International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

Shortest Paths on a Polyhedron, Part I: Computing Shortest Paths*

JINDONG CHEN *and* YIJIE HAN
Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA

Received 14 September 1990
Revised 15 July 1992
Communicated by C. Yap

ABSTRACT

We present an algorithm for determining the shortest path between any two points along the surface of a polyhedron which need not be convex. This algorithm also computes for any source point on the surface of a polyhedron the inward layout and the subdivision of the polyhedron which can be used for processing queries of shortest paths between the source point and any destination point. Our algorithm uses a new approach which deviates from the conventional "continuous Dijkstra" technique. Our algorithm has time complexity $O(n^2)$ and space complexity $\Theta(n)$.

Keywords: Shortest path, motion planning, nonconvex polyhedron, single source shortest path, computational geometry, computational robotics.

1. Introduction

Recent research interest in the field of robotics, terrain navigation, and industrial automation has promoted the study of motion planning. One of the basic problems in motion planning is to find the shortest path. The shortest path problem has several versions. The shortest path problem in the two-dimensional situation with polygonal obstacles can be solved by constructing a visibility graph.[9,11,12,13,23] The known fastest algorithm by Reif and Storer[19] achieves time $O(nk + n \log n)$, where n is the number of vertices and k is the number of disjoint simple polygons. Certain special cases in the two-dimensional situation have also been identified with shortest path problem solvable in $O(n \log n)$ time.[10,13,23] The shortest path problem in the general 3-dimensional situation is $NP-$hard[4] and only exponential time algorithms

*Research supported in part by grant from University of Kentucky Center for Robotics and Manufacturing Systems. A preliminary version of the paper was presented on the 1990 Annual ACM Symposium on Computational Geometry.
Authors' current address: Jindong Chen, Department of Computer Science, Purdue University, West Lafayette, IN 47907.
Yijie Han, Electronic Data Systems, Inc., 37350 Ecorse Rd., Romulus, MI 48174.

are known.[3,20,23] An important special case in the 3-dimensional situation is the shortest path problem along the surface of a single polyhedron. The single source shortest path problem asks the construction of the subdivision on the surface of the polyhedron such that the shortest path between a fixed source point and any destination point can be reported quickly.

The single source shortest path problem for a single *convex* polyhedron was first studied by Sharir and Schorr.[23] Their algorithm runs in $O(n^3 \log n)$ time, where n is a measure of the complexity of the scene which we may take as the number of edges of the polyhedral surface. After building the subdivision, the shortest path problem can be transformed into a point location problem and the shortest path from the fixed source point to a given query point can be computed in time $O(k + \log n)$, where k is the number of edges in the corresponding shortest path edge sequence.

An improved algorithm for a convex polyhedron was given by Mount[15] which reduces the running time to $(n^2 \log n)$. For an arbitrary polyhedron, which need not be convex, O'Rourke, Suri, and Booth gave an $O(n^5)$ algorithm.[17] Subsequently, Mitchell, Mount, and Papadimitriou presented an $O(n^2 \log n)$ algorithm.[14]

All the previous algorithms for the problem can be regarded as preprocessing of the polyhedron which builds a subdivision on the polyhedron surface such that points in a region of the subdivision have the same shortest path edge sequence. Our algorithm accomplishes the same task.

The essence of a technique called "continuous Dijkstra"[14] is used in the previous designs.[15,23] Mitchell *et al.*[14] were the first to formalize and generalize the technique, and gave the name of "continuous Dijkstra".[14] In an application of this technique,[14,15,23] certain entities (vertices, edges, or faces) are treated as nodes in a graph, and the entity of the shortest distance from the source point is always extracted for expansion as in the Dijkstra's algorithm.[7] To facilitate the extraction of the entity of the shortest distance, a priority queue of entities is maintained.[14,15,23]

In this paper we present a new algorithm for the single source shortest path problem on the surface of a polyhedron which need not be convex. Our algorithm uses a rather different approach. It does not have the features of the Dijkstra's algorithm. In particular, we do not maintain a priority queue and the expansion of entities in our algorithm is not necessarily performed upon those of the shortest distance from the source. Our algorithm has a very simple structure. The simplicity comes from our new observations of the problem. The observation "one angle one split"(Section 3) results in upper bounding the number of branches in the sequence tree. Yet another observation (Section 4) gives the new inward layout of the polyhedron which can be viewed as a dual of the outward layout of Sharir and Schorr.[23] This inward layout results in computing the Voronoi diagram once for all faces instead of once for each face as in the previous designs. The time complexity of $O(n^2)$ we have achieved is a consequence of these observations. Besides, we are able to maintain the sequence tree with merely $O(n)$ nodes, thus achieving optimal space complexity $\Theta(n)$ for our algorithm.

Our recent work[6] extends the results presented here. Previously, Sharir and Schorr[23] uses $O(n^2)$ space to store shortest path information for a convex polyhedron. Mount achieved space complexity $O(n \log n)$ for storing shortest path information. For a nonconvex polyhedron $O(n^2)$ space is achievable,[14] but no result better than $O(n^2)$ has been claimed. We have a scheme for storing the shortest path information in $O(n \log n / \log d)$ space to support the processing of a query in $O(d \log n / \log d)$ time, where $1 < d \leq n$ is an adjustable integer.[6]

Parallel but independent of our work, Aronov *et al.*[1] and Aronov and O'Rourke[2] studied the inward layout of convex polyhedrons which they termed star unfolding of polyhedrons. Aronov and O'Rourke[2] proved that the inward layout of a convex polyhedron does not overlap. This result simplifies the computation of the Voronoi diagram for the inward layout in our algorithm in the convex polyhedron case. The time and space complexities of our algorithm are not affected by their results.

The rest of the paper is organized as follows. For a clear exposition, from section 2 through section 4 we restrict our discussion to the case of a convex polyhedron. In section 2 we review some preliminary knowledge of shortest paths on the surface of a polyhedron. In section 3 we present an algorithm for building the sequence tree with $O(n)$ branches. This algorithm establishes the fact that a shortest path from a source to a destination can be computed in $O(n^2)$ time and $\Theta(n)$ space. In section 4 we present the details of computing the inward layout and building the subdivision. In section 5 we adapt the algorithm to the nonconvex case. Conclusions are given in section 6.

2. Preliminary

Let P be a polyhedron. The surface of P consists of a set F of faces f_i, a set E of edges e_i, a set V of vertices v_i, a set Θ of angles θ_i. The angles formed by the edges of faces are used later to demonstrate the important property of shortest paths, namely "one angle one split." Let n be the complexity of the polyhedron, where n can be the numbers of edges, vertices, faces, and angles.

In order to obtain a uniform model easy to work with, we triangulate all the faces of the given polyhedron and also triangulate the face containing the source point S, so that S becomes a vertex, as is done in Ref. 14,15. This process takes $O(n)$ time, where n is the number of edges of the polyhedron.[5] After triangulation the complexity of the given polyhedron remains the same, *i.e.*, the number of edges is still $O(n)$.

Unfolding is a method used in studying shortest paths. Assume f_1, e_1, f_2, e_2,...,f_i, e_i, f_{i+1},..., e_{m-1}, f_m is a sequence of faces and edges, where $e_i = f_i \cap f_{i+1}$, and f_1 is a face whose boundary contains the source point S. The procedure of unfolding this sequence can be described as

```
$F := \{f_1\}$;
for $i := 1$ to $m-1$ do
    Rotate $F$ around $e_i$ until
        $F$, $f_{i+1}$ are co-plane and lie on
        different sides of $e_i$;
    $F := F \cup \{e_i, f_{i+1}\}$.
```

Upon finishing, all the faces are unfolded into a common plane $\mathcal{L}$. Any shortest path π passing through these faces is unfolded into a straight line segment $\overline{\pi}$.[23] We call the edge sequence $e_1, e_2, ..., e_i$ traversed by a shortest path *a shortest sequence*.

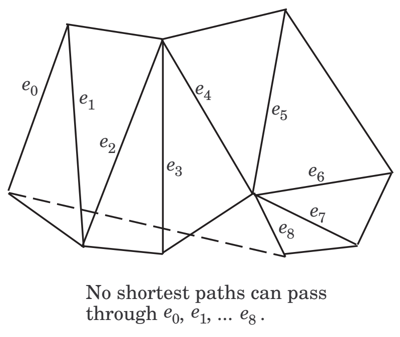

e_0
e_1
e_2
e_3
e_4
e_5
e_6
e_7
e_8
No shortest paths can pass through e_0, e_1, ... e_8 .

Fig. 1. A nonshortest sequence.

Certain edge sequences are not shortest sequences, *i.e.*, no shortest path can traverse such a sequence, as shown in Fig. 1. We can avoid these sequences by maintaining on each edge e_i *source image* I_{e_i}, the image of the source point S in the planar coordinate system of face f_i upon unfolding, and $Proj_{e_i}^{I_{e_i}}$, the projection of I_{e_i} on edge e_i which is a closed segment such that the shortest path to any point P in the segment through the sequence $e_1, e_2, ..., e_{i-1}$ can be unfolded into a straight line segment (such a locally optimal path is called a *geodesic path*[14,23])[a] (Fig. 2.). That is, for any point P in $Proj_{e_i}^{I_{e_i}}$ we can connect I_{e_i} and P with a straight line segment which traverses only the edges $\overline{e_1}$, $\overline{e_2}$, ...,$\overline{e_{i-1}}$, where $\overline{e_j}$ is the image of e_j in the planar coordinate system of face f_i upon unfolding. It is easy to compute the I_{e_i} and $Proj_{e_i}^{I_{e_i}}$ using $I_{e_{i-1}}$ and $Proj_{e_{i-1}}^{I_{e_{i-1}}}$. $Proj_{e_i}^{I_{e_i}}$ is the intersection of e_i and the *shadow* of $Proj_{e_{i-1}}^{I_{e_{i-1}}}$ on face f_i (Fig. 3.). We unfold f_{i+1} around e_i only if $Proj_{e_i}^{I_{e_i}}$ is not empty.

The following properties for shortest paths on a convex polyhedron are also known.[15,23]

(1) No face can be passed by a shortest path more than once. Therefore, a shortest path cannot pass more than n faces, where n is the total number of faces of the polyhedron.

(2) Two shortest paths cannot intersect each other except possibly at the source point or the destination point.

Since shortest paths do not intersect each other, the set of shortest paths to the vertices of the polyhedron can be arranged into a *circular order* according to the angles at which they emanate from the source.

A ridge point is a point having more than one shortest paths from the source point S.[23] If no vertex is a ridge point, the circular order for shortest paths to the vertices can be viewed as a circular order for vertices except the source (which can

[a]We use simplified notation I_{e_i} and $Proj_{e_i}^{I_{e_i}}$ here. In fact I_{e_i} and $Proj_{e_i}^{I_{e_i}}$ defined here depend on e_1, e_2, ..., e_{i-1}, e_i.

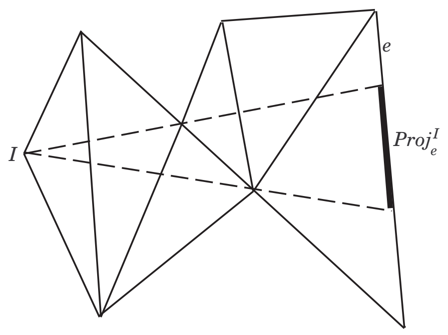


Fig. 2. Geodesic path.

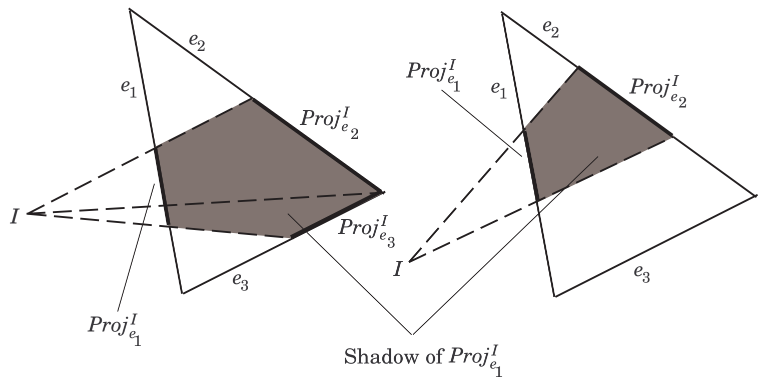


Fig. 3. Projection.

also be a vertex). When vertex v is a ridge point having k shortest paths, we may extend the circular order for vertices by creating k duplicates of v (including the original v) and have these duplicates arranged into the circular order. Thus we induce a circular order for the vertices of the polyhedron from the circular order of the shortest paths to the vertices.

3. Sequence Tree

Based on what we have discussed in the previous section, a naïve algorithm can be devised for the problem.

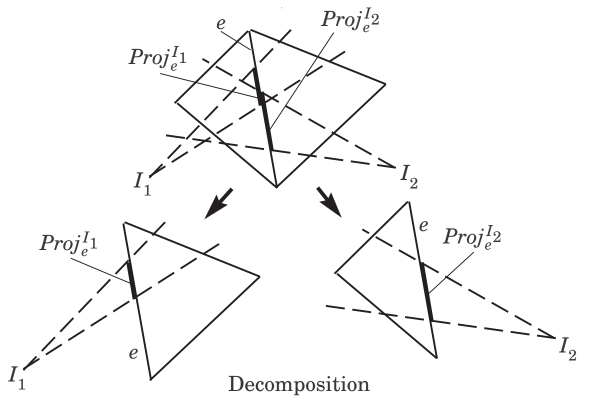


Fig. 4. Decomposition.

We decompose an edge into two oriented edges as in Ref 15, such that the image source can only be on the left side of the oriented edge (Fig. 4.). The term edge we used in the following text of this section is meant to be an oriented edge unless otherwise stated. The face to the other side of an oriented edge is the edge's *shadowed face*. A face can be the shadowed faces of three of its edges. We can view a face as having three layers each of which is a shadowed face of one of its edges. The following algorithm builds a tree called a sequence tree. A node in this tree other than the root is a triple, $n' = (e, I, Proj_e^I)$, where I is the image source of e. We say node n' is on edge e, e is the edge of n', and n''s shadow is the shadow of $Proj_e^I$.

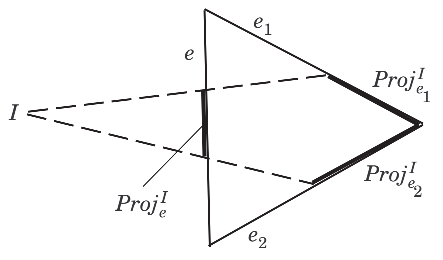
Algorithm 1

```
I.  $root := S$;
    for all the edges $e$ opposite to $S$ do
       insert $(e, S, e)$ as root's child;
    /* $e$ is an edge of the face containing $S$.*/
    /* $S$ is a vertex after triangulation.      */
II. for $i := 1$ to $n$ do
    /* $n$ is the number of the faces. */
       for all the leaves $(e, I, Proj_e^I)$ at the $i$th level do
```

```
begin
    unfold $I$ about $e$ to $\overline{I}$;
    /* $\overline{I}$ is co-planar with $\Delta ABC$,*/
    /* the shadowed face of $e$.        */
for $e' := AB, CA$ do
    calculate $Proj_{e'}^{\overline{I}}$;
    if $Proj_{e'}^{\overline{I}}$ nonempty then
        insert $(e', \overline{I}, Proj_{e'}^{\overline{I}})$ as
        $(e, I, Proj_e^I)$'s child;
end
```

Theorem 1: *All shortest sequences starting from S are among the path sequences computed in Algorithm 1.*

Proof: When the tree is built, each node in the tree defines an edge sequence which consists of those edges which are edges of the nodes on the way from the root up to this node. For a point p on face f_i such that p is contained in node D's shadow, node D guarantees the existence of a geodesic path to p through the edge sequences defined by node D. By selecting the node whose sequence corresponds to the shortest distance of p from S, we can find the shortest path for p. Algorithm 1 is correct because the shortest path can be unfolded into a straight line and the number of faces passed by the shortest path is no more than n. $\square$



Two children, $(e_1, I, Proj_{e_1}^I)$ and $(e_2, I, Proj_{e_2}^I)$

Fig. 5. Path splitting.

The tree could have an exponential number of nodes because all the nodes whose shadows cover the opposite angle have two children in the tree (Fig. 5.). We now prove that at most one node among those whose shadows cover the same angle can have two children.

Lemma 1: *Given on the same edge CB of ΔABC two nodes n_1 and n_2 whose shadows cover A, the vertex of $\angle CAB$, at most one of them can have two children which can be used to define shortest sequences.*

Proof: Unfold I_{n_1}, I_{n_2}, the source images of n_1 and n_2 around CB, to be coplanar with ΔABC (Fig. 6.). $\overline{I_{n_1}}$, $\overline{I_{n_2}}$, are the source images after unfolding. If $|\overline{I_{n_1}}A| < |\overline{I_{n_2}}A|$, then the shorter paths to points on AB and CA which are sufficiently close

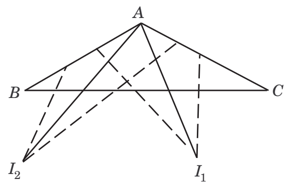


Fig. 6. Two sequences meet at vertex A.

to A are paths passing through the edge sequence represented by n_1, therefore n_1's two children can be used to define shortest sequences while only one of n_2's children can be used to define a shortest sequence. By the same reason, if $|\overline{I_{n_1}A}| > |\overline{I_{n_2}A}|$ then n_2's two children can be used to define shortest sequences while only one of n_1's children can be used to define a shortest sequence. Finally, if $|\overline{I_{n_1}A}| = |\overline{I_{n_2}A}|$, then n_1 *ties* n_2, and both n_1 and n_2 have at most one child which can be used to define a shortest sequence. □

With Lemma 1 we only let the node n' which is the closest to the angle fork into two children. We say n' *occupies* the angle. If we maintain "one angle one split" in our sequence tree, the number of leaves of the tree will be reduced to $O(n)$, because there are $O(n)$ angles in the polyhedron. When a leaf $n' = (BC, I_{BC}, Proj_{BC}^{I_{BC}})$ on edge BC of face ΔABC is being processed, if its shadow doesn't cover A, insert the only child it can have. If its shadow covers A, first check if it can occupy $\angle CAB$. If not, insert the only child it can have. Otherwise, insert its children and mark $\angle CAB$ as occupied by n'. If n'' is the node which previously occupied $\angle CAB$, clip off one of its children according to Lemma 1 and delete the subtree rooted at this child.

Algorithm 2

```
I.   $root := S$;
     for all the edges $e$ opposite to $S$ do
        insert $(e, S, e)$ as root's child;
II.  for $i := 1$ to $n$ do
        /* $n$ is the number of faces. */
        for all the leaves $n' = (e, I, Proj_e^I)$ at the $i$th level do
            begin
                unfold $I$ about $e$ to $\overline{I}$;
                    /* $\overline{I}$ is co-planar with $\Delta ABC$,*/
                    /* the shadowed face of $e$.      */
                calculate $Proj_{AB}^{\overline{I}}$, and $Proj_{CA}^{\overline{I}}$;
                /* the projections on $AB$,$CA$. */
                if $n'$'s shadow covers $A$ then
                    if $n'$ can occupy $\angle CAB$ over $n''$,
                    /* $n''$ previously occupied $\angle CAB$.*/
                    then
(a)                 clip off one of $n''$'s two children
```

```
                that cannot possibly define a
                shortest sequence and delete the
                subtree rooted at this child;
            insert $(AB, \overline{I}, Proj^{\overline{I}}_{AB})$,
                $(CA, \overline{I}, Proj^{\overline{I}}_{CA})$ as $n'$'s children;
            mark $\angle CAB$ as occupied by $n'$;
        else /* $n'$ cannot occupy the angle. */
            calculate $Proj^{\overline{I}}_{e'}$,
            /* $e'$ is either $CA$ or $AB$ whichever          */
            /* possibly defines a shortest sequence.          */
            insert $(e', \overline{I}, Proj^{\overline{I}}_{e'})$;
                as the sole child of $n'$;
    else /* $n'$'s shadow does not cover $A$ */
        $e'$ := either $AB$ or $CA$ which is nonempty;
        insert $(e', \overline{I}, Proj^{\overline{I}}_{e'})$ as the sole
            child of $n'$, accordingly;
end
```

Theorem 2: *All shortest sequences starting from S are among the path sequences computed in Algorithm 2.*

Proof: By Lemma 1, the only nodes deleted in algorithm 2 were those found incapable of defining a shortest sequence. Therefore, all shortest sequences are contained in the output tree. □

Theorem 3: *Algorithm 2 runs in $O(n^2)$ time.*

Proof: The property of "one angle one split" is maintained throughout the execution of Algorithm 2. Thus after an iteration of the loop in Algorithm 2, the tree has only $O(n)$ leaves. For the moment, consider only the time consumed in generating the tree, not the time consumed in deleting subtrees in step (a). Since there are $O(n)$ leaves after each iteration of the loop, it takes $O(n)$ time to generate the next level of the tree, thus resulting in a total of $O(n^2)$ time for generating the tree. Now consider the time consumed in step (a) for deleting the subtrees. It can be calculated by a counting trick. When we are generating the tree, we allow two time units for each node generated, one is used for generating the node, the other is saved (may be considered as labeled on the node). If the node is later deleted, then the saved time unit can be used. Thus the time complexity of Algorithm 2 is $O(n^2)$. □

It will be clear later on that the essential piece of information generated by Algorithm 2 is the collection of shortest paths to the vertices of the polyhedron. In order to record this information, we introduce the vertex node into the sequence tree. A vertex node is a triple (e, I, v), where v is a vertex, e is an edge, and I is a source image. Let a be a current node on edge CB in the sequence tree. a's children is to be generated. If a occupies or ties with another node on CB at $\angle CAB$, node (CB, I_{CB}, A) will be inserted as one of a's children. For a vertex node (e, I, v), we use pointers to link it to vertex v and edge e. After the sequence tree is built, the shortest distance to vertex v can be found by examining vertex nodes linked to v.

The shortest path to vertex v can then be traced out on the sequence tree.

We note that the information on shortest paths to vertices can be obtained from Algorithm 2 using merely $O(n)$ space. In the process of generating the sequence tree we need only keep the leaves and the interior nodes which have more than one child in the current tree. If a leaf in the current tree generates only one child at the next level, the leaf will be discarded and replaced by its child. The sequence tree thus generated will have more than one child for each of its interior nodes. For each angle, we store the node which occupies the angle. If there are several nodes which tie on the same angle we store the two outmost tying nodes for the angle. For two nodes n_1 and n_2 in the sequence tree, where n_2 is a child of n_1, we can trace out the sequence of edges from n_1 to n_2 by starting from n_1 and, at each angle, comparing the nodes stored for the angle to decide which edge of the angle is in the edge sequence. Thus the time taken for tracing out the edge sequence from n_1 to n_2 is proportional to the number of edges in the edge sequence.

In computing the inward layout in section 4 we need the circular order of the shortest paths to the vertices of the polyhedron. Initially the children of the root in the sequence tree can be arranged by the circular order. This circular order can be maintained if we take care when generating the next level of the sequence tree. When the sequence tree is built, the circular order of the shortest paths to the vertices of the polyhedron can be obtained by a traversal of the tree.

We also note that, if we treat the destination as a vertex, we obtain immediately the following corollary to Theorem 2:

Corollary: *The shortest path between two points can be computed in $O(n^2)$ time and $O(n)$ space.*

Note that if there are multiple shortest paths to vertex v, all of them can be found by examining the nodes of the form (e, I, v) in the sequence tree.

4. Subdivision

The purpose of computing the subdivision of the polyhedron is mainly to store the shortest path information for quick retrieval. The task of computing the subdivision used to be done[14,15,23] by computing the Voronoi diagram on each face of the polyhedron. Our scheme allows the computation of the Voronoi diagram to be done once for all faces of the polyhedron. We first compute the inward layout of the polyhedron. We then compute the Voronoi diagram on the layout. Both can be done in $O(n^2)$ time and $O(n)$ space. The problem of storing the shortest path information for retrieval is addressed at the end of the section.

4.1. Layouts

The planar layout of the surface of a convex polyhedron due to Sharir and Schorr[23] is obtained by cutting all the ridge lines. After cutting the ridge lines the surface of the polyhedron can be laid out on a plane. The layout is a star-shaped polygon. The edges of the polygon are the ridge lines. The vertices of the polygon are either vertices of the polyhedron or the Voronoi vertices on the ridge lines. Such a star-shaped layout is shown in Fig. 7.

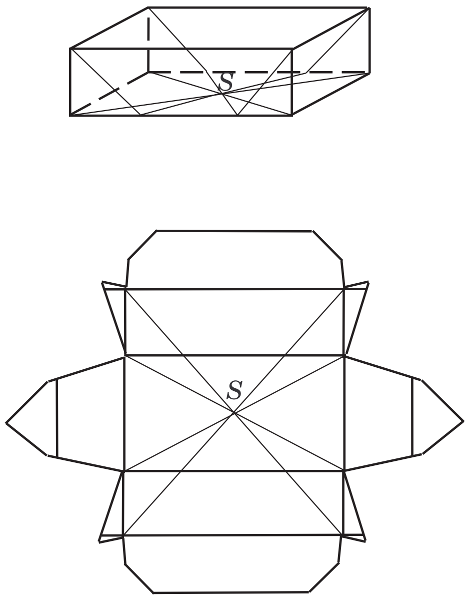


Fig. 7. Outward layout.

Here we give a new layout which is obtained by cutting the shortest paths from the source point to the vertices of the polyhedron.

To be formal, we define the planar layout of a set of faces $\mathcal{F}$ as the image of $\mathcal{F}$ on a plane through a set of transformations $\mathcal{T}$ which transforms each face in $\mathcal{F}$ to the plane; $\mathcal{T}$ preserves the shape and the connectivity of faces, *i.e.*, two faces which are on different sides of an edge before transformation must have their images on different sides of the image of the edge after transformation. We have the following theorem.

Theorem 4: *The surface of a polyhedron cut by the shortest paths to the vertices of the polyhedron can be laid out on a common plane.*

Proof: We cut the surface of the polyhedron along the shortest paths to each vertex (Fig. 8). If there are more than one shortest paths to a vertex, we cut an arbitrary one shortest path to that vertex. Some of the faces will be divided into pieces(call them regions) by the cutting. Draw a *dual* graph $\mathcal{D}$ of these regions. Take each region as a vertex of $\mathcal{D}$ (we take those faces which have not been cut by the shortest paths to the vertices of the polyhedron as a single region containing the whole face). Two vertices are connected by an edge if the corresponding regions share a common (undirected) edge of the polyhedron. The edges of $\mathcal{D}$ do not intersect the cut shortest paths to the vertices of the polyhedron. Graph $\mathcal{D}$ is acyclic. For, if there were a cycle in $\mathcal{D}$, this cycle would cut the surface of the polyhedron into two parts. There would then exist a vertex D of the polyhedron in

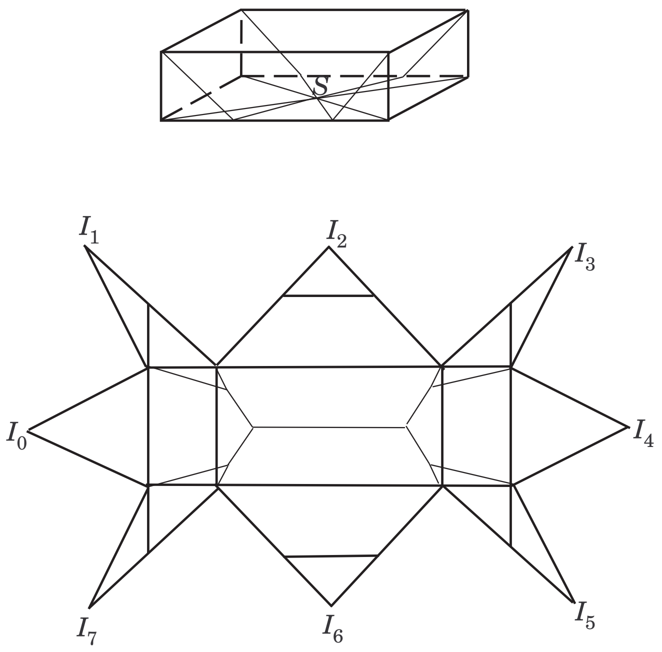


Fig. 8. Inward layout.

the part which does not contain the source point S. Thus the shortest path from S to D would intersect the cycle. A contradiction.

Starting from a region, unfold the regions which share a common edge with this region into a common plane $\mathcal{L}$. Since the dual graph $\mathcal{D}$ is acyclic, we will not come back to the same region no matter in what direction we unfold the regions. Therefore, we can unfold all the regions into a common plane. $\square$

The layout obtained is also a star-shaped polygon. The edges of the layout polygon are the shortest paths from the source to the vertices of the polyhedron. The vertices of the layout polygon are the images of the source. Such a star-shaped layout is shown in Fig. 8.

We shall call the layout due to Sharir and Schorr[23] the *outward layout*, for shortest paths emanate from the center, *i.e.* the source point, outwards toward the destinations, and we shall call the layout presented here the *inward layout*, where a shortest path from the source to a destination goes inwards toward the interior of the layout polygon.

In a layout (either an inward layout or an outward layout), the vertices of the polyhedron, except the source, can be arranged into a circular order according to the circular order of the shortest paths to the vertices. If we connect the adjacent vertices in the circular order by straight line segments, these line segments form a closed curve. For each line segment, we define its inside to be the side of the source image. Therefore this closed curve decomposes the surface of the polyhedron into two regions. The region containing the source is called the *arctic* while the other region is called the *antarctic*. This closed curve is called the *equator* of the polyhedron with respect to the given source point.

Layouts are important constructs in storing shortest path information. In particular, the inward layout enables us to compute the Voronoi diagram once for all faces.

We note that Agarwal *et al.*[1] made similar observations about the inward layout independently. Aronov and O'Rourke[2] recently showed that the inward layout for a convex polyhedron does not overlap. Before their results we treated the inward layout as if it were overlapped because this is inevitably the case for nonconvex polyhedrons. The nonoverlap property for convex polyhedrons is not essential in our scheme, but it helps simplify our algorithm in the case of convex polyhedrons.

Below we give an algorithm for computing the inward layout using the sequence tree as input. Note that we do not store the intersection between edges of the polyhedron and the shortest paths to the vertices of the polyhedron. Thus the inward layout can be represented by $O(n)$ line segments.

Algorithm 3:

```
I.  Traverse the sequence tree to obtain the
    circular order of vertices on the layout;
II. For every two consecutive vertices in
    the circular order do
        compute their shortest paths from $S$;
        cut the surface along these two paths;
        unfold the portion of the surface
            until the two paths co-plane on plane $\mathcal{L}$;
```

Theorem 5: *Algorithm 3 computes the inward layout of a polyhedron in $O(n^2)$ time and $O(n)$ space.*

Proof: Since shortest paths to vertices are cut and surface of the polyhedron is unfolded, we obtain the inward layout. A traversal of the tree, and the cutting and unfolding of the surface take $O(n^2)$ time and $O(n)$ space. □

4.2. Voronoi Diagram and Subdivision

After the layout is built, the source point S has $O(n)$ images on the layout. We now compute a Voronoi diagram with respect to these images. The layout is divided into regions by the Voronoi diagram. The points in the same region are closer to the corresponding image of S than to other images, and their shortest paths to S pass the same edge sequence.

Within the antarctic on the inward layout, the Voronoi diagram is a tree of $O(n)$ edges, and each leaf of the tree is a vertex of the polyhedron (Fig. 8.).[23] $O(n \log n)$ time and $O(n)$ space are sufficient for computing the Voronoi diagram.[22] We omit the details of constructing the Voronoi diagram here because in the next section we will outline the construction of the Voronoi diagram for the more complicated case of nonconvex polyhedron.

Theorem 6: *The subdivision of the surface of a polyhedron can be computed in $O(n^2)$ time.*

Proof: The subdivision of the surface of the polyhedron can be obtained by finding the intersection of the faces of the polyhedron with the ridge lines and the shortest paths to the vertices. Ridge lines, when laid out on the inward layout, are the edges of Voronoi diagram. Thus we can obtain the subdivision by picking edges of the Voronoi diagram and shortest paths to the vertices on the inward layout, and wrapping them on the surface of the polyhedron. These operations take $O(n^2)$ time. □

Since there are $O(n^2)$ regions in the subdivision, it requires $O(n^2)$ space for storing the subdivision. Note that our algorithm for computing the subdivision uses $O(n)$ space except for the storing of the regions in the subdivision. The subdivision can be used to answer queries of shortest path.[14,15,23]

Theorem 7: *After the subdivision of a polyhedron is built, the length of the shortest path from the source point to a query point can be determined in time $O(\log n)$ while the shortest path can be determined in time $O(\log n + k)$, k is the number of edges the shortest path passes through.*

Proof: Given a query point Q on face f of the polyhedron, by performing a point location on the subdivision of face f, we obtain the length of the shortest path to Q as in Ref. 14,15,23. The point location can be done in time $O(\log n)$ by known point location techniques.[8,18] The shortest path can be determined by tracing the regions back to the source point, which can be done in time $O(k + \log n)$, where k is the number of edges passed by the shortest path. □

Our recent work[6] shows that the shortest path information can be stored in

$O(n \log n / \log d)$ space to support the processing of a query in $O(d \log n / \log d)$ time, where $1 < d \leq n$ is an adjustable integer. This result enables us to cut the overall space requirement of our algorithm to $O(n \log n / \log d)$ while supporting the processing of a query in $O(d \log n / \log d)$ time.

5. Nonconvex Case

Complications under the nonconvex case are discussed in this section. Our treatment of nonconvex polyhedrons follows that of Mitchell *et al.*.[14]

Theorem 8: *The sequence tree for a nonconvex polyhedron can be built in* $O(n^2)$ *time and* $\Theta(n)$ *space.*

Proof: If a polyhedron is nonconvex, it is possible for a shortest path to pass through some vertices.[14] Suppose v is the last vertex on the shortest path from source S to a destination D. The shortest path from source S consists of the shortest path from S to v and the shortest path from v to D. We may view each vertex v as a *pseudo-source* with an initial distance equal to the length of the shortest path from S to v. S itself is a pseudo-source with 0 as its initial distance. We use two kinds of nodes in the sequence tree. A node is either a *vertex node* which is a pair (v, δ) or an *edge node* which is a quadruple $(e, I, Proj_e^I, \delta)$, where δ is the shortest distance known so far from S to a pseudo-source v and I is the image of v. The length of shortest path to a destination D through v is $\delta + |DI|$. An edge node can have at most two edge nodes and one vertex node as its children. A vertex node can have at most twice as many children as the number of edges incident with it. Half of them are edge nodes and half of them are vertex nodes. Accordingly, the principle "one angle one split" needs to be modified slightly for the nonconvex case. Only the edge nodes which occupy an angle can have three children. Therefore each angle in the polyhedron contributes at most two more leaves in the sequence tree. The total number of leaves in the sequence tree contributed by all angles of the polyhedron is $O(n)$. Vertex node $n_i = (v_i, \delta_i)$ can have at most $2m_i$ children, where m_i is the number of edges incident with v_i. Although a vertex can be occupied more than once, we need at most one vertex node for each vertex of the polyhedron in the sequence tree. Thus the total number of leaves in the sequence tree contributed by all vertex nodes is $O(n)$. Therefore the number of leaves at any time during an execution of Algorithm 2 is $O(n)$. The correctness of Theorem 8 follows. □

As in the convex case, the sequence tree gives the shortest paths from the source point S to all vertices of the polyhedron.

In the convex case a ridge point is defined as a point R to which there are more than one shortest paths from the source.[23] This definition is not suitable in the nonconvex case. Instead we define ridge points as the loci of the Voronoi diagram constructed on the inward layout of the polyhedron from the images of the pseudo-sources. Such a Voronoi diagram is constructed by first labeling the image of each pseudo-source I with the shortest distance from source S to I (*i.e.* $|SI|$), then points P with the property $|PI_1| + |SI_1| = |PI_2| + |SI_2| = |SP|$, where I_1, I_2 are two different images of pseudo-sources, are labeled as points on the Voronoi daigram.

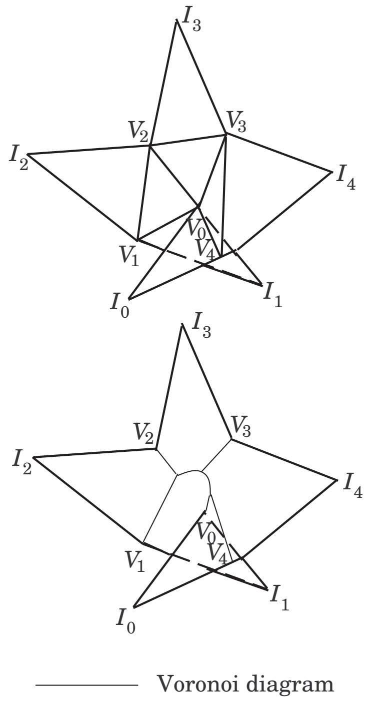


Fig. 9. Inward layout and Voronoi diagram for a nonconvex polyhedron.

The inward layout is a polygon which may overlap with itself (Fig. 9.). A path between two points on the inward layout is confined within the layout polygon. The shortest distance of two points on the layout is the minimum length of the paths between the two points. After unfolding, the pseudo-sources have a total of $O(n)$ source images on the layout because we cut $O(n)$ shortest paths to the vertices of the polyhedron.

Theorem 9: *The Voronoi diagram on the inward layout of a nonconvex polyhedron can be computed in $O(n \log n)$ time and $\Theta(n)$ space.*

Proof: The algorithm for computing the Voronoi diagram on the inward layout can be obtained by modifying the algorithm for traditional Voronoi diagrams.[22] Here we only give a sketch of the modified algorithm. A circular order of the source images on the inward layout can be obtained by traversing the boundary of the inward layout. We define a linear order by breaking the circular order somewhere. Suppose that A, the set of image sources, is divided into two subsets L and R, each containing $|A|/2$ image sources, such that every image source of L is to the "left" of those of R. Assume that we already have the Voronoi diagrams $\mathcal{V}(L)$ and $\mathcal{V}(R)$ of L and R, respectively. $\mathcal{V}(L)$ and $\mathcal{V}(R)$ can be merged in linear time $O(|A|)$ to form the Voronoi diagram $\mathcal{V}(A)$ for the entire set as in Ref. 22. Splitting the problem recursively will give an $O(n \log n)$ algorithm.

The merge is done similar to that in Ref. 22. We only note the differences here. The poly-hyperline to be constructed in the merging is to start from the edge of the layout polygon which connects the rightmost source image I_L in L and the leftmost source image I_R in R. When I_L and I_R have different initial distances, the actual poly-hyperline starts from either I_L or I_R, and a segment of the poly-hyperline may be a hyperbola. When the polyline is extended toward the interior of the layout polygon we may use the same procedure outlined in Ref. 22. The poly-hyperline may stop at an edge or a vertex of the layout polygon or it may be infinite during the construction of the Voronoi diagram. It never jumps from one level of the layout polygon to a different level by passing through an edge or a vertex of the inward layout. □

Thus the shortest path problem on nonconvex polyhedrons can be solved in the same time and space as we do in the convex case.

6. Conclusions

We have given an $O(n^2)$ algorithm to compute the subdivision of the surface of an arbitrary polyhedron such that the length of the shortest path from a given source point S to any destination point D on the surface may be determined merely by locating D in the subdivision. We expect that the non-Dijkstra approach could be applied to other versions of the shortest path problem in robot motion planning.

References

1. P. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon, " Star unfolding of a polytope with applications", *Proc. of the Scandanavian Workshop on Algorithm Theory.* LNCS 447, 1990.

2. B. Aronov, J. O'Rourke. "Nonoverlap of the star unfolding", in *Proc. 1991 ACM Symp. on Computational Geometry.*

3. J. Canny, "A new algebraic method for robot motion planning and real geometry", *Proc. 1987 IEEE FOCS*, pp. 39-48.

4. J. Canny, J. Reif, "New lower bound techniques for robot motion planning problems", *Proc. 1987, IEEE FOCS*, 49-60.

5. B. Chazelle, "Triangulating a simple polygon in linear time", *Discrete and Computational Geometry*, Vol. 6, 485-524(1991).

6. J. Chen and Y. Han, "Storing shortest paths for a polyhedron", *Proc. 1991 Int. Conf. on Computing and Information. Lecture Notes in Comput. Sci.*, 497, 169-180.

7. E. W. Dijkstra, "A note on two problems in connection with graphs", *Numer. Math.*, 1(1959), pp. 269-271.

8. D. G. Kirkpatrick, "Optimal search in planar subdivisions", *SIAM J. COMPUT.* 12(1983), pp.28-35.

9. D. T. Lee, "Proximity and reachability in the plane", Ph.D. thesis, Technical Report ACT-12, Coordinated Science Laboratory, Univ. of Illinois, Chicago, IL, Nov. 1978.

10. D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear boundaries", *Network*, 14(1984), pp. 393-410.

11. T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles", *Comm. ACM*, 22(1979), pp. 560-570.

12. J. S. B. Mitchell, "Shortest paths in the plane in the presence of obstacles", Manuscript, Dept. of Operations Research, Stanford Univ., Stanford, CA, 1984.

13. J. S. B. Mitchell, "Planning shortest paths", Ph.D. thesis, Dept. of Operation Research, Stanford, CA, August, 1986.

14. J. S. B. Mitchell, D. M. Mount and C. H. Papadimitriou, "The discrete geodesic problem", *SIAM J. COMPUT.* 16 (1987), pp.647-668.

15. D. M. Mount, "On finding shortest paths on convex polyhedra", Technical Report 1495, Department of Computer Science, University of Maryland, Baltimore, MD, 1985.

16. D. M. Mount, "The number of shortest paths on the surface of a polyhedron", Tec. Rep., Computer Sci. Dept., Univ. of Maryland, College Park, 1986.

17. J. O'Rourke, S. Suri and H. Booth, "Shortest path on polyhedral surfaces", Manuscript, The Johns Hopkins Univ., Baltimore, MD, 1984.

18. F. P. Preparata, "New approach to planar point location", *SIAM J. COMPUT.* 10(1981), pp.473-482.

19. J. Reif, J. A. Storer, "Shortest paths in Euclidean space with polyhedral obstacles", Tec. Rep. CS-85-121, Computer Science Dept., Brandeis Univ., Waltham, MA, April, 1985.

20. J. Reif, J. A. Storer, "3-Dimensional shortest paths in the presence of polyhedral obstacles", *Proc. 1988 Foundations of Computer Sci.*, pp. 85-92.

21. C. Schevon, J. O'Rourke, "The number of maximum edges sequences on a convex polytope", *Proc. Allerton Conference*, 1988.

22. M. Shamos and D. Hoey, "Closest-point problems", *Proc. 17th Annual IEEE FOCS*, (Oct. 1975) pp. 151-162.

23. M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces", *SIAM J. COM-*

PUT. 15(1986), pp. 193-215.