CAR-TR-120
CS-TR-1495

F-49620-83-C-0082
May 1985

ON FINDING SHORTEST PATHS ON CONVEX POLYHEDRA

David M. Mount
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

Applications in robotics and autonomous navigation have motivated the study of motion planning and obstacle avoidance algorithms. The special case considered here is that of moving a point (the object) along the surface of a convex polyhedron (the obstacle) with n vertices. Sharir and Schorr have developed an algorithm that, given a source point on the surface of a convex polyhedron, determines the shortest path from the source to any point on the polyhedron in linear time after $O(n^3 \log n)$ preprocessing time. The preprocessed output requires $O(n^2)$ space.

By using known algorithms for fast planar point location, the shortest path query time for Sharir and Schorr's algorithm is shown to be $O(k + \log n)$ where k is the number faces traversed by the path. We give an improved preprocessing algorithm that runs in $O(n^2 \log n)$ time requiring the same query time and space. We also show how to store the output of the preprocessing algorithm in $O(n \log n)$ space while maintaining the same query time.

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

1. Introduction

The study of motion planning in Euclidean 2-space and 3-space has arisen from important applications in the area of robotics. In general, one is interested in whether a set of objects subject to some possible restrictions in motion can be moved from one position in space to another. An important subproblem to consider is that of finding the shortest path from one point to another avoiding polygonal or polyhedral obstacles.

Previous work on this problem includes an $O(n^2 \log n)$ algorithm for finding the Euclidean shortest path in 2-space around polygonal obstacles [LW79, SS84]. In the 2-dimensional case, shortest paths lie along straight line segments passing between the source, the destination and the vertices of the polygons. Thus, the shortest path can be constrained to pass through finitely many line segments. The graph formed by these line segments, called the *visibility graph*, may contain as many as $O(n^2)$ edges and can be constructed in $O(n^2 \log n)$ time. The shortest path is found using standard graph algorithms. In the case that the obstacles are parallel line segments in the plane [LP81, SS84] or the path is constrained to lie within simple polygon [LP81] the problem can be solved in $O(n \log n)$ time.

In Euclidean 3-space when objects consist of polyhedra this technique cannot be employed. Unlike the 2-dimensional case in which shortest paths pass through the finitely many vertices of the polygon, in 3-space shortest paths may pass anywhere along the edges of the polyhedron. Sharir and Schorr present a doubly exponential algorithm for determining the sequence of edges that are traversed along the shortest path using symbolic calculations [SS84].

The visibility graph method can be exploited in 3-space to give a polynomial time approximation [LW79]. This technique consists of covering the edges of the polyhedra with closely placed points. Shortest paths, constrained to pass through these points, can be determined by the visibility graph approach. However, in order to achieve k-digits of precision, exponentially many points may be needed.

The special case in which the obstacle is a single convex polyhedron can be solved in polynomial time. The shortest path between two points on the surface of a convex polyhedron that does not pass through the interior of the polyhedron travels entirely along the polyhedron's surface. Sharir and Schorr present an algorithm that determines the shortest path from a fixed source to any point on the surface polyhedron in $O(n)$ time after $O(n^3 \log n)$ preprocessing time [SS84]. Here n is the number of vertices in the polyhedron. The preprocessed output requires $O(n^2)$ space.

In this paper we continue to investigate the time and space complexity of this problem. We show that the query time can be reduced to $O(k+\log n)$ time, where k is the

number of faces traversed by the shortest path, by using standard techniques from computational geometry. We demonstrate an improved shortest path preprocessing algorithm that runs in $O(n^2 \log n)$ time. Like Sharir and Schorr's algorithm, our algorithm works by partitioning the faces of the polyhedron into polygonal regions according to the discrete structure of shortest paths. Our essential improvement is the observation that these regions arise as the Voronoi diagram of a particular set of points.

We also show how to decrease the space of the preprocessed output to $O(n \log n)$ while maintaining the same query time. For query problems of this type, space is an important resource since it is a factor throughout the lifetime of queries. Space reduction is also of interest, since an immediate impediment to more efficient algorithms (an obvious target is $O(n \log n)$ since this is the complexity of Dijkstra's algorithm on planar graphs). The technique for reducing storage involves the development of a method for sweeping the surface of the polyhedron, which starts at the source and expands monotonically with respect to distance from the source. This polyhedral sweep is somewhat analogous to sweeping the latitude lines on a globe starting from one of the poles.

In Section 2 we summarize pertinent aspects of Sharir and Schorr's analysis of shortest paths on a convex polyhedron. In Section 3 we present our improved algorithm. In Section 4 we develop a data structure representing the shortest path information requiring $O(n \log n)$ space and show how queries are processed on this structure. Throughout, calculations on real numbers are performed approximately using floating point operations.

2. The Structure of Shortest Paths

We begin by recalling the basic structure of shortest paths that affect the preprocessing algorithm. We give many facts without proof and refer the reader to Sharir and Schorr's analysis of the structure of shortest paths [SS84].

Consider a convex polyhedron in 3-space with n vertices. By Euler's formula, the number of edges and number of faces in the polyhedron is $O(n)$. The polyhedron is assumed to be represented by a planar graph with additional geometric information describing the exact location of vertices and edges. This representation can be constructed from the simpler representation as the intersection of half-spaces in $O(n \log n)$ time [PM83]. Let s_0 denote the source point. Let $f_0, f_1, \ldots, f_m$ denote the faces of the polyhedron where f_0 is the face containing s_0. For simplicity of presentation, we assume that s_0 lies on the interior of a face. This assumption may be met by making an infinitesimal adjustment in the location of the source point. Each face is a closed convex polygon. The sum of the angles formed by the edges incident on any vertex of a convex polyhedron is at most 2π. The *trivial* vertices for which the sum equals 2π can be

removed from the polyhedron, without altering the shape of the polyhedron or the shortest paths.

The points on each face are represented using a 2-dimensional coordinate system associated with the face. A point lying on the polyhedron is represented by specifying the face on which the point lies and the position of the point relative to the face's coordinate system. Consider a pair of adjacent faces, f_i and f_j, sharing the common edge e. There is a orthogonal transformation that maps the points of f_j into the plane of f_i by *unfolding* the edge between the faces. This transformation, called the ***planar unfolding*** of f_j relative to f_i, essentially changes coordinate systems from f_j to f_i.

There are three observations leading to the shortest path algorithm [SS84]:

Observation 2.1

(1) Suppose that it is known that the shortest path from point s_0 to point x on the surface of the polyhedron passes through the interiors of the edges $e_1, e_2, \ldots, e_m$. Then the actual shortest path can be determined by unfolding the sequence faces incident on these edges. The shortest path after unfolding is the straight line joining the points (see Fig. 2.1).

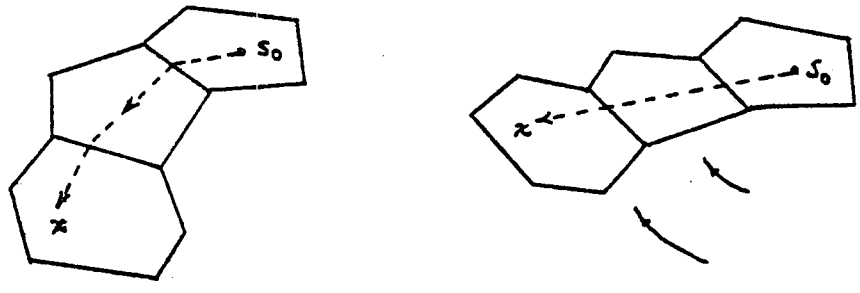

Fig. 2.1 Unfolding to Determine Shortest Path

(2) Two shortest paths, emanating from a common source do not cross.

(3) A shortest path cannot pass through a vertex of the polyhedron.

The third observation follows by recalling that the sum of the angles formed between the edges incident on any nontrivial vertex of a convex polyhedron is strictly less than 2π. This means that the faces incident on a vertex cannot be unfolded onto the plane without splitting one of the faces. There are two perspectives through which this splitting can be viewed. The first perspective arises by looking along the shortest path from the source toward the vertex. Planar unfolding proceeds both in a clockwise and counterclockwise direction around the vertex until the face opposite this path is split in two parts. This splitting leads to the *peels* of Sharir and Schorr's algorithm. By

splitting the polyhedron into its peels, the polyhedron can be unfolded onto the plane so that shortest paths are just straight lines from the source (see Fig. 2.2). In Section 4, we will return to study this structure.

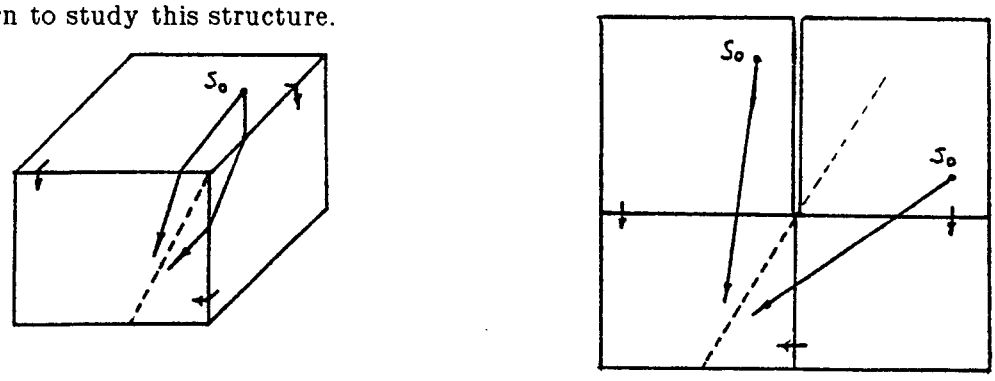

Fig. 2.2 Splitting a Face into Peels

The second perspective arises by considering a point on the face that was split by the above operation. Looking back from the point toward the source, the path from the source to the vertex is split when unfolded around the vertex. In other words, there are two images of the source point relative to the face. The perpendicular bisector between these points passes through the vertex, and partitions the face into two regions. The region in which a point lies determines to which side of the vertex the shortest path lies. These regions are the same as the peels described earlier (See Fig. 2.3).

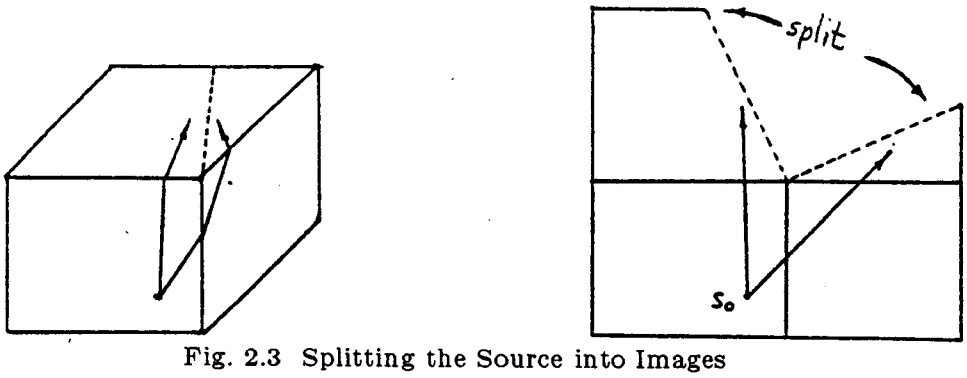

Fig. 2.3 Splitting the Source into Images

We can define an equivalence relation on the points of the polyhedron, equating two points if the shortest paths to the points traverse the same sequence of faces. An equivalence class of this relation forms a region on a face of the polyhedron called a *slice*. (Our usage of the term slice differs from Sharir and Schorr's. Our slice corresponds to the intersection of their peel with a face.) Of course, the edges of the polyhedron define part of the boundary between slices. The other points forming slice boundaries are called the *ridge points*. For example, the bisector of Fig 2.3 consists of ridge points. Ridge points are characterized by the fact that there are at least two shortest paths to each ridge point (See Fig. 2.4).

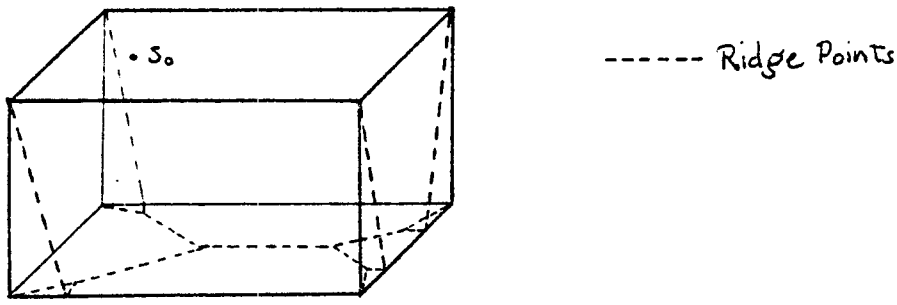

Fig. 2.4 Slices and Ridges

A shortest path traverses a given face at most once, thus the length of the sequence of faces giving rise to a slice is bounded by the number of faces, and hence the number of slices is finite. In fact, quite a bit more can be said about the slices:

Lemma 2.1 (Sharir and Schorr [SS84])

(1) The set of ridge points is the union of finitely many line segments called *ridges*.

(2) The ridges form a continuous tree on the polyhedron whose leaves are the vertices of the polyhedron and whose vertices of degree 2 are the intersection of ridges and polyhedron edges.

(3) Each slice is a convex polygon.

(4) Shortest paths do not pass through ridges.

We also add an observation, which follows from the unfolding rule for geodesics.

Lemma 2.1b Consider a vertex of the tree defined in Lemma 2.1 (2) of degree 2. The pair of ridges incident on this vertex unfold to a straight line.

It follows from Lemma 2.1 (2) that the number of slices on a given face is bounded above by the number of vertices on the polyhedron. Hence, there are $O(n^2)$ slices over the entire polyhedron. The output of the preprocessing phase consists of the set of slices represented by the line segments forming their boundaries. It follows from planarity and Euler's formula that there are of $O(n^2)$ ridges.

Since there are infinitely many points in a slice, the distance from the source cannot be represented explicitly for each point. However, there is a simple finite representation. For all the points in a slice, the planar unfolding of the shortest paths are straight lines converging at a single point, the unfolded image of the source. A slice is associated with this unfolded source image, expressed in the coordinate system of the face on which the

slice lies. Once we know which slice a point lies in, the shortest path can be easily determined by drawing a straight line to the unfolded source and folding this line over the polyhedron. Hence, the shortest path problem is reduced to the problem of point location. Using standard algorithms for point location [Ki83, LT77, Co83, Pr81] and observing that there are $O(n^2)$ slices we have:

Observation 2.2 After determining the slices, queries may by answered in $O(k + \log n)$ time (where k is the number of faces traversed by the shortest path) from a data structure of size $O(n^2)$. Certain queries of constant output length, such as determining the length of the shortest path or determining the initial direction of the shortest path can be answered in $O(\log n)$ time.

Both Sharir and Schorr's preprocessing algorithm as well as ours proceed much like Dijkstra's algorithm for single source shortest paths in graphs. The computation sweeps outwards radially from the source point to increasing distances. A priority queue is maintained holding distances to the upcoming significant events. To picture the algorithm, imagine a fluid dispersing at a fixed rate of speed from the source. A function representing the distance of the shortest path to each point is maintained. As the fluid passes from one face to another, this function is updated. This update operation basically consists of unfolding the shortest path information across the edge to the next face.

3. Shortest Path Function

Sharir and Schorr's algorithm operates essentially by explicitly constructing the ridges. Our approach is to represent the shortest path function (relative to a single face) simply by a set of points, and show that the Voronoi diagram [SH75] of these points define the ridges. Each point is the image of the source after unfolding the faces along a shortest path (see Fig. 3.1).

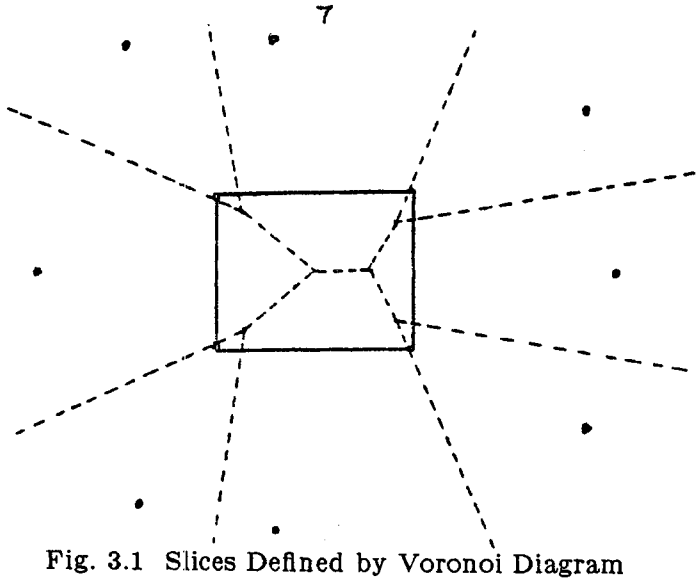

Fig. 3.1 Slices Defined by Voronoi Diagram

We have defined the planar unfolding of a path on the polyhedron. Since the unfolded paths that we deal with are line segments, we say that a point p is the planar unfolding of the shortest path from a point x on the polyhedron if the segment px is the planar unfolding of the shortest path to x. We define a set of points parameterized by face and distance.

Definition - $Src_i(d)$ and Trav(p):
For a face f_i, $Src_i(d)$ consists of the set of points p (given in the coordinate system for f_i) that are the planar unfolding of a shortest path of length at most d from the source to some point on f_i (other than a vertex). For $p \in Src_i(d)$, Trav(p) denotes the sequence of faces traversed by the planar unfolding giving p.

Each slice containing a point at distance at most d from the source, contributes a single point to $Src_i(d)$. If $\hat{d} \leq d$ then $Src_i(\hat{d}) \subseteq Src_i(d)$. For sufficiently large d, $Src_i(d)$ contains one point for each slice on f_i. Since there are at most n slices on a given face, $Src_i(d)$ contains at most n points. Since the position of a point $p \in Src_i(d)$ is determined uniquely by Trav(p), it can be shown that no two points of $Src_i(d)$ have the same position, relative to f_i.

The exclusion of the vertices of face as destinations for points in $Src_i(d)$ does not significantly affect the slices that are generated, since shortest paths do not pass through vertices, and a slice that intersects a face only at a vertex may easily be ignored. The elimination of the vertices simplifies the explanation of the algorithm by removing some special cases that would otherwise have to be considered.

Next we show that $Src_i(d)$ contains enough information to infer the slice boundaries. Let $|xy|$ denote the Euclidean length of the segment xy.

Definition - $Vor_i(p,d)$:

For $p \in Src_i(d)$, let $Vor_i(p,d)$ denote the set of points in f_i that:

(1) lie within distance d of p and

(2) are closer to p than to any other point in $Src_i(d)$.

$Vor_i(p,d)$ is the intersection of the face f_i, the Voronoi polygon for p relative to $Src_i(d)$, and the circle of radius d about p. We consider $Vor_i(p,d)$ to include its boundary. Define $Vor_i(p)$ to be the same as $Vor_i(p,d)$ without the distance bound.

It is not clear, that just because a line segment can be drawn from a point on a face to an unfolded image of the source, that a path of this length exists along the polyhedron. However, the next lemma says that we cannot underestimate the length of the shortest path by doing so. This lemma depends crucially on the convexity of the polyhedron and the fact that shortest paths unfold to straight lines. This observation is fundamental to our improvement of Sharir and Schorr's algorithm.

Lemma 3.1 Let x be a point on a face f_i and let $p \in Src_i(d)$ such that $|px| = \hat{d}$, then the shortest path from the source to x has length no greater than $\hat{d}$.

Proof

Since $p \in Src_i(d)$, there is a point y on f_i such that p is the planar unfolding of the shortest path to y. Let q be the planar unfolding of the shortest path to x. It suffices to show that $|qx| \leq |px|$. If $q = p$ then we are done. Otherwise, consider the segment xy, which lies entirely on f_i by convexity. Let $z_1, z_2, \ldots, z_m$ denote the intersection of xy with the ridges as encountered from x to y (left to right as shown in Fig. 3.2). Let $z_0 = x$. Let q_j be the planar unfolding of the shortest path to the points on the segment (z_j, z_{j+1}) (so $q_0 = q$ and $q_m = p$). Because there are finitely many slices there are finitely many points z_j.

If z_i do not exist then it must be that $p = q$ — Explain why

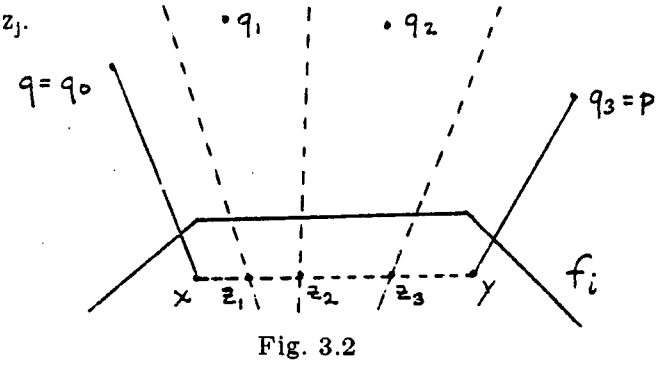

Fig. 3.2

Because the distance function on the surface of the polyhedron is continuous and is equal to the Euclidean distance to the unfolded source image, the distance

$|q_{j-1}z_j| = |q_jz_j|$. Therefore, the bisector between q_{j-1} and q_j passes through z_j. This means that the set of points on xy to the left of z_j are closer to q_{j-1} than to q_j, and in particular $|q_{j-1}x| \leq |q_jx|$. By induction we have $|qx| = |q_0x| \leq |q_mx| = |px|$.

□

Lemma 3.2 Consider a point x on f_i and $p \in Src_i(d)$. Then $x \in Vor_i(p,d)$ if and only if the length of the shortest path from the source to x is at most d and p is the planar unfolding of the shortest path to x.

Proof

Suppose that p is the planar unfolding of the shortest path to x of length $\hat{d} \leq d$. Clearly, $|px| = \hat{d}$. If $x \notin Vor_i(p,d)$ then there must be a point $q \in Src_i(d)$ such that $|qx| < \hat{d}$ implying by Lemma 3.1 that the shortest path to x is less than $\hat{d}$, a contradiction.

Suppose that $x \in Vor_i(p,d)$. (We assume that x lies in the interior of $Vor_i(p,d)$. The general result follows by continuity of the distance function.) Let $\hat{d} = |px|$. Clearly, $\hat{d} \leq d$. By Lemma 3.1, the shortest path to x is of length no greater than $\hat{d}$. If p is not the planar unfolding of the shortest path to x then there is a point $q \in Src_i(d)$ such that $|qx| \leq \hat{d}$ implying that $x \notin Vor_i(p,d)$, a contradiction.

□

Our goal is to compute $Src_i(d)$, where d is as large as the longest shortest path on the polyhedron. Our algorithm constructs a set Src_i^* for each face f_i and we show Src_i^* is equal to $Src_i(d)$. For each point $p \in Src_i^*$ we also construct $Trav^*(p)$, which we show to be equal to Trav(p). For $p \in Src_i^*$ and an edge e on f_i, let $Vor_i^*(p,e)$ denote the Voronoi polygon of p with respect to the points in Src_i^*, restricted to the interior of e with no distance restriction. As mentioned earlier, the computation is controlled by a priority queue containing pending significant events. Each event is associated with a distance, and events are extracted from the queue in increasing distance.

We briefly describe the computation of Src_i^* without giving implementation details and prove the algorithm's correctness. Consider a point $\hat{p} \in Src_j(d)$ for some d. Let e be an edge on f_j and let f_i be the other face incident on e. If there is a point x on the interior of e such that $x \in Vor_j(\hat{p},d)$, then the point p, defined to be the planar unfolding of $\hat{p}$ across e, is in $Src_i(d)$. The smallest d for which $p \in Src_i(d)$ is the closest point to $\hat{p}$ in the intersection of $Vor_j(\hat{p},d)$ and the interior of e. (If the closest point is a vertex of e,

then we take x to be this vertex, but we add the additional constraint that the intersection of $Vor_j(\hat{p},d)$ and the interior of e is nonempty.) This event is called the *extension* of $\hat{p}$ to p.

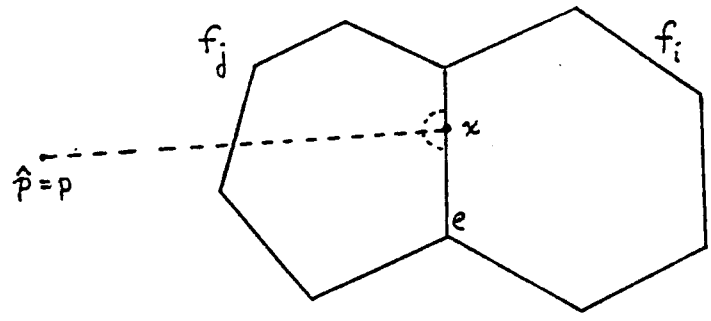

Fig. 3.3 Extension of $\hat{p}$ to p

Each entry in the priority queue corresponds to such an event. An event is described by the tuple, $<d,x,\hat{p},j,i>$ meaning that $\hat{p} \in Src_j^*$, x is the closest point to $\hat{p}$ in $Vor_i^*(\hat{p},e)$ where e is the edge between f_i and f_j and $|\hat{p}x| = d$. (Since $Vor_i^*(\hat{p},e)$ may be open, we take x to be the limit point closest to $\hat{p}$ subject to the constraint given above that $Vor_i^*(\hat{p},e)$ is nonempty.)

Algorithm 3.1

$Src_0^* := \{s_0\}$;

$Trav^*(s_0) := <f_0>$;

for i = 1 to m do $Src_i^* := \{\}$;

for each edge e incident on f_0 do

  enqueue the event associated with the closest point on e to s_0;

while priority queue is nonempty do

  extract event $<d,x,\hat{p},j,i>$ from queue for which d is smallest;

  {let e denote the edge between f_i and f_j}

  p := planar unfolding of $\hat{p}$ across e;

  $Src_i^* := Src_i^* \cup \{p\}$;

  $Trav^*(p) := Trav^*(\hat{p})$ concatenated with $<f_i>$;

  for each edge $\hat{e}$ of f_i other than e do

    if $Vor_i^*(\hat{p},e)$ is nonempty then

      enqueue the event associated with the closest point

        to p in $Vor_i^*(\hat{p},e)$;

    fi

    update $Vor_i^*(\hat{q},e)$ and priority queue entries affected

by $Vor_i^*(\hat{p},e)$;

end

end

To establish the correctness of the algorithm, we prove the following invariant:

Lemma 3.3 Inv(d_2): Let d_2 be the distance to the next event in the priority queue, and let d_1 be the distance of the most recently processed event from the priority queue. For d such that $d_1 \leq d < d_2$, $p \in Src_i(d)$ if and only if $p \in Src_i^*$ and $Trav(p) = Trav^*(p)$.

Proof

Suppose that $p \in Src_i(d)$, where $d_1 \leq d < d_2$. Let x be the nearest point to p on f_i (other than a vertex) for which p is the planar unfolding of the shortest path. Let $\hat{d} = |px|$. Clearly, x lies on the interior of an edge e between f_i and an adjacent face f_j, implying that x lies on both f_i and f_j. Let $\hat{p}$ be the planar unfolding of the shortest path to $\hat{x}$. Clearly, p is the planar unfolding of $\hat{p}$ across e, and $|\hat{p}x| = |px|$.

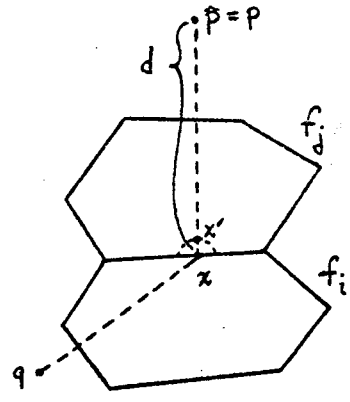

Fig. 3.4

If $\hat{d} < d_1$ then p is already a member of Src_i^* by Inv(d_1). If $\hat{d} = d_1$ then since x lies on the interior of e, there is a point $\hat{x}$ on the interior of f_j lying on the segment $\hat{p}x$ such that $|\hat{p}\hat{x}| < d_1$. By Inv(d_1), $\hat{p} \in Src_j^*$. If $x \notin Vor_j^*(\hat{p},e)$ then there exists a point $q \in Src_j^*$ such that $|qx| < |\hat{p}x| \leq d_1$. By Inv($d_1$), $q \in Src_j(d_1)$ implying that $\hat{p}$ is not the planar unfolding of the shortest path to x, a contradiction. Therefore, $Vor_j^*(\hat{p},e)$ is nonempty (since it contains x) and thus $\hat{p}$ has been extended to p in Src_i^*.

Finally, suppose that $d_1 < \hat{d} < d_2$. If $x \notin Vor_j^*(\hat{p},e)$ then, as above, it follows that $\hat{p}$ is not the planar unfolding of the shortest path to x. If $x \in Vor_j^*(\hat{p},e)$ then there is an event in the priority queue at distance $\hat{d}$, contradicting the assumption that the next event is at distance d_2.

Conversely, suppose that $p \in Src_i^*$. Let $\hat{d}$ be the distance of the event at which p was added. Clearly, $\hat{d} \leq d_1$ since events are processed from the priority queue in ascending distance. If $\hat{d} < d_1$ then $p \in Src_i(d)$ by $Inv(d_1)$. If $\hat{d} = d_1$ then p was added to Src_i^* by the extension of some point $\hat{p} \in Src_j^*$. Hence, p is the planar unfolding of $\hat{p}$ across e. There is a point $x \in Vor_j^*(\hat{p},e)$ such that $|\hat{p}x| = d_1$. The point x is also a point of f_i. If, p is not the planar unfolding of the shortest path to x, then there is a point $q \in Src_i(d_1)$ such that $|qx| < |px| = d_1$ and $x \in Vor_i^*(q,e)$. By $Inv(d_1)$, $q \in Src_i^*$ implying that there is a point $\hat{q} \in Src_j^*$ that is the planar unfolding of q. But $|\hat{q}x| < |\hat{p}x|$ implying that $x \notin Vor_j^*(\hat{p},e)$, a contradiction.

That $Trav(p) = Trav^*(p)$ is an simple consequence of all of this.

□

Finally, we describe the implementation of the algorithm and analyze its complexity. As mentioned earlier, the number of slices on a face is bounded by n and hence there are $O(n^2)$ slices overall. Since the points of Src_i^* are in 1-1 correspondence with the slices, there are $O(n^2)$ extractions from the priority queue, which can be processed in $O(n^2 \log n)$ time. We show that the remaining operations can also be performed within this bound.

For an edge e, lying between f_i and f_j, two separate structures are maintained, one for each face. The structure for e in f_i contains the points of Src_i^* for which $Vor_i^*(p,e)$ is nonempty. By convexity of Voronoi polygons, each intersection is an interval along e, and these intervals are ordered from one end of e to the other by storing them in a balance tree. For each point p, the closest point to p in $Vor_i^*(p,e)$ is also recorded.

Since there are at most n slices on each face it suffices to show that the system of intervals on one edge e can be maintained through n insertions in total $O(n \log n)$ time. Since there are $O(n)$ edges on the polyhedron, this will give an $O(n^2 \log n)$ algorithm. We outline this straightforward procedure.

Suppose that a new point p is added to Src_i^*. The algorithm determines the pair of points, q_1 and q_2, already in Src_i^* whose orthogonal projection onto the infinite extension of e lies just to the left and right of the orthogonal projection of p onto e. Note that the nonempty Voronoi intervals on e are ordered along e exactly as are the orthogonal projections of the points defining these intervals. Hence, if the Voronoi interval for p is nonempty, it lies between the intervals for q_1 and q_2. This operation can be done in $O(\log n)$ time by bisection.

The algorithm works outward to both the left and right of the point lying between these two Voronoi intervals. For each interval encountered (let q denote the source

image for this interval), determine whether some or all of the interval is closer to p than to q, and if so either trim the interval or delete it altogether. If we are to the left of p, the interval is trimmed along its right side, and vice versa. If an interval is deleted, continue the process on the next interval in order. The correctness of this operation is easy to show. The time required is O(1) for each interval, and hence is proportional to the number of points deleted. But overall, the total number of points deleted from each edge is no more than the number of points added, namely n. Note that whenever a new point is added, there may be many intervals deleted but only two intervals, one on the left and one on the right are trimmed. Over the entire algorithm there are O(n) intervals trimmed on each edge.

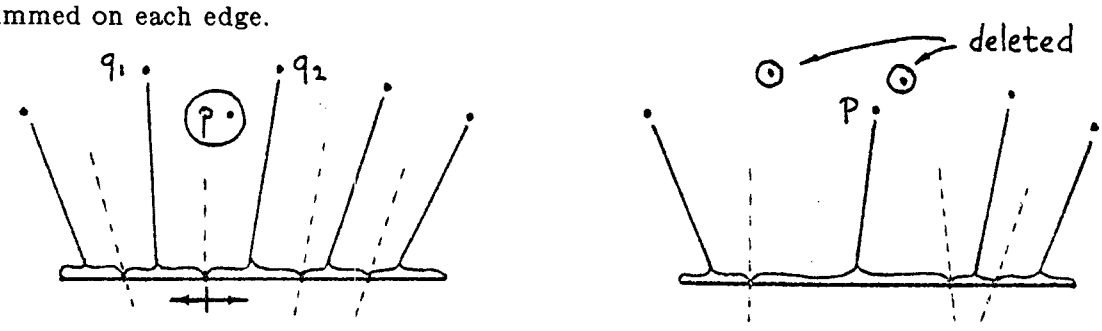

Fig. 3.5 Trimming Voronoi Intervals

If $Vor_l^*(q,e)$ is deleted by this procedure, then we need to delete the corresponding entry in the priority queue. If this interval is trimmed, then the point closest to q may change. If this is the case, we delete the existing entry from the priority queue for $Vor_l^*(q,e)$ and insert a new one. These updates can be done in O(log n) time. The final step of the preprocessing algorithm, after the sets Src_l^* have been constructed, is to compute the Voronoi diagram for Src_l^*. The segments of the Voronoi diagram are the ridges. This requires O(n log n) time per face, and hence $O(n^2 \log n)$ over the entire polyhedron [SH75].

As for space requirements, recall that there are $O(n^2)$ ridges over the polyhedron. $Trav^*(p)$ can be stored in O(1) space for each p by observing that if p was added to Src_l^* by the extension of the point $\hat{p}$ then $Trav^*(p)$ is equal to $Trav^*(\hat{p})$ concatenated with $<f_l>$. Hence, to recover $Trav^*(p)$ it suffices to store f_l and a pointer to $Trav^*(\hat{p})$. Overall we require $O(n^2)$ space to store the preprocessed output.

At this point, the output of our algorithm is the same as the output from Sharir and Schorr's algorithm, thus we have:

Theorem 3.4 Given a convex polyhedron with n vertices and a source point on the polyhedron, the shortest path from the source to a query point on the surface of the polyhedron can be determined in $O(k + \log n)$ time after $O(n^2 \log n)$ preprocessing from a structure requiring $O(n^2)$ storage. Here, k is the number of faces traversed by the shortest path.

4. Storing Shortest Path Information

The object of this section is to show that the $O(n^2)$ slices can be represented efficiently in $O(n \log n)$ space so that shortest path queries can still be processed in $O(\log n)$ time. This data structure can be built in $O(n^2 \log n)$ time assuming that the preprocessing of the previous section has been completed.

We apply two principle ideas in reducing the storage. The first idea is standard in computational geometry, that of storing $O(n)$ different but similar lists, each of size $O(n)$ in less than $O(n^2)$ space [DM80, Co83]. The second idea is particular to the processing of convex polyhedra. A typical technique for linearizing the processing of a polyhedron is to decompose the polyhedron with respect to some fixed direction [DM80, Co83]. The reason that this approach in inappropriate in our case is that shortest paths on polyhedra are not necessarily monotone with respect to any fixed direction. We show how to generalize the plane sweep technique to a type of radial sweep on a convex polyhedron.

The algorithm described here starts with the $O(n^2)$ representation of the shortest path structure produced by the preprocessing algorithm of Section 3 and reduces it to $O(n \log n)$ representation. Recall that the output of the preprocessing is a partition of each face into a set of at most n convex regions called slices bounded by ridges. The slices arise as the Voronoi polygons of a set of unfolded source image points. Processing a query is reduced to the problem of determining the region in which the query point lies. To simplify query processing we begin by further decomposing the polyhedron by triangulating face. In addition, whenever a ridge point of degree 3 or higher (a Voronoi vertex) does not fall on an edge or vertex of the polyhedron, we add three edges joining this point to the vertices of the enclosing triangle. The ridges partition each new triangular face into at most n regions. Adjacent regions are separated by a single straight line (see Fig. 4.1).

Fig. 4.1 Triangulation of Faces

Recall from Lemma 2.1 that the ridges form a tree on the surface of the polyhedron with n leaves. Thus there are at most n-2 ridge points of degree 3 or higher. The number of vertices in this triangulated polyhedron is still O(n). Since the graph of the polyhedron is planar, the number of edges and faces is also linear in n.

During this triangulation we may have have introduced trivial vertices for which the sum of angles about the vertex is 2π. The importance of this restriction is that shortest paths do not pass through nontrivial vertices. Note that shortest paths still do not pass through vertices because any trivial vertices created are on ridge points, and shortest paths do not pass through ridge points.

We now consider the global structure of shortest paths on the polyhedron. Each point x on the polyhedron can be associated with two parameters: Ang(x) is the angle that the shortest path makes with the source (with respect to the coordinate system of the face containing the source), and Dist(x) is the distance of the shortest path to x. Each point on the polyhedron is mapped to the polar plane at the coordinates (Ang(x), Dist(x)) (see Fig. 4.2). Ridge points are mapped to two or more points. This *planar layout* mapping is 1-1 since the location of the point can be recovered from these values by folding the corresponding path back onto the polyhedron. The image of the polyhedron under this map is a closed continuous region of the polar plane containing the origin. Sharir and Schorr observe that this region is a star-shaped polygon. Since shortest paths do not cross ridges or vertices, the boundary of this polygon consists of the ridges and vertices. The planar layout can be physically interpreted as cutting the polyhedron along the ridges and unfolding the resulting object onto the plane.

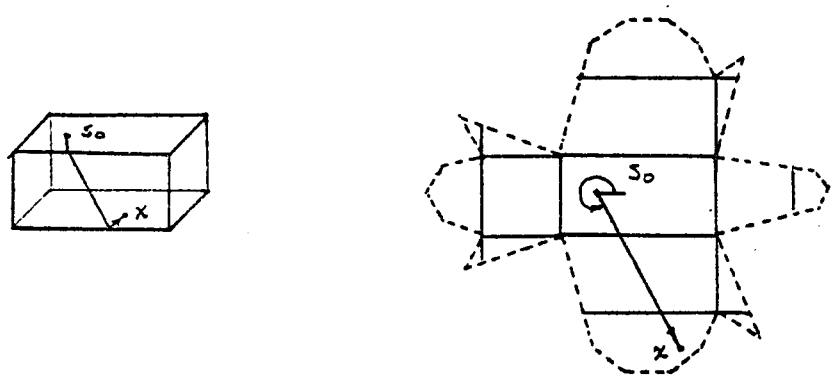

Fig. 4.2 Planar Layout of a Polyhedron

Lemma 4.1

(1) The shortest path from the source to a point x is mapped to a ray of length Dist(x) at angle Ang(x).

(2) Each ray emanating from the origin intersects the image of an edge in at most one point.

Proof

(1) is obvious from the observations made in section 2. (2) follows from the fact that the initial segment of a ray terminating on the boundary of the polygon is the image of a shortest path. Shortest paths cannot cross an edge more than once, and they cannot travel along an edge without passing through a vertex, which we have shown cannot happen.

□

Since the same edges are unfolded to reach the points within a given slice, the planar layout maps each slice orthogonally into the plane by simply unfolding these edges with respect to the source. The output of the preprocessing can be easily augmented to include the orthogonal transformation mapping the point x of a slice to Ang(x) and Dist(x). This map can be represented as a 3 by 3 homogeneous transformation. The planar layout of an edge is split at the point that a ridge point intersects the edge. In case a ridge segment lies along the edge, we may arbitrarily consider the ridge segment to lie slightly to one side or the other of the edge for simplicity. In this way, an edge is split into finitely many segments by the planar layout mapping. Since each face may contain at most n slices, each edge may be split into at most n segments, called its *layout segments*.

As a point x travels from one end of the edge to the other the value of Ang(x) varies continuously and monotonically within a layout segment and may vary discontinuously at a ridge point. It is tempting to think that Ang(x) changes monotonically clockwise or counterclockwise as x travels from one end of an edge to another, but this is generally not the case. The reason is that for a given edge, the shortest paths may cross the edge from either side.

Consider an edge e to be arbitrarily oriented having a head and tail. The left and right sides of e are defined with respect to an observer standing on e facing its head. For each of e's layout segments (given e's orientation), consider the directed line containing the image of this segment under the planar layout map. Lemma 4.1 implies that this line does not pass through the origin. Let $l_1, l_2, \ldots, l_j$ be the segments of e for which the origin lies to the left of this line, and let $r_1, r_2, \ldots r_k$ be the segments for which the origin lies to the right. Equivalently, l_i (resp. r_i) can be defined to be the points on the edge for which shortest paths cross from the left (resp. right). These sequences are ordered from the tail of e to its head. (See Fig. 4.3. Shortest paths are shown as dashed lines.)

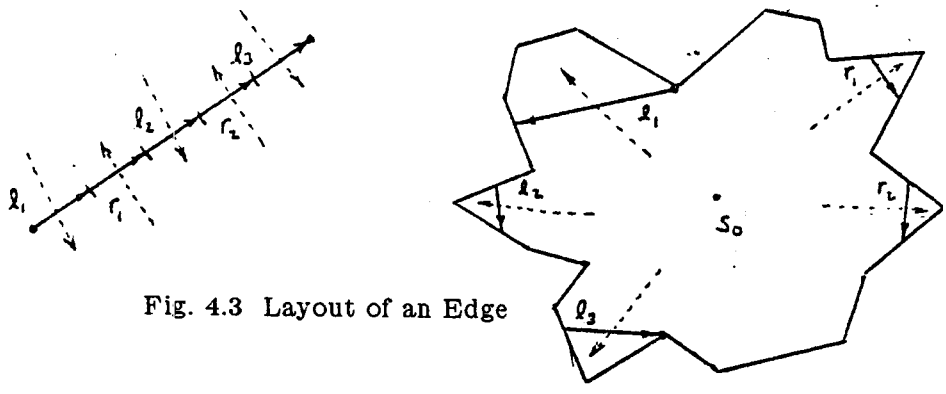

Fig. 4.3 Layout of an Edge

Since no ray intersects an edge at more than one point, we can talk about the cyclic ordering of the points on the edge induced by the planar layout.

Lemma 4.2 Given the directed partition of layout segments defined above, as θ advances cyclically counterclockwise:

(1) the segments of the edge e are intersected by the ray at angle θ in the order $l_1, l_2, \ldots, l_j, r_k, r_{k-1}, \ldots, r_1$.

(2) the points of the segment l_i (resp. r_i) are intersected by the ray at angle θ in order from tail to head (resp. head to tail).

Proof

The lemma follows by simple topological arguments after noting that the planar layout maps shortest paths continuously to the plane and hence preserves the properties that shortest paths do not cross each other and they do not cross an edge more than

once. The proof is most easily visualized by the topological deformation of the polyhedron onto the plane shown below.

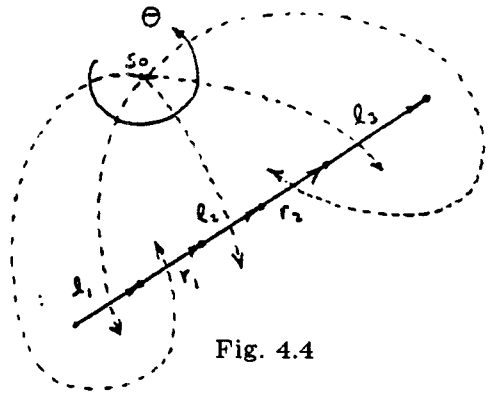

Fig. 4.4

□

This segment partition can be built in $O(n^2)$ time by considering the intersection of edges and ridges, which can be made available from the preprocessing and triangulation phase of the algorithm.

The importance of these observations will become apparent soon. Our approach is to develop a method of sweeping the surface of the polyhedron while maintaining the shortest path information incrementally. The key quality of the sweep is that it proceeds in monotonically increasing distances from the source. The sweep is represented as a total order on the edges of the polyhedron. To establish monotonicity, we order the edges by the relation, $e_1 < e_2$ if a shortest path traverses e_1 before traversing e_2. In general, however, this relation cannot be embedded in a total order.

There are two difficulties in determining a total order. First, if given two edges e_1 and e_2, both edges are traversed from both sides, then we may have $e_1 < e_2$ and $e_2 < e_1$ (see Fig. 4.5a). We overcome this difficulty by replacing each edge by two oppositely directed edges. We make the convention from this point on that a path from the source traverses a *directed edge* only if it crosses the edge from the left side of the edge to the right.

The second difficulty arises when edges overlap spirally with respect to shortest paths so that $e_1 < e_2 < \cdots < e_k < e_1$ (see Fig. 4.5b).

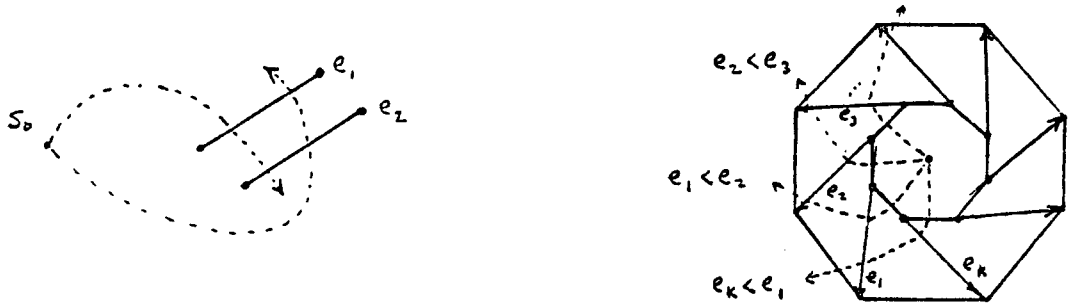

Fig. 4.5

The difficulty arises from the spherical nature of the polyhedron. The spiral can be broken by cutting the polyhedron along a path that simultaneously splits all spirals. Edges that cross this path are split into two edges. Rather than explicitly defining the path, we define its intersection with the edges. For each edge, consider the point x such that Ang(x) is minimal. If this point is not a vertex, then place a new vertex at this point, and retriangulate the face. This means that if x and y are two points on a directed edge such that x is closer to the head of the edge than y and both x and y are traversed by a shortest path from the left side then Ang(x) > Ang(y). We may introduce vertices through which shortest paths travel by this operation and we will treat these as special cases.

The point at which to add this new vertex can be found as follows. Map each layout segment of the edge onto the plane by the planar layout transformation. If any segment crosses the positive x axis, then place the new vertex at the point of intersection. Otherwise, select the segment endpoint mapped to the polar coordinates (θ, d) for which θ is minimum. Invert the planar layout map to determine the location of the new vertex. This can be done in time proportional to the number of layout segments, which is $O(n^2)$.

Lemma 4.3 After splitting the edges as described above, the relation $e_1 < e_2$ can be embedded in a total order.

Proof

We say that a curve on the polar plane is monotone if, as the curve is traced by a point $x = (\theta, r)$ moving from one end of the curve to the other, the value of θ changes monotonically clockwise or counterclockwise. Given two monotone curves, c_1 and c_2, we say that $c_1 < c_2$ if there is a ray at some angle θ intersecting the curves at the coordinates (θ, r_1) and (θ, r_2) respectively, and $r_1 < r_2$. We say that two monotone curves do not cross each other if $c_1 < c_2$ and $c_2 < c_1$ are not both true. Note that noncrossing curves may intersect.

A sufficient condition for embedding the $<$ relation on noncrossing, monotone curves in a total order is that no curve cross the ray $\theta = 0$. By Lemma 4.2 the planar layout maps each *directed* edge to a possibly discontinuous monotone polar curve. It suffices to extend the planar layout of each directed edge to a continuous curve on the polar plane such that

(1) each curve is monotone,

(2) curves do not cross, and

(3) no curve crosses the ray $\theta = 0$.

Consider each segment of an edge e to be directed in the same way that the edge is directed. Since we considering directed edges, shortest paths traverse the edge from only the left side. Let the layout segments be $l_1, l_2, \ldots, l_k$, listed from the tail of e to its head. The endpoint of each segment is a ridge point and so is mapped by the planar layout to the boundary of the star shaped polygon. We connect the head of l_i to the tail of l_{i+1} for $1 \leq i < k$ along the polygon in a counterclockwise direction. Lemma 4.2 and the fact that the polygon is star-shaped guarantees that the resulting curve is monotonic and couterclockwise directed.

To see that the resulting curve does not cross the ray $\theta = 0$ first note that no layout segment crosses this ray for otherwise there is a point x on the segment for which $\mathrm{Ang}(x) = 0$ implying that a vertex has been added here. The connection between two layout segments l_i and l_{i+1} cannot cross this ray, for otherwise we would have a point x on l_{i+1} that is closer to the head of the edge than a point y on l_i and $\mathrm{Ang}(x) < \mathrm{Ang}(y)$, which cannot occur.

Finally, to see that the resulting curves do not cross suppose to the contrary that a shortest path P_1 traverses the (directed) edge e_1 before e_2 and another shortest path P_2 traverses e_2 before e_1. It is important to note that the traversals occur from the same side of each edge (See Fig. 4.6).

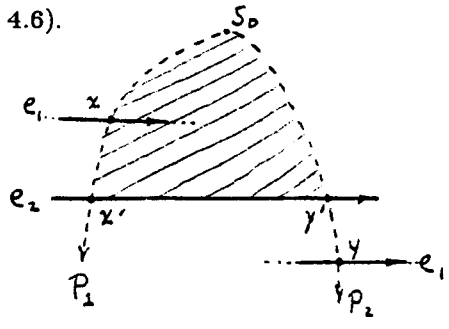

Fig. 4.6

Consider the closed region on the polyhedron bounded by e_2 and the shortest paths. Consider the point x at which P_1 traverses e_1. At this point e_1 crosses the boundary of

this region. The point y at which P_2 traverses e_1 is entirely outside of the region. Let $\hat{x}$ and $\hat{y}$ be the points at which P_1 and P_2 traverse e_2 respectively. There are three possibilities:

(1) If $Ang(x) = Ang(y)$ then one of the paths P_1 or P_2 is a subpath of the other. The longer of the two paths contains the points x, y, $\hat{x}$ and $\hat{y}$ ordered in distance from the source so that $x < \hat{x}$ and $\hat{y} < y$. This in turn means that the segments xy and $\hat{x}\hat{y}$ overlap implying that the edges themselves overlap, which cannot occur.

(2) If $Ang(x) < Ang(y)$ then x lies closer to the tail of e_1 than y and hence e_1 must eventually leave the region either by crossing e_2 or by recrossing one of the shortest paths which is impossible.

(3) Suppose that $Ang(x) > Ang(y)$. By definition of the region, $\hat{x}$ is closer to the tail of e_2 than $\hat{y}$, but $Ang(\hat{x}) = Ang(x) > Ang(y) = Ang(\hat{y})$ which cannot be.

□

The $<$ relation on edges can be determined in $O(n^2 \log n)$ time by a cyclic counterpart to standard line sweep algorithms (see [Pr81], for example). The resulting total order gives us the desired sweep procedure for the polyhedron.

Next we consider the shortest path information that is maintained for each edge. Recall that each face of the polyhedron is a triangle that is partitioned by O(n) ridges that do not intersect in the interior of the face. Also recall that the ridges are the Voronoi diagram of a set of source image points.

For a given face, consider these source image points. Note that except for the face containing the source, for which shortest path queries can be trivially answered, the shortest paths must traverse an edge of the face. It suffices to restrict attention to the shortest paths traversing each edge. For each directed edge e, we define a structure containing the source images, Src(e), of the shortest paths that pass through e from the left. From Section 2 we know that the planar unfolding of the shortest path to a point x on e is the straight line segment joining x to the nearest point in Src(e). Therefore, in the Voronoi diagram of Src(e) each Voronoi polygon intersects e. By convexity of Voronoi polygons, each intersection is a single interval along e. The points of Src(e) are ordered according to the intersection of their Voronoi polygons with e. Note that the Voronoi diagram defined by Src(e) is not necessarily the same as the set of ridges, since the former only includes paths that traverse e from the left side. We give details later on the implementation of this structure, but it suffices for now to assume that we can perform bisection on the set Src(e).

Given such a structure we answer a query as follows. A query point is represented by the face (of the original polyhedron) n which it lies and the coordinates of the point relative to the face. By standard point location techniques, we can determine which of the triangular subfaces contain this point in O(log n) time [Ki83, LT77, Co83, Pr81]. Let x be a query point lying on a triangular face f (that does not contain the source) whose clockwise directed edges are e_1, e_2 and e_3. The shortest path to x must traverse one of these three edges. To determine the shortest path to x is suffices to determine the closest point to x in $Src(e_1)$, $Src(e_2)$ and $Src(e_3)$ separately and then select the closest of the three. To determine the closest point to x in Src(e) we need the following lemma.

Lemma 4.4 Consider a face f, and a clockwise directed edge e_1 on that face and the Voronoi diagram of $Src(e_1)$. The Voronoi diagram restricted to the interior of f contains no Voronoi vertices.

Proof

Let e_2 and e_3 be the other clockwise directed edges, and let $S = Src(e_1) \cup Src(e_2) \cup Src(e_3)$. By construction we know that the Voronoi diagram of S restricted to the interior of f does not contain any Voronoi vertices. This implies that the Voronoi polygons of S intersect at least two edges of f. Thus, the Voronoi polygons of $Src(e_1)$ intersect at least two edges of f since they are supersets of their counterparts in S. Suppose that the Voronoi diagram of $Src(e_1)$ contains a vertex on the interior of f. There are at least three points, p_1, p_2 and p_3, of $Src(e_1)$ equidistant from this vertex. Assume that these points are ordered from left to right by the intersection of their Voronoi polygons with e_1 (See Fig. 4.7).

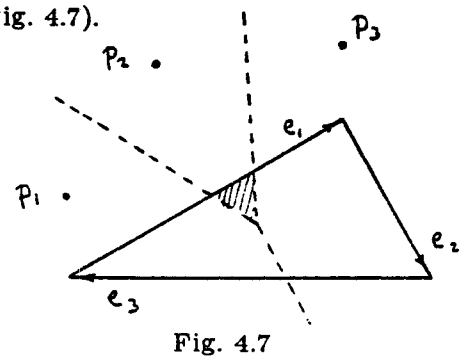

Fig. 4.7

The bisectors between p_1 and p_2 and between p_2 and p_3 intersect the edge e_1 because the Voronoi polygons of the three points intersect e_1. But the Voronoi polygon for p_2 restricted to f is contained in the triangle bounded by these bisectors and e_1. By convexity this triangle does not intersect any edge other than e_1, a contradiction.

□

The closest point to x in Src(e) can be found by bisection as follows. Select the midpoint p_l of Src(e) and the point p_{l+1} to its right. By the ordering of points in Src(e), the bisector between these two points is part of the Voronoi diagram. Because there are no Voronoi vertices on the face, this bisector separates the face into two regions, those points closer to $p_1, \ldots, p_l$ and those closer to the remaining points in Src(e). After O(log n) probes the closest point to x is found. Once the region containing the point is found it is an easy matter to determine the shortest path.

We turn our attention to the data structure that we will use to store the points of Src(e). The points of Src(e) cannot be listed explicitly, since doing so would require $O(n^2)$ storage. Instead we store the points in a set of trees, one tree associated with each directed edge. We take advantage of similarities in local path structure by sharing subtrees.

Let $p \in$ Src(e) where p arises as the planar unfolding of the source s_0 through the directed edges $e_1, e_2, \ldots, e_k$. Let P_i denote the planar unfolding transformation of the edge e_i (mapping a point on the face to the left of e_i to the coordinate system for the face on the right of e_i). Clearly, $p = P_k P_{k-1} \cdots P_1 s_0$. We can recover p by knowing the matrix product $P_k \cdots P_1$.

Since the points of Src(e) are ordered, we may store these points as the leaves of a balanced search tree. For concreteness we will use a 2-3 tree [AH74]. This search tree is not to be confused with the tree formed by the ridges on the surface of the polyhedron. Each vertex of this search tree is labeled with a transformation. Suppose that $p \in$ Src(e) is stored in a leaf of this tree, and the transformations along the path from the root to this leaf are $Q_j, Q_{j-1}, \ldots, Q_1$. We maintain the condition that the product $Q_j Q_{j-1} \cdots Q_1 = P_k P_{k-1} \cdots P_1$. Since the tree is balanced, the length of the Q-products is O(log n) even though the number of unfolded edges is O(n).

The search tree associated with the edges are built as follows. The edges are processed according to the total order introduced earlier. First, consider the three edges that are oriented counterclockwise around the face containing the source. Every shortest path traverses one of these edges before any other edge, hence we may assume that these edges are first in the order. Clearly, Src(e) for each of these edges contains only one point, hence the associated tree consists of a single leaf labeled with the transformation unfolding the edge.

In processing the remaining edges, we construct a new tree by modifying the trees of previously processed edges. (Later we show how to recover the information from the modified trees.) Let e be a directed edge to be processed. Let f be the face to the left of

e, implying e is oriented counterclockwise about f. Let e_1 and e_2 be the other edges of f, listed in clockwise order and directed clockwise. Let $\overline{e}_1$ and $\overline{e}_2$ denote the directed complements of these edges. Where i is 1 or 2, let j be the value 2 or 1 respectively. Clearly any shortest path traversing e must first traverse either e_1 or e_2, meaning that one or both of these edges has already been processed.

For i = 1, 2, if e_i has already been processed, let T_i be the search tree for e_i. We begin by identifying those points in $Src(e_i)$ whose slice does not extend to the surrounding faces. We emphasize that the slice referred to here is the shorest path slice and not the Voronoi polygon of $Src(e_i)$. The slice is a Voronoi polygon of the union of Src for all three edges. As mentioned in Lemma 4.4, each slice intersects at least one edge of f other than e_i. If the intersection of a slice with e consists of a ridge point or ridge segment then no shortest path from the corresponding source point traverses e (see Fig. 4.8).

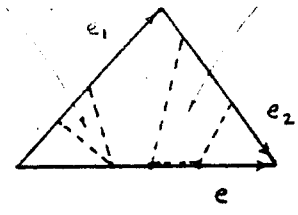

Fig. 4.8 Terminating a Path

Observe that such a slice has a Voronoi vertex on the edge e. The set of such source points can be determined in time proportional to the number of Voronoi vertices on e. Over the entire polyhedron, there are O(n) Voronoi vertices, hence the set of these source points can be determined in O(n) time. We delete each such point from the tree T_i. The deletion of a single point can be performed in O(log n) time. Since each deletion occurs at a Voronoi vertex, the number of deletions is bounded by the number of faces, and hence the number of edges incident on the vertex. From Lemma 2.1 these are edges incident on vertices of degree 3 or more in a tree containing n leaves. There are O(n) such edges, so this operation requires O(n log n) time over the entire polyhedron.

Of the remaining points of $Src(e_i)$, some subset represents shortest paths traversing e (the others traverse $\overline{e}_j$). This set consists of those source images whose slices intersect e. Since these slices do not overlap, and since the points of $Src(e_i)$ are ordered with respect to the intersection of these convex polygons with e_i, the order of intersection with e is identical. In other words, once the leaves of T_i that contain these source points have been identified, no reordering within the set is necessary. The slices of the points of $Src(e_i)$ can be partitioned into three sets, those intersecting only e, those intersecting

only $\overline{e}_j$ and those intersecting both e and $\overline{e}_j$. By convexity, at most one slice intersects both edges. Let p be the source point corresponding to this slice. Because of the ordering of source points, those slices intersecting e lie to one side of the slice for p and those slices intersecting $\overline{e}_j$ lie to the other side. Therefore, the tree of source points from $Src(e_i)$ corresponding to shortest paths traversing e can be found by splitting the tree T_i about the point p so that p is the leftmost leaf of one tree and the rightmost point in the other. Let the tree containing the points for e be called $T_{e,i}$. The remaining tree can be saved until $\overline{e}_j$ is processed. This construction is shown schematically in Fig. 4.9.

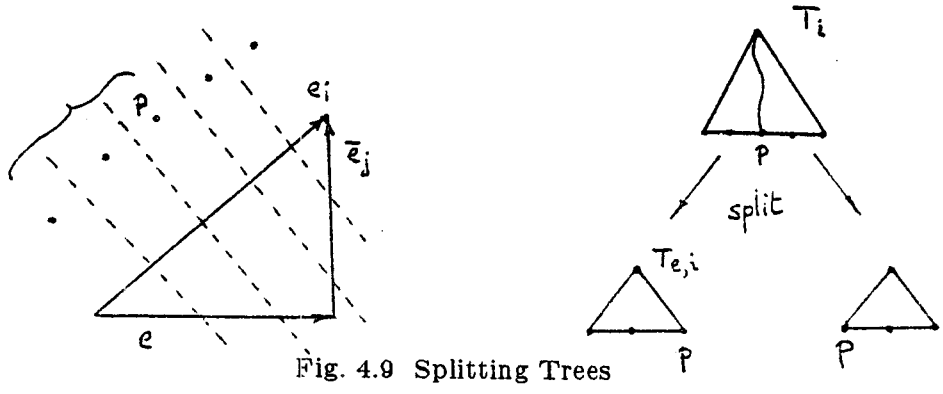

Fig. 4.9 Splitting Trees

The point p can be determined by bisection in O(log n) time. This tree manipulation can be performed in O(log n) time on a 2-3 tree.

If only one of $T_{e,1}$ and $T_{e,2}$ is nonempty then let T_e be this tree. Otherwise, T_e is the concatenation of these trees where $T_{e,1}$ is on the left and $T_{e,2}$ is on the right. As before, there may be at most one point p whose slice intersects both e_1 and e_2. This point that appears in both trees is stored only once in the concatenation. This operation is illustrated in Fig. 4.10.

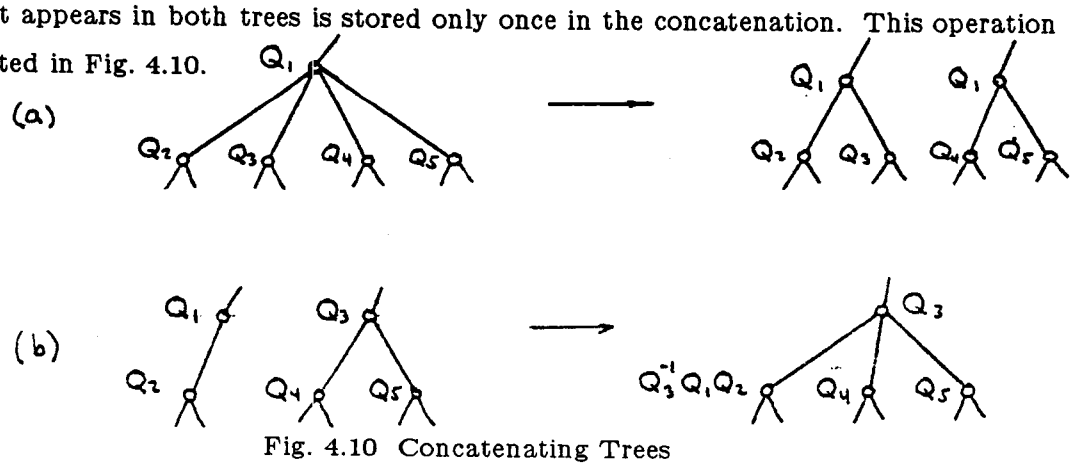

Fig. 4.10 Concatenating Trees

Again, the fact that slices do not overlap guarantees that the points are properly ordered in the resulting tree. This operation can be performed in O(log n) time on 2-3 trees. The tree T_e is completed by premultiplying the root of the tree by the transformation unfolding the edge e.

The correctness of this construction is immediate from the observations we have made. The complexity for deletions was shown to be O(n log n) overall. The complexity of splitting and concatenation is clearly O(log n) for each of O(n) edges, hence O(n log n) overall.

We must still describe how to maintain the multipliers on the tree's vertices during these operations. The key invariant is that the product from the root to each leaf remains the same. The fundamental operations performed on the 2-3 tree are:

(1) Split a 4-node into two 2-nodes by duplicating the transformation for each new subtree (see Fig. 4.11a).

(2) Merge a 1-node into a 2-node preserving the multiplier for the 2-node but premultiplying the inverse of this transformation on the 1-node (see Fig. 4.11b).

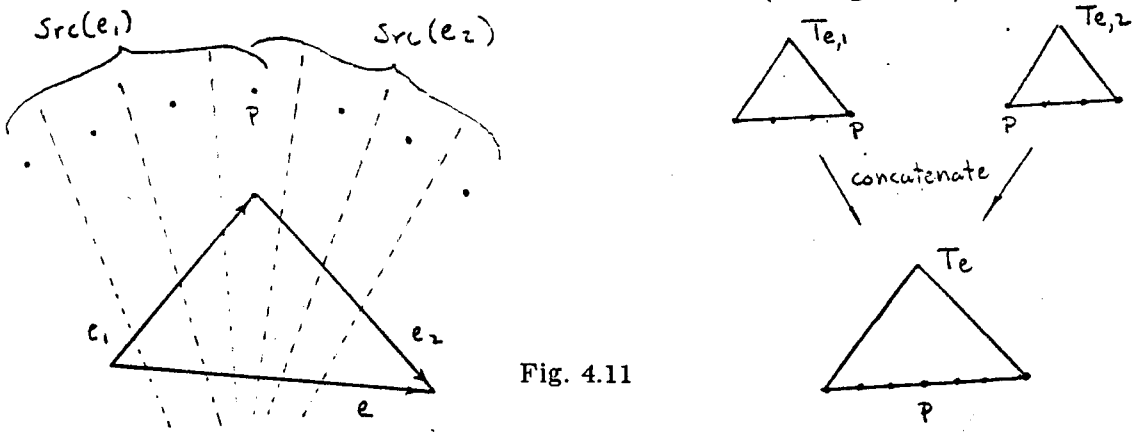

Fig. 4.11

Thus the product from the root to the newly merged subtree is $Q_3Q_3^{-1}Q_1Q_2 = Q_1Q_2$ as it was originally. Note that operations (1) and (2) can be performed in O(1) time and generally may be applied O(log n) times during each 2-3 operation.

(3) Concatenate two subtrees of different heights by performing a premultiplication by the inverse similar to case (2). In this case the product from the root to the point of insertion is inverted (see Fig. 4.12).

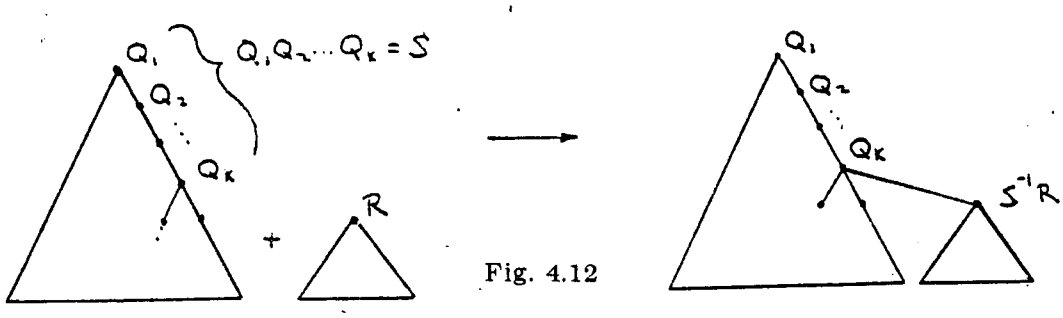

Fig. 4.12

This operation requires O(log n) time, but is performed only once when concatenating trees.

The procedure described here constructs the tree for an edge by modifying (and hence destroying) previously built trees. In order to save the information from the previously processed edges, we employ a simple trick. Rather than modifying a node of the tree, we duplicate the node and perform the operation on the duplicate. Since O(n log n) time suffices to build the trees, O(n log n) nodes may be created in the process. Each tree by itself contains O(n) edges. Each directed edge of the polyhedron is linked to the root of the corresponding search tree.

The bisection outlined earlier used to process queries is implemented by the standard descent on search trees. Since we wish to quickly determine the equation of the bisector between a pair of points, we include an additional piece of information. For each pair of sibling subtrees in the structure, we determine the equation of the bisector between the rightmost point of the left subtree and the leftmost point in the right subtree. This bisector is stored at the common ancestor node. This information can be determined in O(n log n) time by a traversal of the tree structure.

At this point we may dispense with the ridges and store only the triangulated polyhedron and the balanced search tree structure.

Theorem 4.5 Given a convex polyhedron with n vertices and a source point on the polyhedron, the shortest path from the source to a query point on the surface of the polyhedron can be determined in O(k+log n) time after $O(n^2 \log n)$ preprocessing from a structure requiring O(n log n) storage. Here, k is the number of faces traversed by the shortest path.

5. Further Remarks

The results of this paper suggest a number of other problems. Although this paper treated the case of the source and query point lying on the surface of the polyhedron, the results can easily be generalized to the case where the source lies external to the polyhedron and the query point lies on the polyhedron. We observe that in this case, the faces of the polyhedron can be partitioned into two classes: those faces for which the source is visible and those faces for which the source is not visible. For the former set, the shortest path is a straight line, and for the latter, the shortest path is determined by reducing the problem to the convex hull of the source and the polyhedron. This convex hull can be computed in linear time. However, the case in which the query point is exterior to the polyhedron cannot be solved by these methods.

The most interesting generalization would be to find an efficient algorithm for the shortest path problem amidst general nonconvex polyhedral obstacles. Sharir and Schorr [SS84] give a doubly exponential algorithm for this case, and suggest that the problem may be NP-hard. But they consider the problem of finding an exact solution. Can numerical techniques be applied yielding an approximate solution running in time that is polynomial in n and the number of digits of precision? The shortest path problem addressed in this paper is essentially a two-dimensional problem, whereas the general case clearly is a three-dimensional problem.

O'Rourke, Suri and Booth have given an algorithm for finding the shortest path between a pair of points on a nonconvex polyhedron subject to the restriction that the path travel along the surface of the polyhedron in $O(n^5)$ time [OS84]. It would be of interest to find an algorithm solving the single source problem in this case, extending the techniques of this paper. Another problem would be the solution of the all pairs problem on the convex polyhedron, that is, can the polyhedron be preprocessed so that given an arbitrary pair of points on or above the surface of the polyhedron, can the shortest path be determined in time $O(\log^k n)$ for some k.

It is of some interest to determine whether the O(n log n) space can be improved to O(n). Cole has shown how to search in similar lists using only O(n) space [Co83] however it is not clear how to adapt this data structure to handle the additional complication of transformations.

References

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

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

[DM80] Dobkin, D. P., Munro, J. I. "Efficient Uses of the Past," in *Proc. 21st IEEE Symp. on Foundations of Computer Science* (1980), pp. 200-206".

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

[LP81] Lee, D. T., Preparata, F. P. "Euclidean Shortest Paths in the Presence of Rectilinear Boundaries," *Proc. 7th Conf. on Graphth. Concepts in Comp. Sci.*, Carl Hanser (1981, 1982), 303-316.

[LT77] Lipton, R. J., Tarjan, R. E. "Applications of a Planar Separator Theorem,"

in *Proc. 18th IEEE Symp. on Foundations of Computer Science* (1977), pp. 162-170.

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

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

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

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

[SH75] Shamos, M. I., Hoey, D. "Closest-Point Problems," in *Proc. 16th IEEE Symp. on Foundations of Computer Science* (1975), pp. 151-162.

[SS84] Sharir, M., Schorr, A. "On Shortest Paths in Polyhedral Spaces," in *Proc. 16-th ACM Symp. on Theory of Computing* (1984), pp. 144-153. also Tel Aviv Univ. Computer Science Tech. Rept. 84-001, March 1984

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-120 CS-TR-1495	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Res.
6c. ADDRESS (City, State and ZIP Code) Center for Automation Research College Park, MD 20742	7b. ADDRESS (City, State and ZIP Code) Bolling Air Force Base Washington, DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F-49620-83-C-0082
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) ON FINDING SHORTEST PATHS ON CONVEX POLYHEDRA	
12. PERSONAL AUTHOR(S) David M. Mount	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO N/A	14. DATE OF REPORT (Yr., Mo., Day) May 1985 15. PAGE COUNT 30
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

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

Applications in robotics and autonomous navigation have motivated the study of motion planning and obstacle avoidance algorithms. The special case considered here is that of moving a point (the object) along the surface of a convex polyhedron (the obstacle) with n vertices. Sharir and Schorr have developed an algorithm that, given a source point on the surface of a convex polyhedron, determines the shortest path from the source to any point on the polyhedron in linear time after $O(n^3 \log n)$ preprocessing time. The preprocessed output requires $O(n^2)$ space.

By using known algorithms for fast planar point location, the shortest path query time for Sharir and Schorr's algorithm is shown to be $O(k + \log n)$ where k is the number faces traversed by the path. We give an improved preprocessing algorithm that runs in $O(n^2 \log n)$ time requiring the same query time and space. We also show how to store the output of the preprocessing algorithm in $O(n \log n)$ space while maintaining the same query time.

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

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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE