from monotonic shortest paths to $e_i$ passing through the adjacent face or by the vertices of f (which can be thought of as degenerate wedges whose origin and base coincide at a single point). The bases of these wedges partition the boundary of f. Let $W = \{w_1, w_2, \ldots\}$ be this set of $O(n)$ wedges ordered cyclically about f. Except for its base, each wedge lies exterior to f. Define the distance from a point x in f to a wedge origin $o_i$ to be the length of the shortest path starting at x, then passing through the base of $w_i$, then to $o_i$ plus the weight of $o_i$. This path either consists of a single straight line segment from x to $o_i$ or else a pair line segments, one from x to an endpoint of the base of $w_i$ and another from this endpoint to $o_i$. The Voronoi cell of a wedge $w_i$, $V_i$, is the set of points of f that are as close to $o_i$ as to any origin (given this modified definition of distance) (see Fig. 5.1). The Voronoi region of a subset of wedges S, V(S), is the union of the Voronoi cells of S.

Let $Z = \{z_1, z_2, \ldots\}$ be the $O(n)$ wedge endpoints. For each such point z lying between two wedges $w_1$ and $w_2$, the continuity of distances implies that the weighted distances from z to the corresponding origins, $o_1$ and $o_2$, are equal. That is, z lies on the bisector between the origins. Call this (weighted) distance the weight of a point in Z. We enlarge the set W by creating new (trivial) wedges without lateral boundaries whose origins and bases are the points of Z. We call these *end-wedges*.

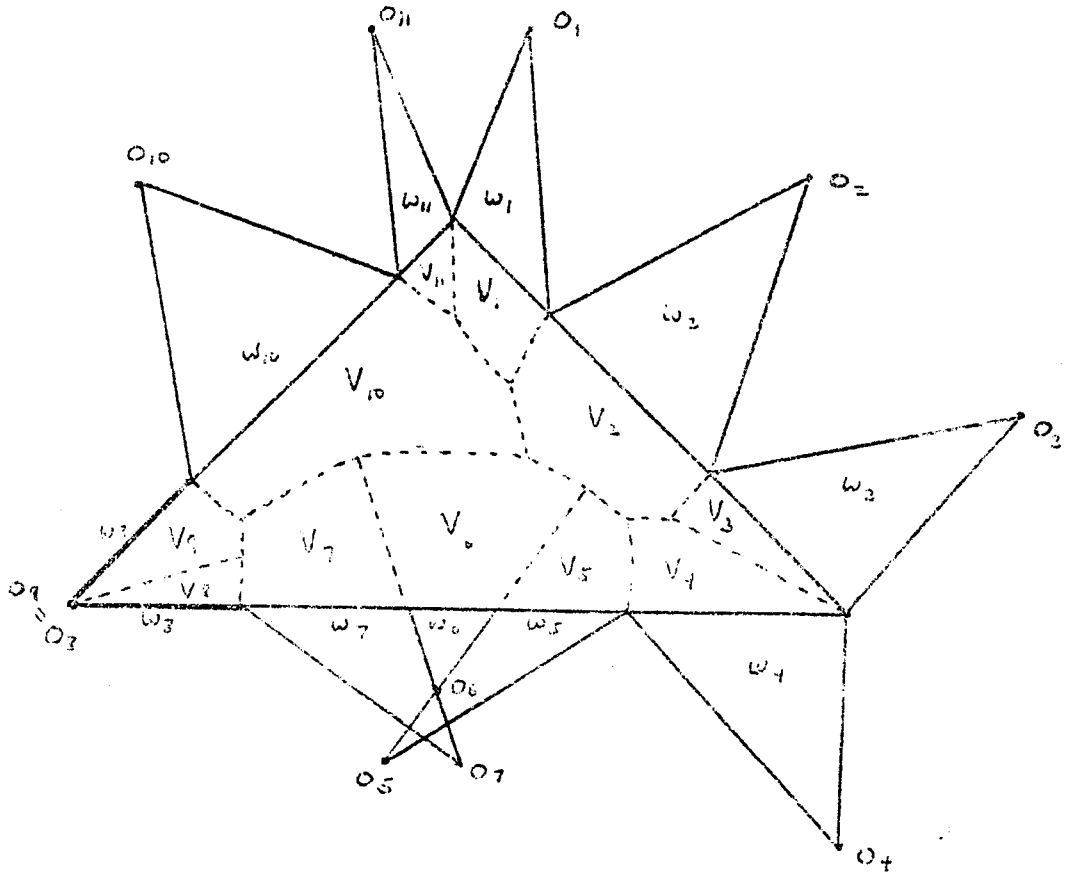

Fig. 5.1 Voronoi Diagram with Wedges

Consider a subset of wedges $S \subseteq W$ such that if $w_i$ is in S, then its associated end-wedges are also in S. Because of the end-wedges, the distance from a point x to the nearest origin in S is a straight line. The distance from x to the nearest origin is no less after the inclusion of the end-wedges, thus these wedges do not affect the lengths of shortest paths. (In fact, in the final diagram the cells of end-wedges will be empty because shortest paths are geodesics and hence pass through the edge in a straight line. The end-wedges serve a function simply in guaranteeing that intermediate Voronoi diagrams constructed from subsets of W have a nice structure.)

We employ the standard divide-and-conquer technique for computing the Voronoi diagram [SH75, Sh85]. At each stage we compute the Voronoi diagram of a set of wedges. This set is split into two subsets L and R of approximately equal size. Of course, there is no global notion of "left" and "right" assumed here; it is convenient to use these terms to describe the algorithm. The original set S and the subsets L and R satisfy the following properties:

(i) they contain the end-wedges associated with each wedge base,

(ii) the bases form a connected subchain about f,

(iii) (for L and R) either all of the wedges lie on a single edge of f, or the set of edges on which L lie and the set of edges on which R lie are disjoint.

It is easy to see that a divide-and-conquer algorithm can be designed that satisfies these conditions. For example, at the highest level of the recursion the set L consist of the wedges of $e_1$, and R consists of the wedges of $e_2$ and $e_3$. At the second level of the recursion R is split into two sets consisting of the wedges of $e_2$ and the wedges of $e_3$ separately. After this, the divide-and-conquer proceeds by partitioning each edge into connected subsets of approximately equal size.

According to the standard procedure for Voronoi diagrams, the Voronoi cells of L and R are computed recursively, and the *contour* separating V(L) from V(R) is computed. Portions of V(L) lying to the right of the contour are discarded as are portions of V(R) lying to the left of the contour. Note that, because the diagram is restricted to f, the contour may be disconnected. We describe later how this is handled.

In general, each Voronoi vertex has at least degree 3. It is possible that there are Voronoi vertices with degree greater than 3. By applying an infinitesimal transformation if needed, we can assume that there are no such degeneracies in the diagram. That is, there are no points that are equidistant from four wedge origins. This implies that each vertex of the Voronoi diagram is of at most degree three. The topology of the diagram may be altered in the process, but distances are only affected infinitesimally.

Property (i) implies that L and R may contain one or two common end-wedges where they meet. Normally, when constructing Voronoi diagrams by a divide-and-conquer method it is assumed that sets are disjoint. Although this case can be handled by describing a slightly more complex algorithm for tracing the contour, we suggest a simple fix that alters the diagram only infinitesimally. We separate the shared end-wedge points infinitesimally, and treat these points as though they were distinct. One point is moved slightly to the right and is considered a part of R and the other is moved slightly to the left and is consider a part of L.

The following properties of the Voronoi diagrams arising in our case are relevant to its construction. These properties are standard for generalized Voronoi diagrams, and similar arguments appear in [Sh85].

(a) Each component of the contour is a simple curve. [This follows because the contour as part of the diagram is incident on vertices of degree at most three. But exactly two edges of a vertex of degree three can separate the regions V(L) and V(R).]

(b) Voronoi cells are star-shaped with respect to their origin, that is, if x lies on $V_i$, then the straight line from x to $o_i$ does not pass through the interior of any other Voronoi cells. [We have shown that the end-wedges insure that the shortest path to the origin of the containing cell is a straight line. Suppose that the line from x to $o_i$ passes through a point y in the interior of another cell $V_j$. Then there is a shorter path from x to $o_j$ passing through y, a contradiction.]

(c) Each connected component of the contour has endpoints on the boundary of f. [Since the contour partitions f into two possibly disconnected regions, any component with no endpoint on the boundary of f must be a cycle lying entirely within f. However, this violates (b) because the origins lie on f's exterior.]

(d) The Voronoi diagram is the union of bisector segments between weighted points, that is, hyperbolic segments whose foci are origins or straight line segments.

(e) No two edges of the Voronoi diagram are tangent, and no edge of any cell is tangent to a ray emanating from the cell's origin. [To see the first observation, suppose that two Voronoi edges are tangent. Then there are two edges in the same cell that are tangent. Consider the region of the Voronoi cell lying on the opposite side from the wedge origin. This region contains a neighborhood of some point in its interior. However, not all of the straight line segments connecting the points in this neighborhood to the origin can pass through the tangent point, violating (b). The second observation follows from (d) and noting that no point on a hyperbola can be tangent to a ray emanating from one of its foci.]

(f) The Voronoi diagram of a set of m wedges contains O(m) edges. [This follows from Euler's formula and the fact that Voronoi vertices have degree 3.]

Suppose that we are at the stage of building the diagram for a subset of wedges, S, where $|S| = m$. We identify two principle tasks:

(1) determine O(m) points such that each connected component of the contour has one of these points as an endpoint and

(2) trace each of the components of the contour.

We first consider the task of determining the contour endpoints. Note that L and R lie on some subset of the edges of f. By property (b) it follows that each connected component of the contour has at least one of its endpoints on an edge that is in this subset. Hence we may limit our search to this subset of edges. From condition (iii) above, either L and R are on the same edge, or else they lie on distinct edges. First consider the case when L and R lie on the same edge. One of the contour's endpoints is the bisector between the leftmost point of R and the rightmost point of L. (These points

arose from the infinitesimal end-wedge separation mentioned earlier.) By property (b), there can be no other contour end points on this edge.

Next consider when L and R lie on different edges. For concreteness, say that the wedges of R lie entirely on one edge $e_1$, and the wedges of L lie on one or both of the other edges of f, $e_2$ and $e_3$. We describe the procedure for determining the endpoints of the contour that lie on $e_1$, and the cases for $e_2$ and $e_3$ follow analogously. Each contour endpoint is equidistant from the nearest origin in L and the nearest origin in R. Consider the wedges lying on opposite sides of $e_1$, that were constructed in the previous section. R consists of a subset of the wedges lying outside of f and some additional end-wedges. Let $R^*$ denote the subset of R with end-wedges removed. All of the wedges on $e_1$ lying inside of f arose from the extension of a wedge lying outside $e_2$ or $e_3$ or from the vertices of f. Let $L^*$ denote the subset of wedges on $e_1$ lying inside of f that arose from the extension of wedges in L. By the definition of wedge extension given in the previous section, the distance from a point on $e_1$ to an origin of $L^*$ is equal to the distance to the corresponding origin of L (see Fig. 5.2).

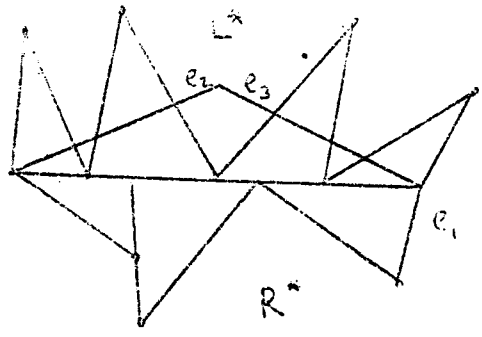


Fig. 5.2 Determining Contour Endpoints

The contour endpoints on $e_1$ are the points equidistant between the wedges $R^*$ and $L^*$. To find these points points we join the two copies of $e_1$ created by the monotonic problem and consider each of the O(m) intervals on $e_1$ where a wedge base from $R^*$ overlaps a base from $L^*$. Within each interval the nearest neighbors in R and L are uniquely determined. For each interval we construct the bisecting hyperbola between the respective origins. The points at which the hyperbola intersects $e_1$ within the overlap interval are are equidistant from their nearest origins in L and R. Therefore, these points are contour endpoints. Each overlap interval may contribute at most 2 points. Hence the number of contour endpoints (and the number of contour components) is O(m). The fact that this set of endpoints includes at least one endpoint from each contour component follows from the remarks made above in a straightforward manner.

The task remaining is to trace each component of the contour. The contour need not be monotone and Voronoi regions may not be convex, as is true for standard Voronoi diagrams, so we use a standard technique from generalized Voronoi diagrams to aid in the tracing process. We draw *spokes*, line segments emanating from the origin of a Voronoi cell to each of the the Voronoi vertices of the cell (see [Ki79, Sh85]). Because the cells are star-shaped with respect to the origins, this partitions each Voronoi cell into Voronoi *subcells* that are bounded by at most 4 sides: an edge of f, two spokes, and a hyperbolic bisector. Determining the intersection of a hyperbolic curve and such a subcell can be carried out in constant time. Each Voronoi cell is the union of its subsells.

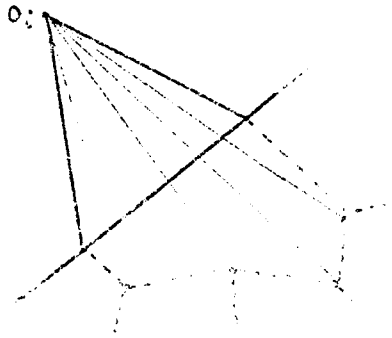


Fig. 5.3 Voronoi Region with Spokes

The tracing proceeds in the standard fashion starting with a contour endpoint in a subcell of a wedge $w_1$ in L and a subcell of a wedge $w_2$ in R. Ignoring degeneracies contour endpoints have degree 1. We follow the bisector between the origins until it reaches the boundary of one of the containing subcells. With the addition of spokes the intersection with the boundary of a subcell can be determined in constant time. By eliminating degeneracies as mentioned above we may assume that the contour reaches the boundary in the interior of a Voronoi edge. Suppose for concreteness that the boundary lies between subcells for $w_2$ and $w_3$, both in R. Property (e) guarantees that the contour will indeed enter $w_3$. We continue along the bisector between $o_1$ and $o_3$. The process is repeated until an edge of f is encountered.

The algorithm for constructing the Voronoi diagram is summarized below:

**Algorithm 5.1** Compute the Voronoi diagram of a set of wedges W lying exterior to and covering the boundary of a face f.

1. Enlarge W to include the end-wedges. Each end-wedge has an origin and a base coinciding at the point where adjacent wedge bases touch.
2. Perform steps 3-6 recursively in divide-and-conquer fashion. Let S=W initially.

3. Divide the connected set of wedges S into two connected sets L and R satisfying properties (i)-(iii) listed above. Recursively compute the Voronoi diagrams of L and R.

4. Construct contour endpoints E as follows for each edge e of f on which either L or R lie:

   4a. If L and R both lie on e, then E contains the point at which the rightmost base of L touches the leftmost base of R.

   4b. If L and R lie on different edges, then E is constructed by considering the overlapping wedge pairs on e formed by L, R or their extensions through f. E contains the points within each overlap region at which the bisecting hyperbola of the corresponding origins intersects e.

5. For each unvisited point in E, trace the contour in standard Voronoi fashion. Spokes are used to simplify the task of searching for the next intersection of the contour with a Voronoi edge.

6. Discard the portions of Vor(L) and Vor(R) lying on opposite sides of the contour and construct spokes to the newly created Voronoi vertices.

7. Discard the spokes.

The correctness and O(m) complexity of the tracing procedure follow from the discussion above. The complexity of finding the contour endpoints is clearly linear, hence the complexity of the divide-and-conquer algorithm is O(n log n) per face because there are O(n) wedges on each face. This implies that the complexity of constructing the Voronoi diagram over the entire polyhedron is $O(n^2 \log n)$. The number of hyperbolic arcs in the diagram is O(n) per face and so the total amount of storage required to store the diagram is $O(n^2)$.

Finally, we consider how to answer a shorest path query. With each wedge w we maintain a pointer to the parent wedge whose extension is w. This imposes a tree structure on the set of wedges, called the *wedge-tree*. Given a query point, we use standard point location techniques to determine the Voronoi cell containing the point [Pr81, Co83] in O(log n) time. The distance to the point can be determined in constant time by computing the weighted distance to the associated wedge origin. The shortest path can be given in O(k) time, where k is the number of vertices and edges traversed by the path, by tracing back the path in the wedge-tree and folding the path over the polyhedron. O(n) space suffices to store the Voronoi diagram for each face, thus altogether $O(n^2)$ storage is used by the algorithm.

## References


[AH74] Aho, A. V., Hopcroft, J. E. and Ullman, J. D. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[Co83] Cole, R. "Searching and Storing Similar Lists," Tech. Rept. 88, Courant Institute, (Oct 1983).

[FA84] Franklin, W. R. and Akman, V. "Shortest Paths Between Source and Goal Points Located On/Around a Convex Polyhedron," 22nd Allerton Conference on Communication, Control and Computing, 1984

[GJ78] Garey, M. R., Johnson, D. S., Preparata, F. P. and Tarjan, R. E. "Triangulating a Simple Polygon," *Inf. Proc. Letters*, 7 (1978), 175-179.

[Ki79] Kirkpatrick, D. G. "Efficient Computation of Continuous Skeletons," in *20th IEEE Foundations of Comp. Sci. Symposium* (1979), pp. 18-27.

[Ki83] Kirkpatrick, D. G. "Optimal Search in Planar Subdivisions," *SIAM J. Comput.*, 12 (1983), 28-35.

[LD81] Lee, D. T. and Drysdale, R. L. "Generalization of Voronoi Diagrams in the Plane," *SIAM J. Comput.*, 10 (1981), 73-87.

[Le78] Lee, D. T. "Proximity and Reachability in the Plane," Ph.D. Thesis, Tech. Rept. ACT-12, Coordinated Science Laboratory, Univ of Illinois, (Nov 1978).

[LP84] Lee, D. T. and Preparata, F. P. "Euclidean Shortest Paths in the Presence of Rectilinear Boundaries," *Networks*, 14 (1984), 393-410.

[LW79] Lozano-Perez, T. and Wesley, M. A. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Commun. ACM*, 22 (1979), 560-570.

[MM85] Mitchell, J., Mount, D. M. and Papdimitriou, C. H. "The Discrete Geodesic Problem," Manuscript, Stanford University, submitted SIAM J. Comput., 1985.

[Mo84] Mount, D. M. "On Finding Shortest Paths on Convex Polyhedra," Tech. Rept. 120, Center for Automation Research, University of Maryland, (May 1985).

[OS84] O'Rourke, J., Suri, S. and Booth, H. "Shortest Paths on Polyhedral Surfaces," Manuscript, Johns Hopkins University, 1984.

[PM83] Preparata, F. P. and Muller, D. E. "Finding the Intersection of n Half-Spaces in Time O(n log n)," *Theoret. Comp. Sci.*, 8 (1979), 45-55.

[Pr81] Preparata, F. P. "A New Approach to Planar Point Location," *SIAM J. Comput.*, 10 (1981), 473-482.

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS<br>N/A |
| 2a. SECURITY CLASSIFICATION AUTHORITY<br>N/A | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE<br>N/A | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>CAR-TR-121<br>CAR-TR-1496 | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>N/A |
| 6a. NAME OF PERFORMING ORGANIZATION<br>University of Maryland<br>6b. OFFICE SYMBOL (If applicable)<br>N/A | 7a. NAME OF MONITORING ORGANIZATION<br>Air Force Office of Scientific Res. |
| 6c. ADDRESS (City, State and ZIP Code)<br>Center for Automation Research<br>College Park, MD 20742 | 7b. ADDRESS (City, State and ZIP Code)<br>Bolling Air Force Base<br>Washington, DC 20332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>-<br>8b. OFFICE SYMBOL (If applicable)<br>- | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>F49620-83-C-0082 |
| 8c. ADDRESS (City, State and ZIP Code)<br>- | 10. SOURCE OF FUNDING NOS.<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO. |
| 11. TITLE (Include Security Classification)<br>VORONOI DIAGRAMS ON THE SURFACE OF A POLGHEDRON | |
| 12. PERSONAL AUTHOR(S)<br>David M. Mount | |
| 13a. TYPE OF REPORT<br>Technical<br>13b. TIME COVERED<br>FROM _____ TO N/A | 14. DATE OF REPORT (Yr., Mo., Day)<br>May 1985<br>15. PAGE COUNT<br>21 |
| 16. SUPPLEMENTARY NOTATION | |
| 17. COSATI CODES<br>FIELD / GROUP / SUB. GR. | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We present an algorithm that computes the Voronoi diagram of a set of points lying on the surface of a possibly nonconvex polyhedron. Distances are measured in the Euclidean metric along the surface of the polyhedron. The algorithm runs in $O(n^2 \log n)$ time and requires $O(n^2)$ space to store the final data structure, where n is the maximum of the number of edges and source points on the polyhedron. This algorithm generalizes or improves the running times of a number of shortest path problems both on polyhedra and in the plane amidst polygonal obstacles. By applying standard algorithms for point location, we can determine the distance from a query point to the nearest source in $O(\log n)$ time and can list the shortest path in $O(k + \log n)$ time, where k is the number of faces traversed by the path. The algorithm achieves its efficiency by a novel method of searching the polyhedron's surface.

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT<br>UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ | 21. ABSTRACT SECURITY CLASSIFICATION<br>UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE NUMBER (Include Area Code)<br>22c. OFFICE SYMBOL |

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

[SH75] Shamos, M. I. and Hoey, D. "Closest-Point Problems," in *16th IEEE Foundations of Comp. Sci. Symposium* (1975), pp. 151-162.

[Sh85] Sharir, M. "Intersection and Closest-pair Problems for a Set of Planar Discs," *SIAM J. Comput.*, 14 (1985), 448-468.

[SS84] Sharir, M. and Schorr, A. "On Shortest Paths in Polyhedral Spaces," in *16th ACM Symposium on Theory of Computing* (1984), pp. 144-153.