A Workbench to Compute Unobstructed
Shortest Paths in Three-Space

Wm. Randolph Franklin and Varol Akman
Dept. of Electrical, Computer, and Systems Eng.
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
(518) 266-6077

Abstract

Recently, the following problem has gained considerable importance in computational geometry:

FINDPATH: Given a set of obstacles and two points (source and goal), calculate the shortest path between these points under the Euclidean metric, constrained to avoid intersections with the given obstacles.

We present three implementations to solve specific instances of FINDPATH in three-space where the obstacles become polyhedra. In the first two cases there exists a single convex polyhedron and the source and the goal points are on its boundary or exterior. These solutions make use of planar developments of polyhedra and polyhedral visibility. The last case is based on a locus method. It partitions the boundary of a convex polyhedron given only the source on it so that for a later goal on the boundary, the shortest path is found efficiently. It makes use of standard point location algorithms for a straight-edge planar subdivision once the partitioning is done.

0. Introduction and Recent Algorithms in Three-Space

Let \( P = \{ P_1, \cdots , P_n \} \) be a prescribed set of disjoint polyhedra and \( s.g \in R^3 \) be distinct points which are not internal to any \( P_i \). The class of rectifiable curves which have endpoints \( s \) and \( g \) and which do not intersect any int\( P_i \) will be denoted by \( C(s,g;P) \). For \( C \) in this class, \( l(C) \) will denote the
length of $C$ under the Euclidean ($L_2$) metric. An interesting problem in computational geometry asks for the shortest one among these curves:

FINDPATH
INSTANCE: Polyhedra $P = \{P_1, \cdots, P_n\}$ such that $P_i \cap P_j = \emptyset$, $i \neq j$, and $s, g \in \mathbb{R}^3$ such that $s \neq g$ and $s, g$ do not belong to $\text{int}P_i$, $1 \leq i \leq n$.
QUESTION: Which $C \in C(s, g; P)$ has the shortest length?

The following theorem must be intuitively clear:

THEOREM 0.1 There exists a $C^* \in C(s, g; P)$ such that for all $C \in C(s, g; P)$, $l(C^*) \leq l(C)$. Moreover, every such $C^*$ is a polygonal path with its possible bend points belonging to some edges of some members of $P$.

Proof. Omitted. □

It is noted that the polygonal path $C^*$ found in theorem 0.1 is not necessarily unique. (In fact, there may be exponentially many shortest paths in terms of $n$.) We shall call any such path an $s$-to-$g$ shortest path. Thus, FINDPATH asks for the characterization (i.e., the determination of the bend points) of an $s$-to-$g$ shortest path.

There is a wealth of material in the general area of motion planning which comprises other familiar robotics problems such as FINDSPACE, MAKESPACE, etc. in addition to FINDPATH. For brevity, we shall refer the reader to a recent work by Akman[1] which deals with FINDPATH in greater detail and lists numerous references. Franklin and Akman also inspect the same problem from various perspectives in [8, 9, 10, 11].

Sharir and Schorr's work[32] is another detailed study on shortest paths. They mainly consider the case of BOUNDARY FINDPATH (locus) (cf. section 3) and present an algorithm which works in time $O(n)$ per query where $n$ is the measure of complexity of the polyhedron, say the number of vertices. Their algorithm is based on the idea of "ridge" points on the object. A ridge point on the polyhedron has the property that there exists at least two shortest paths to it from the source. It turns out that the set of ridge points is a union of line segments and that the union of the vertices and the ridge points is a closed connected set. Defining the union of the latter with the shortest paths from the source to every vertex, one partitions the polyhedron boundary into disjoint connected regions (called "peels") whose interiors are free of vertices or ridge points. The peels can be constructed in $O(n^3 \log n)$ (preprocessing) time using complicated techniques whose implementation seems extremely difficult. The size of the data structure that is created by the preprocessing step is $O(n^2)$. 

-9-
O'Rourke et al[24] use Sharir and Schorr's ideas to extend the problem to a polyhedral surface which is no more supposed to be convex. (However, it should be orientable.) The shortest path that they calculate is only a "geodesic", i.e., it is confined to the surface and thus may not be the true shortest path. Their algorithm runs in $O(n^5)$ time. Furthermore, it does not follow a locus approach: using their algorithm on each new query (goal point) would take $O(n^4)$ time.

Mitchell and Papadimitriou[23] give an algorithm to solve this last problem in a locus setting. (It is noted that they are still computing geodesics, not true shortest paths.) Theirs is an $O(n^2 \log n)$ time algorithm for subdividing the surface of an arbitrary polyhedron (possibly of positive genus) so that the length of the shortest path from a given source to any goal on the surface is obtainable by simple point location. It has striking similarities to Dijkstra's method for shortest paths in graphs. As in our algorithm to be presented in section 3, point location is achieved in time $O(\log n)$, after which the actual shortest path is backtracked in time proportional to the number of faces that it traverses on the boundary.

1. Shortest Paths on a Convex Polyhedron

Consider the following specific instance of FINDPATH:

BOUNDARY FINDPATH
INSTANCE: Convex polyhedron $P$, specified points $s, g \in \text{bd}P$
where $s \neq g$.

QUESTION: Which $C \in \text{bd}P$ has the shortest length?

Before we solve this problem we give an argument as to its importance. Let $P = \{P_1, \ldots, P_n\}$ be a set of convex polyhedra. If we are allowed to compute a reasonably short (but not the shortest) $s$-to-$g$ path then we can pursue the following strategy. Let $P'$ be the subset of $P$ such that every member of $P'$ is intersected by $sg$. (If a polyhedron is intersected by $sg$ only once then it is not included in $P'$; thus $P'$ consists of polyhedra intersected by $sg$ at precisely two points.) Applying BOUNDARY FINDPATH to each member of $P'$ we obtain an $s$-to-$g$ polygonal path which comprises two types of curves: line segments through free space, and polygonal paths along the boundaries of the polyhedra between where the path "lands" from free space and "takes off" again (figure 1). Repeated optimization of this path is possible and will frequently yield a better (with fewer bend points) and shorter path (figure 2) although it is not difficult to come up with cases where repeated optimization might cause clashes.

We shall assume that the boundary representation is used to define a
polyhedron $P$. In this representation scheme each vertex is defined by its $x,y,z$ coordinates and each face is given as a list of pointers to the vertices, ordered in counterclockwise around the boundary of the face with respect to a point above it. It is convenient to think of vertex labels or face labels as positive integers.

DEFINITION The face graph ($F_{\text{graph}}$) of a convex polyhedron $P$ is an undirected graph $F_{\text{graph}} = (FV, FE)$ with unit arc weight, $FV = \{i : F_i$ is a face of $P\}$ and $FE = \{(i,j) : F_i$ and $F_j$ are adjacent$\}$. (Two faces are adjacent if they share an edge.) □

EXAMPLE Figure 3(a) shows the face graph of a cube. In figure 3(b), the face graph of a parallelepiped is given to note that $F_{\text{graph}}$ only preserves the adjacency information. Figure 3(c) shows that two faces with a common point only are not considered adjacent by this definition. □

DEFINITION Let $C \in \text{bd}P$ be an $s$-to-$g$ polygonal path. The sequence of faces that $C$ enters defines the face visit sequence of $C$ which will be denoted by $f\text{vs}C$. □

Thus $f\text{vs}C$ is a walk in $F_{\text{graph}}$ between the nodes corresponding to faces $F_s$ and $F_g$, the faces of $P$ containing $s$ and $g$, respectively. An immediate consequence of the above definition is:

LEMMA 1.1 Let $C' \in \text{bd}P$ be an $s$-to-$g$ shortest path. Then $f\text{vs}C'$ is a simple walk in $F_{\text{graph}}$.

Proof. If this is not true then $C'$ enters a face at least twice. Recalling the fact that the faces of $P$ are all convex, we can then further shorten $C'$, a contradiction. □

From now on, all face visit sequences will therefore be assumed to be simple.

In addition to $F_{\text{graph}}$, a useful construct that will be used by BOUNDARY FINDPATH is the planar development of a given face visit sequence. It is well-known that the boundary of a convex polyhedron has the structure of a planar graph. Therefore, the totality of the faces of $P$, situated in three-space in certain mutual relationship, can be represented in two-space (specifically the $xy$-plane) by a system of polygons identified with the faces of $P$. The relationship between a planar development and the planar polygonal polygonal schema will be apparent after the following description of how to obtain the latter.

In the $xy$-plane associate with each face of $P$ a polygon having the same metrical form. (Two polygons have the same metrical form if they can be
made to coincide by translations and rotations.) Define the glue relationship between the pairs of edges of these polygons such that two glued edges come from the same edge of $P$. Figure 4 illustrates this for a cube. Each edge in the planar polygonal schema is glued to exactly one edge.

**Definition** A planar development corresponding to a face visit sequence $1, \ldots, k$ is a union of polygons $F_1, \ldots, F_k$ of the planar polygonal schema of $P$. In the planar development two polygons $F_i$ and $F_{i+1}$, $1 \leq i < k$ are united along the edge that they are glued to each other, and do not overlap.

**Definition** The image of a point on a polyhedron under a planar development is the point in the plane that it ends up under the development.

**Definition** A planar development is legal if the line segment connecting the images of the source and the goal is internal to the development.

**Example** Figure 5 shows several planar developments computed and drawn by SP, our shortest path workbench (cf. appendix). The objects are as follows. Figure 5(a): cube, figure 5(b): icosahedron, figure 5(c) and (d): dodecahedron. It is noted that the last development is not legal.

It should be apparent that once a planar development is built, it can be moved to any position and orientation in the plane without changing the intrinsic geometry of the paths. We shall now give a procedural definition of a planar development:

**Definition** To compute the planar development of a face visit sequence $1, \ldots, k$, start with $F_1$. If $aff F_1$ is parallel to the $xy$-plane then translate $P$ by a suitable amount so that $F_1$ is now in the $xy$-plane; otherwise, let the dihedral angle between $aff F_1$ and the $xy$-plane be $D$ and and rotate $P$ about the line $aff F_1 \cap xy$-plane by $D$ to map $F_1$ to the $xy$-plane. The remaining faces $F_i$ are inductively handled as follows. Let $e$ be the common edge of $F_{i-1}$ and $F_i$ whose dihedral angle is $D$. Rotate $P$ by $D$ about $aff e$ to place $F_i$ to the $xy$-plane while avoiding overlaps with $F_{i-1}$'s polygon which is already there.

Now we are ready for:

**Algorithm** BOUNDARY FINDPATH
1. Let $F_s$ and $F_g$ be the faces of $P$ including $s$ and $g$, respectively. Assume that $F_s \neq F_g$; otherwise the shortest path is $C = sg$.
2. Let $FVS$ be the set of all simple walks in $F_{\text{graph}}$ of $P$, between the nodes corresponding to $F_s$ and $F_g$. Initialize $vs = \emptyset$ and $l^* = +\infty$.
3. For each member of $FVS$ do the following steps:

3.1 Compute the planar development corresponding to this face visit sequence. Let $s'$ and $g'$ be the images of $s$ and $g$ in the $xy$-plane under the same development. (They can be computed along with the planar development.)

3.2 If the development in step 3.1 is not legal then continue with step 3. Otherwise, if $d(s', g') < l'$ then replace $vs'$ with the current face visit sequence, let $l' = d(s', g')$, and continue with step 3.

4. At this point $l'$ is the length of the shortest path and $vs'$ is the face visit sequence that should be used to compute the shortest path itself. To do this, first compute the planar development of $vs'$ (and $s'$ and $g'$) and intersect $s'g'$ with all pairwise common edges of the polygons in the development. The intersection points in the plane are then easily used to compute the bend points of the shortest path $C'$. We know from the planar development of $vs'$ the distance of an intersection point from a vertex in the plane. All we need is to identify the vertex of $P$ in three dimensions that led to this vertex. Then marking the point which is away from this vertex the same distance over the edge touched by the shortest path we locate the bend point for one intersection. The others are found completely analogously.

End

An efficient way of checking whether a planar development is legal follows. Let $e_1, \cdots, e_t$ be the sequence of edges that are glued in the development. Then the development is legal if $s'g'$ intersects every $e_i$. Note that this is always easier than testing if $s'g'$ is internal to the planar development’s boundary.

To list the simple walks between two nodes of a graph we can use the algorithm of Yen[36] which works in $O(kn^3)$ if there are $n$ nodes in the graph and we require the first $k$ simple walks in increasing walk length. Katoh et al[15] give an improved algorithm for the same task with a time complexity $O(kf(n,m))$ under the assumption that shortest walks form one node to all others can be found in $f(n,m)$ time where $m$ is the number of arcs in the graph. Since $f(n,m)$ is either $O(n^2)$ or $O(m \log n)$ in the worst case, this algorithm is more efficient than Yen’s.

Determining the value of card$FVS$ is difficult. Garey and Johnson[12] state that the following problem is NP-hard:

K-th SHORTEST PATH
INSTANCE: Graph $G = (V,E)$, positive integer lengths $l_e$ for each $e \in E$, specified nodes $s,t \in V$, positive integers $b$ and $k$.
QUESTION: Are there $k$ or more distinct simple walks from $s$ to
They also mention that K-th SHORTEST PATH remains NP-hard even if $l_e = 1$ for all $e \in E$, and is solvable in pseudo-polynomial time (polynomial in \(\text{card} \, V, \, k, \) and \(\log b\)) and accordingly, in polynomial time for any fixed $k$ (e.g., Yen's algorithm). The difficulty of K-th SHORTEST PATH basically resides in the following counting problem which was proven to be \#P-complete by Valiant[34]:

**S-T PATHS (SELF-AVOIDING WALKS)**

**INSTANCE:** Graph $G = (V,E)$, specified nodes $s,t \in V$.

**QUESTION:** What is the number of walks from $s$ to $t$ that visit every node at most once?

The problem of counting $(s,t)$-walks in $(s,t)$-planar graphs is also \#P complete[28]. (A graph is called source-sink planar, or $(s,t)$-planar in short, if it has a planar representation with nodes $s$ and $t$ on the boundary.) Provan[29] states that the approximation problem for $(s,t)$-walks is unsolved, in the sense that the following problem is open:

"For any fixed $\epsilon < 1$, does there exist a polynomial algorithm which for a given $(s,t)$-planar graph $G$, will give an approximation $N_0$ for the number $N$ of $(s,t)$-walks in $G$ which satisfies $|N - N_0| < \epsilon N$?"

On the other hand, for any fixed $\epsilon > 0$, can it be proven that the above problem is NP-hard? Currently, the only known approximations are to count the minimum length walks or to enumerate as many walks as possible (using a large value of $k$ in Yen's algorithm, for example).

It is not hard to find a convex polyhedron which has an exponential number of simple walks in its $F$graph (figure 6). If there are $l$ lateral faces of this pyramid-like object (not counting the small triangular faces) then the number of simple walks between the nodes $F_s$ and $F_g$ in the figure is $\Omega(2^{l/2})$. This result also shows that our BOUNDARY FINDPATH algorithm is of exponential time complexity in the number of faces of $P$. As mentioned before, there exist polynomial algorithms by other researchers for this problem. Unfortunately, theirs do not seem to admit practical implementations. In the light of this, the algorithm presented in this section is applicable for objects of moderate complexity. We can also try to test only a certain section (e.g., first few in increasing walk length) of the face visit sequences between the source and the goal faces for an object with many faces with the hope that the shortest path is generated by a short face development sequence. The shortest path rendered by the legal developments found among the developments that
these sequences give may be taken as the true shortest path although this is certainly vulnerable to an adversary.

2. Shortest Paths around a Convex Polyhedron

Now we inspect the following variant of BOUNDARY FINDPATH:

**EXTERIOR FINDPATH**

**INSTANCE:** Convex polyhedron $P$ and points $s, g$ where at least one of them is external to $P$, $s \neq g$.

**QUESTION:** Which $C \in C(s, g; P)$ has the shortest length?

Without loss of generality, we shall treat the case where $s$ and $g$ are both outside $P$. In this case, the following fact is useful:

**LEMMA 2.1** Let $H = \text{conv}(\{s, g\} \cup \text{vert} P)$. Then an $s$-to-$g$ shortest path for EXTERIOR FINDPATH is entirely on $\text{bd} H$.

**Proof.** Omitted. \(\Box\)

Thus, once $H$ is computed using standard three-space convex hull algorithms, we can apply BOUNDARY FINDPATH to the instance made of $H$, $s$, and $g$. (Preparata and Hong[26] give an algorithm to find the three-dimensional convex hull in $O(n \log n)$ time for $n$ points.) However, there is a slight difficulty with this approach. Assuming that we want to know which edges of the original polyhedron the shortest path touches, we must keep extra information with $H$, i.e., which vertex of $H$ comes from which vertex of $P$. Below, we shall give a more direct method to obtain $H$ while keeping this information implicitly using a visibility-based approach. (Sutherland et al[33] give an overview of polyhedral visibility.)

**DEFINITION** For a convex polyhedron $P$ and a point $x$ external to it, the **silhouette edges** of $P$ are members of

$$\{e : e \in F_1 \text{ and } e \in F_2 \text{ where } F_1 \in \text{vis and } F_2 \in \text{invis}\}$$

Here, $F_{\text{vis}}$ (resp. $F_{\text{invis}}$) is the set of visible (resp. invisible) faces of $P$ from viewpoint $x$. \(\Box\)

Clearly, $F_{\text{vis}} \cap F_{\text{invis}} = \emptyset$ since a face of a convex polyhedron is either completely visible or completely hidden to an observer. Let $E_{\text{sil}, s}$ (resp. $E_{\text{sil}, g}$) denote the silhouette edges of $P$ from $s$ (resp. $g$). It is clear that the faces of $H$ will be the union of three *disjoint* sets:
bdH = F_{tri,s} \cup F_{tri,g} \cup (F_{invis,s} \cap F_{invis,g})

Here \( F_{tri,s} \) (resp. \( F_{tri,g} \)) is a set of triangular faces each characterized by an edge of \( E_{sil,s} \) (resp. \( E_{sil,g} \)) and \( s \) (resp. \( g \)). In essence, these are the lateral faces of a pyramid-like object with (generally nonplanar) basis \( E_{sil,s} \) (resp. \( E_{sil,g} \)) and apex \( s \) (resp. \( g \)). It is noted that the geometric complexity of object \( H \) is the same as with \( P \).

We conclude this section with an algorithm to compute the silhouette edges of \( P \) from a point \( x \) external to it:

**Algorithm** SILHOUETTE
1. Compute \( F_{vis,x} \), the visible faces of \( P \) from viewpoint \( x \), by checking line segments \( xc_i \) where \( c_i \) is the center of mass of face \( F_i \) against all \( F_j, j \neq i \) for intersection. \( F_{vis,x} \) consists of all \( F_i \) which do not cause any intersection.
2. Let the totality of the visible edges of \( P \) from \( x \) be \( E_{vis,x} = \{ e : e \in F \text{ where } F \in F_{vis,x} \} \). Sort \( E_{vis,x} \) and eliminate both of duplicate members. The remaining edges are precisely the edges of \( E_{sil,x} \).

*End*

Figure 7 demonstrates the working of EXTERIOR FINDPATH on a simple object.

3. Partitioning the Boundary of a Convex Polyhedron

Finally, consider the following variant of BOUNDARY FINDPATH:

**BOUNDARY FINDPATH** (locus)

**INSTANCE:** Same as in BOUNDARY FINDPATH except that \( g \) is not given. However, it is guaranteed that, when specified, \( g \) will be on \( \text{bd}P \).

**QUESTION:** Preprocess \( P \) so that for any specified \( g \in \text{bd}P \) the \( s \)-to-\( g \) shortest path is computed efficiently.

A practical case which would benefit from BOUNDARY FINDPATH (locus) is as follows. Consider a truck on the surface of a mountain modeled as a convex polyhedron, say, a pyramid. Suppose that the truck is required to carry material from a fixed location on the surface to several points, say, construction sites. Then, it is reasonable to compute the shortest routes for the truck (approximated as a point) more efficiently than can be achieved by repeated applications of BOUNDARY FINDPATH for each specified destination.

This is a powerful paradigm of computational geometry known as the *locus*
approach and is studied in Overmars[25]. In solving a problem using this approach, we are allowed to spend some initial effort (i.e., preprocessing) to construct a data structure which will let us answer future requests (i.e., queries) quickly. To be effective, this assumes two things. First, the number of the query points must be large to validate such an initial effort. Second, the data structure must embody succinctly the locus of the required solution and must enjoy the existence of a fast search procedure to retrieve it.

We shall now summarize one such data structure suitable for solving BOUNDARY FINDPATH (locus). The data structure is known as the Voronoi diagram and was first introduced to computational geometry by Shamos[31]. Let $S = \{x_1, \ldots, x_n \}$ be a subset of $\mathbb{R}^2$. For $1 \leq i \leq n$ let

$$\text{regnx}_i = \{y : d(x_i, y) \leq d(x_j, y) \text{ for all } j \}$$

be the Voronoi region of point $x_i$. The Voronoi diagram of $S$, denoted by $\text{vor}S$, partitions the plane into $\text{card}S = n$ regions, one for each member of $S$. The (open) Voronoi region of point $x_i$ consists of all points of $\mathbb{R}^2$ closer to $x_i$ than any other point of $S$. For $1 \leq i, j \leq n$, letting $H_{ij} = \{y : d(x_i, y) \leq d(x_j, y)\}$ (the halfspace defined by the perpendicular bisector of $x_i, x_j$), it is seen that for $i \neq j$, $\text{regnx}_i = \bigcap H_{ij}$. Thus, regnx$_i$ is a convex polygonal region and $\text{vor}S$ is equal to the union of the boundaries of all regnx$_i$. For every vertex $x$ of $\text{vor}S$ there are at least three points $x_i$, $x_j$, $x_k$ in $S$ such that $d(x, x_i) = d(x, x_j) = d(x, x_k)$. The Voronoi diagram for a set of $n$ points has at most $2(n - 2)$ vertices and $3(n - 2)$ edges[22].

The Voronoi diagram of $S$ can be computed in $O(n \log n)$ time [30] and this is optimal with respect to a wide range of computational models[17]. Unfortunately, practical Voronoi programs which are both fast and reliable are difficult to write mainly due to the special cases in the diagram that are to be handled precisely. Among the published algorithms, those by Avis and Bhattacharya[2], Lee[19], and Guibas and Stolfi[14] seem to be more promising for practical use. On the other hand, it is possible to construct slower implementations which handle special cases without much effort.

We now summarize how to search Voronoi diagrams in logarithmic time in the number of the edges, $n$, of the diagram, i.e., we cite methods which let one find the point $x \in S$ such that for a query point $y \in \mathbb{R}^2$, $d(x, y)$ is minimum. The search methods we shall review are more general than searching Voronoi diagrams in that they are based on searching a planar subdivision, i.e., a straight-edge embedding of a planar graph. The underlying problem is generally known as planar point location in computational geometry. Let us call a subset of the plane monotone if its intersection with any line
parallel to the \( y \)-axis is a single interval (possibly empty). A subdivision is monotone if all its regions are monotone. Mehlhorn[22] shows that a simple planar subdivision (a subdivision with only triangular faces) can be searched in time \( O(\log n) \) after spending preprocessing time \( O(n) \) and storage space \( O(n) \). He also gives the following property to show that the last two bounds apply to general subdivisions also, with a small penalty in preprocessing time:

**Lemma 3.1** If the searching problem for simple planar subdivisions with \( n \) edges can be solved with search time \( O(\log n) \), preprocessing \( O(n) \), and space \( O(n) \), then the searching problem for general subdivisions with \( n \) edges can be solved with the same search time and space but preprocessing \( O(n \log n) \). If all faces of the generalized subdivision are convex then \( O(n) \) preprocessing is sufficient.

*Proof.* Mehlhorn[22]. (Lee and Preparata[18] also show that an arbitrary subdivision with \( n \) edges can be refined to a monotone subdivision having at most \( 2n \) edges in \( O(n \log n) \) time.) □

Now we shall review some planar point location algorithms in the literature which either achieve these bounds or come close. Dobkin and Lipton[3] were the pioneers to obtain an \( O(\log n) \) query time but they use \( O(n^2) \) space. Preparata[27] modifies their method to prove that \( O(n \log n) \) space is sufficient. His solution is implementable. In an important paper Lee and Preparata[18] give an algorithm which is based on the construction of separating chains. Their algorithm achieves \( O(n \log n) \) preprocessing and \( O(n) \) space yet has a query time of \( O(\log^2 n) \). The constants hidden in these expressions are small and make their algorithm practically useful. (Their algorithm works for monotone subdivisions only.) Edelsbrunner and Maurer[4] give a space-optimal solution which works for general subdivisions (and even families of nonoverlapping subdivisions). Their query time is \( O(\log^3 n) \). Lipton and Tarjan[20, 21] give a method with \( O(\log n) \) query time and \( O(n) \) preprocessing and space (and thus optimal in all respects). Although based on a graph separator theorem which has many far-reaching consequences, they admit that their method is of only theoretical interest because of the implementation difficulties. Kirkpatrick[16] gives another method with the same bounds. His method builds a hierarchy of subdivisions and seems to be implementable. Finally, Edelsbrunner et al[5] give a substantial improvement of the technique of Lee and Preparata and again attain the optimal bounds in all three respects. The importance of their method is that it seems to admit an efficient implementation along with extensibility to subdivisions with curved edges.

Now we are ready to present our algorithm for BOUNDARY FINDPATH (locus). In the following we show the partitioning for only a face of \( P \) (other than \( F_s \)) which we shall denote by \( F_g \). To partition \( \text{bd} P \), we apply the
following algorithm for each face. (Note that no partitioning is necessary for $F_s$ due to convexity.)

**Algorithm** BOUNDARY FINDPATH (locus): Preprocessing
1. Find all simple face visit sequences between $F_s$ and $F_g$ using $Fgraph$ of $P$.
2. For each face visit sequence found in step 1, compute the image of $s$ with respect to the planar development which starts with face $F_g$ and ends with $F_s$. (Note that we are *not* required to compute the whole development, but just the image point.)
3. Compute the Voronoi diagram of the image points calculated in step 2 using standard Voronoi programs mentioned above. It is required that with each image point (Voronoi center) we store the face visit sequence which is used to arrive at it.
4. Clip the diagram obtained in step 3 with respect to "window" $F_g$ so as to preserve only those parts of it within the polygon $F_g$. (Any of the standard graphics algorithms such as the one due to Sutherland and Hodge[7] can be used for clipping.)

**End**

Assume that for a given $P$, the above preprocessing is carried out for all faces (except $F_s$). Thus, we have a family of planar subdivisions for each face such that for each region of a given subdivision we know the ordered sequence of faces to be developed to the plane if $g$ specified within that region. More specifically, we can apply the following algorithm when we are queried with a new $g$:

**Algorithm** BOUNDARY FINDPATH (locus): Querying
1. Compute the face $F_g$ holding a given $g$. If $F_g = F_s$ then the shortest path is trivially $sg$.
2. Using standard planar point location algorithms mentioned above locate $g$ inside the planar subdivision belonging to $F_g$.
3. Since we stored the face development sequence used to arrive this region we can now use it to compute the $s$-to-$g$ shortest path. Note that there may be cases where $g$ will be shared by at least two regions and thus there will be at least two shortest paths each of which is obtained via a different development sequence.

**End**

Figure 8 shows the Voronoi partitioning on a face of a cube using the above algorithm. This figure was drawn by SP. It is noted that in figure 8(a) the given face is partitioned into 4 regions whereas in figure 8(b) this number becomes 6. This effect was obtained by simply moving the source to another location on the source face.
In general, the number of regions a given face $F_g$ is partitioned by this algorithm is expected to be small. An informal argument for this is as follows. Consider the images of $s$ in the plane of $F_g$. If the face visit sequence for a particular image is long then the image will usually end up in a point farther away from $F_g$ compared to another image obtained by a shorter visit sequence. Thus, only a small portion of the possible face visit sequences between $F_s$ and $F_g$ can render images in the plane close to $F_g$ and contribute to the Voronoi diagram on it.

Acknowledgment

The material reported herein has been supported by the National Science Foundation under grants ECS 80-21504 and ECS 85-51942.

-- o --

Appendix: SP - A Workbench to Compute Shortest Paths

SP is a family of programs written in Franz Lisp[6] and Macsyma command language[13] to experiment with shortest paths. A detailed description of SP is given in [1].

SP was designed with the following philosophy. Let $W$ be a workspace (e.g., a bounding box) which includes a set of polyhedral obstacles. SP is given a geometric description of the members of $W$ and from that point on should be able to compute shortest paths inside $W$ between given pairs of points using the algorithms some of which presented in the preceding sections. It is imperative that SP has some graphics facilities and can supply the user with views of $W$ so that she can have an intuitive feeling about the correctness of a particular computation. In that sense, SP resembles to Verrilli's system[35]; it provides the user with facilities to carry out needed computations, but at the same time needs her intervention here and there. Following a rapid prototyping approach, we either simply excluded those computations which we do not currently know how to perform effectively, or reformulated them to be controlled by user advice at certain points.

Currently, one can only work with a single convex polyhedron using the Franz part of SP. There are facilities to implement BOUNDARY FINDPATH, EXTERIOR FINDPATH, and BOUNDARY FINDPATH (locus). It is also possible to implement an approximate FINDPATH algorithm for a workspace with several convex polyhedra as outlined in section 1 and depicted in figures 1 and 2. Using Macsyma parts of SP it is possible to compute shortest paths in a general workspace with many objects (which may be nonconvex) although this is not fully automated in the light of the combinatorial
explosion that known FINDPATH algorithms have. (Nevertheless, if the user specifies the list of edges that the shortest path must visit, then the problem is solved without much effort.) It is also possible (using Macsyma) to work on FINDPATH (locus) (which is the general version of BOUNDARY FINDPATH (locus)) although this is not automated yet. (In [11] all computations were done in this way.)

Finally, we give some additional examples computed and drawn by SP. Figure 9(a) and (b) show two shortest paths on the boundary of a dodecahedron. Similarly, figure 10 shows a shortest path on the boundary of an icosahedron. Figure 11 demonstrates a shortest path around a cube. This was computed after constructing a new object via EXTERIOR FINDPATH and then applying BOUNDARY FINDPATH on it. Figure 12 shows the partitioning of the boundary of a cube in the presence of a source point on face 1. Figure 12(a), (b), (c), (d), and (e) respectively depict the regions induced on goal faces 2, 3, 4, 5, and 6.

References


Figure 1 A reasonably short path between \( s \) and \( g \) in the presence of convex polyhedral obstacles.
Figure 2 Further optimization of the path shown in figure 1 yields a shorter path with fewer bend points.
Figure 3 Face graphs of convex polyhedra:
(a) cube, (b) prism, (c) pyramid.
Figure 4 The planar polygonal schema of a cube.
Figure 5 Some planar developments computed and drawn by SP.
The objects that are developed are as follows: (a) cube, (b) icosahedron, (c) and (d) dodecahedron.
Face development sequence: 8 13 16 9
Source face no.: 7
Goal face no.: 9
Source point coords.: 0.948 0.230 0.504
Goal point coords.: 0.192 1.206 0.504
Source point coords. (plane): 0.289 0.500
Goal point coords. (plane): 1.154 2.000
Source-to-goal distance: 1.732
<table>
<thead>
<tr>
<th>Face development sequence:</th>
<th>3</th>
<th>1</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source face no.:</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal face no.:</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source point coords.:</td>
<td>0.380</td>
<td>1.447</td>
<td>0.616</td>
<td></td>
</tr>
<tr>
<td>Goal point coords.:</td>
<td>0.996</td>
<td>-0.447</td>
<td>1.612</td>
<td></td>
</tr>
<tr>
<td>Source point coords. (plane):</td>
<td>0.688</td>
<td>0.500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal point coords. (plane):</td>
<td>-1.276</td>
<td>3.927</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source-to-goal distance:</td>
<td>3.950</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
X

Y

Face development sequence: 1 2 3 4 5 6 7 8 9 10 11 12
Source face no.: 1
Goal face no.: 12
Source point coords.: 0.688 0.500 0.000
Goal point coords.: 0.688 0.500 2.227
Source point coords. (plane): 0.688 0.500
Goal point coords. (plane): 6.782 3.927
Source-to-goal distance: 6.991
Figure 6 There exist an exponential number of simple walks between nodes $F_s$ and $F_g$ in $Graph$ of this object.
Figure 7 Demonstration of how EXTERIOR FINDPATH works via silhouettes.
Figure 8 Demonstration of BOUNDARY FINDPATH (locus) on a face of a cube. In figure 8(a) there are 4 regions on the goal face (the shaded polygon) as a result of Voronoi partitioning. Figure 8(b) shows the effect of moving the source on the source face to another location.
**Figure 9** Shortest paths on the boundary of a dodecahedron.

Both (a) and (b) are shortest paths. This is a perspective view of the object as computed by SP.
Visible face no.:
0
13

Invisible face no.:
1
2
3
4
5
6
7
8
9
10
11

Viewpoint:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.500</td>
<td>0.500</td>
<td>2.500</td>
</tr>
</tbody>
</table>

Source face no.:
1

Goal face no.:
7

Source point:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.288</td>
<td>0.500</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Goal point:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.118</td>
<td>-0.085</td>
<td>1.011</td>
</tr>
</tbody>
</table>

Face development sequence:
1 2 7

Shortest path length:
2.818

Shortest path bend points:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.288</td>
<td>0.500</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.370</td>
<td>0.000</td>
</tr>
<tr>
<td>-0.468</td>
<td>-0.085</td>
<td>0.898</td>
</tr>
<tr>
<td>-0.118</td>
<td>-0.085</td>
<td>1.011</td>
</tr>
</tbody>
</table>
Figure 10 A shortest path on the boundary of an icosahedron. This was computed by SP.
Figure 11 A shortest path around a cube. This was computed by SP after computing the new object and then applying BOUNDARY FINDPATH on it.
Figure 12 This shows the partitioning of the boundary of a cube in the presence of a source on face 1. Parts (a), (b), (c), (d), and (e) respectively show the regions induced on goal faces 2, 3, 4, 5, and 6. These figures were computed by SP.
Development sequences:
1: 4 1
2: 4 5 1
3: 4 3 1
4: 4 5 2 1
5: 4 6 5 1
6: 4 3 3 1
7: 4 8 3 1
8: 4 8 2 1
9: 4 5 6 1
10: 4 2 3 1
11: 4 6 5 3 1
12: 4 3 6 5 1
13: 4 3 2 3 1
14: 4 6 3 3 1
15: 4 6 2 3 1
16: 4 5 6 2 1
17: 4 3 6 2 1
18: 4 5 2 6 3 1
19: 4 8 5 2 3 1
20: 4 3 5 6 3 1
21: 4 3 3 6 5 1
22: 4 6 3 3 3 1
23: 4 5 6 2 3 1
24: 4 3 6 5 5 1

Source face no.: 1
Goal face no.: 4
Source point co-ords: 0.000 0.350 0.350
No. of developments: 24
Development sequences:
1: 5 1
2: 5 4 1
3: 5 2 1
4: 5 4 1 1
5: 5 2 1
6: 5 2 1
7: 5 2 1
8: 5 4 1
9: 5 4 1 1
10: 5 4 3 1
11: 5 4 1 1
12: 5 2 4 1
13: 5 2 4 1
14: 5 2 4 1
15: 5 4 3 1
16: 5 4 3 1
17: 5 4 3 1
18: 5 4 3 1
19: 5 4 3 1
20: 5 4 3 1
21: 5 4 3 1
22: 5 4 3 1
23: 5 4 3 1
24: 5 4 3 1

Source face no.: 1
Goal face no.: 5
Source point coords.: [0.000, 0.250, 0.250]
No. of developments: 24