Shortest Paths on a Polyhedron [1]

Jindong Chen and *Yijie Han*

Department of Computer Science
University of Kentucky
Lexington, KY 40506

ABSTRACT
We present an algorithm for determining the shortest path between a source point and any destination point along the surface of a polyhedron (need not be convex). Our algorithm uses a new approach which deviates from the conventional "continuous Dijkstra" technique. It takes $O(n^2)$ time and $\Theta(n)$ space to determine the shortest path and to compute the inward layout which can be used to construct a structure for processing queries of shortest path from the source point to any destination point.

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 among them 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 [Le] [LW] [Mi1] [Mi2] [SS]. The known fastest algorithm by Reif and Storer [RS1] 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 [LP] [Mi2] [SS]. The known shortest path problem in the general 3-dimensional situation is NP−hard [CR], only exponential time algorithms are known [Ca] [RS2] [SS]. An important special case in the 3-dimensional situation is the shortest path problem along the surface of a single polyhedron. The known algorithms for shortest path problems on a single polyhedron build subdivision on the surface of the polyhedron [HCT] [MMP] [Mo1] [Mo2] [SO] [SS]. One version of the problem is to build a subdivision such that the shortest path between any pair of points on the surface can be reported quickly. This is called the all-pair shortest path problem, for which known algorithms can only report the shortest path between any pairs of points on the edges of the polyhedron. A recently reported algorithm due to Hwang *et al.* [HCT] has time complexity $O(n^6 \log n)$. After building the subdivision, their algorithm can report the shortest path between any pair of points on the edges of the polyhedron in $O(k + \log n)$ time where k is the number of edges crossed by the shortest path. The other version is the single source shortest path problem which 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. This version is the one we shall address in this paper.

The single source problem for a single *convex* polyhedron was first studied by Sharir and Schorr [SS]. 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

[1] Research partially supported by grant from University of Kentucky Center for Robotics and Manufacturing Systems.

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 single source shortest path algorithm for a convex polyhedron was given by Mount [Mol] 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 [OSB]. Subsequently, Mitchell, Mount and Papadimitriou gave an $O(n^2 \log n)$ algorithm [MMP].

All the previous algorithms for the problem can be regarded as preprocessing of the polyhedron which build subdivision on the polyhedron surface such that any point in the same region of the subdivision has the same shortest path edge sequence. Our algorithm accomplishes the same task.

The essence of a technique called "continuous Dijkstra"[MMP] is used in the previous designs [Mol][SS]. Mitchell *et al.* [MMP] were the first to formalize and generalize the technique, and gave the name of "continuous Dijkstra"[MMP]. In an application of this technique [MMP] [Mol] [SS], 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[Di]. To facilitate the extraction of the entity of the shortest distance, a priority queue of entities is maintained [MMP] [Mol] [SS].

In this paper, we present a new algorithm for the single source shortest path problem on the surface of a polyhedron (need not be convex). Our algorithm uses a rather different approach. It does not have the feature 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 to 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 [SS]. 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. Our non-Dijkstra approach to the problem is particularly interesting which may be considered as an alternative in the design of shortest path algorithms in computational geometry.

Our recent work [CH] extends the results presented here. Previously, Sharir and Schorr [SS] 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 [MMP], but no result better than $O(n^2)$ has been claimed. Using the inward layout of the polyhedron obtained by our algorithm presented here, we are able to store 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 [CH].

The rest of the paper is organized as follows. For a clear exposition, we restrict our discussion on a convex polyhedron only from section 2 through section 4. In section 2, some preliminary knowledge of shortest paths on the surface of a polyhedron is reviewed. In section 3, Lemma 1 is given, which makes it possible to build the sequence tree with $O(n)$ branches. Algorithm 2 is presented there which builds this tree. This algorithm establishes the fact that a shortest path from a source to a destination can be computed in $O(n^2)$ time and $O(n)$ space. In section 4, the details of computing the inward layout and building the subdivision are presented. In section 5 adaptation of the algorithm to the nonconvex case is discussed. Conclusions are given in section 6.

2. PRELIMINARY

Let S be the surface of a given polyhedron P, defined by 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. An angle θ_i denotes the inner angle of polygonal face f_j. 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 also becomes a vertex, as is done in [MMP] [Mol]. This process takes $O(n \log n)$ time, where n is the number of edges of the polyhedron[GJPT]. We should also mention that the complexity of the given polyhedron remains the same, *i.e.*, the number of edges is still $O(n)$.

A method used in studying shortest paths is *unfolding*. Assume $f_1, f_2, ..., f_i, ..., f_m$ is a sequence of faces, where f_i is adjacent to f_{i+1}, $e_i = f_i \cap f_{i+1}$, f_1 is the face containing the source point S (thus this sequence can also be denoted by the edge sequence, $e_1, e_2, ..., e_i, ..., e_{m-1}$). 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 $e_i$;
    $F := F \cup \{f_{i+1}\}$.
```

Upon finishing, all the faces are unfolded into a common plane $\mathcal{L}$. And the shortest path π passing through these faces should be unfolded into a straight line segment $\overline{\pi}$ [SS]. We call the edge sequence which is passed by a shortest path *shortest sequence*.

However, for some sequences, it is impossible to have any shortest paths passing through them (Figure 2.1). We can avoid these sequences by maintaining on each edge e_i image source I_{e_i}, the image of the source point S in the planar coordinate system of face f_i upon unfolding, and a *projection* of I (Figure 2.2). A *projection* of the source image I_{e_i} on edge e_i is a closed segment such that the shortest path to any point P in the segment through the sequence can be unfolded into a straight line segment (such a locally optimal path is called a *geodesic path* [MMP] [SS]), namely, we can connect I_{e_i} and P with a straight line segment which crosses only the edges $\overline{e_1}$, $\overline{e_2}$, ...,$\overline{e_{i-1}}$, where $\overline{e_j}$ is the image of e_j. Let us denote the projection of I on e as $Proj_e^I$. It is easy to compute the image source and the projection of e_i using those of e_{i-1}. The projection on e_i is the intersection of e_i and the *shadow* of $Proj_{e_{i-1}}^{I_{e_{i-1}}}$ on face f_i (Figure 2.3). We unfold f_{i+1} around e_i only if the projection on e_i is not empty.

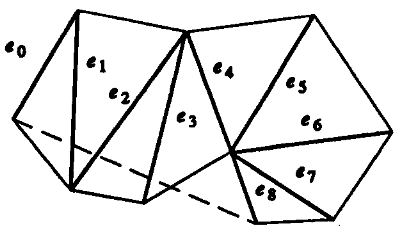


Figure 2.1

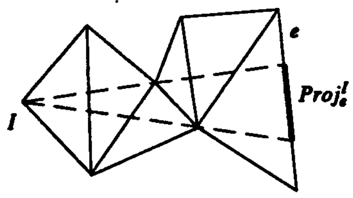


Figure 2.2

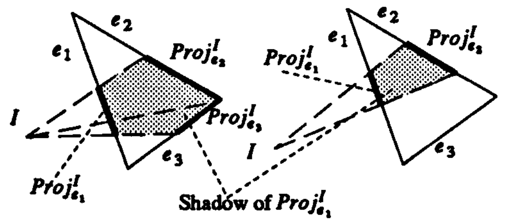


Figure 2.3

The following properties for shortest paths on a convex polyhedron are also known [Mol] [SS].
(1) A shortest path can not pass more than n faces, where n is the total number of faces of the polyhedron.
(2) Two shortest paths can not intersect each other unless 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 arrange into a *circular order* according to the angles at which they emanating from the source.

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 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.

3. SEQUENCE TREE

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

We decompose an edge into two oriented edges as in [Mol], such that the image source can only be on the left side of the oriented edge (Figure 3.1). The term edge we used in the following text of this section is meant to be an oriented edge if not claimed. 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 is a shadowed face of one of its edges. In the following algorithm, a sequence tree is built. 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 and e is the edge of n, n's shadow is the shadow of $Proj_e^I$.

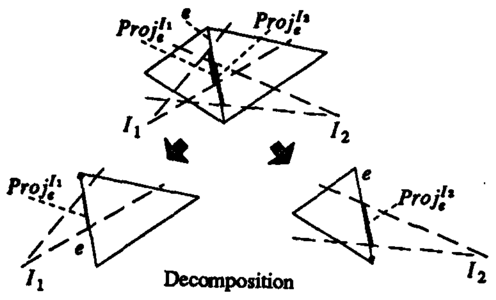


Figure 3.1

Unfolding around e is defined as rotating I around e until it is co-planar with the face on the other side of e, lying on different sides of e.

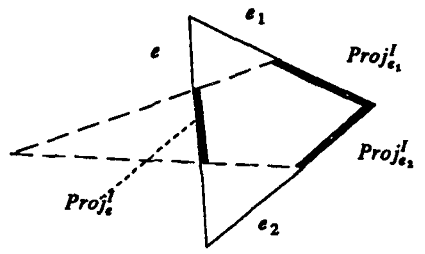
Algorithm 1

```
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
            unfold $I$ 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
                $(e, I, Proj_e^{\overline{I}})$'s child;
```

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 down 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.

However, the size of the tree built by Algorithm 1 could be exponential. In order to obtain an efficient algorithm, we should cut the size of the tree substantially.

The reason the tree could have exponential number of nodes is that all the nodes whose shadows cover the opposite angle has two children in the tree (Figure 3.2). However, Lemma 1 tells us that at most one node among those whose shadows cover the angle can have two children.



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

Figure 3.2

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

Proof: Unfold I_{n_1}, I_{n_2}, the images of n_1 and n_2 around CB to be coplanar with ΔABC (Figure 3.3).

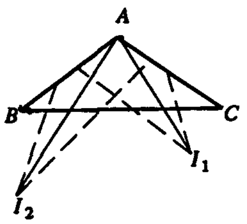


Figure 3.3

$\overline{I_{n_1}}, \overline{I_{n_2}}$, are the 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 to A are paths passing through the edge sequence represented by n_1, therefore only n_1 can fork to two children while n_2 has only one child. We say that n_1 *occupies* $\angle CAB$ over n_2. By the same reason, if $|\overline{I_{n_1}}A| > |\overline{I_{n_2}}A|$ then n_2 can have two children while n_1 has only one child. Finally, if $|\overline{I_{n_1}}A| = |\overline{I_{n_2}}A|$, n_1 *ties* n_2, then both n_1 and n_2 have at most one child. □

With Lemma 1, upon an angle $\angle\theta$ we only let the node n which is the closest to the angle fork into two children. We say n occupies $\angle\theta$. 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 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 which is impossible to define a shortest sequence (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 $n = (e, I, Proj_e^I)$
            of the $i$th level do
            unfold $I$ 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 $n'$,
                /* which currently occupies $\angle CAB$,*/
                then
(a)                 clip off one of $n'$'s two children
                        that is impossible to define a
                        shortest sequence and delete the
                        subtree rooted at this child;
                    insert $(AB, \overline{I}, Proj_{AB}^{\overline{I}})$,
                        $(CA, \overline{I}, Proj_{CA}^{\overline{I}})$ as $n$'s children;
                    mark $\angle CAB$ as occupied by $n$;
                else /* $n$ can not occupy the angle */
                    calculate $Proj_{e'}^{\overline{I}}$,
                    /* $e'$ is either $CA$ or $AB$ other than*/
                    /* the one which has been found     */
                    /* impossible to define a shortest  */
                    /* sequence;                        */
                    insert $(e', \overline{I}, Proj_{e'}^{\overline{I}})$;
                        as the sole child of $n$;
            else /* $n$'s shadow does not cover $A$ */
                $e'$:=either $AB$ or $CA$ which is nonempty;
                insert $(e', \overline{I}, Proj_{e'}^{\overline{I}})$; as the sole
                    child of $n$, accordingly;
```

Theorem 1: *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 $O(n^2)$. □

The sequence tree output by Algorithm 2 has $O(n)$ leaves. All the shortest sequences are contained in this tree, because only those found impossible to define a shortest sequence are deleted.

It will be clear later on that the essential piece of information generated by Algorithm 2 is the shortest paths to the vertices of the polyhedron. In order to record this information, we introduce vertex node into the sequence tree. A vertex node is a triple (e, I, V), where V is a vertex, e is an angle and I is an image source. Let a be a current node on edge CB in the sequence tree and 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 child. 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 node. We leave it to the reader to show that the shortest path to a vertex can be computed in $O(k)$ time using the modified sequence tree, where k is the number of edges the shortest path passes through.

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 arrange 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 1:

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 formed by examining the nodes of the form (e, I, V) on 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 is used to be done [MMP] [Mol] [SS] 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 [SS] 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 the Voronoi vertices on the ridge lines. Such a star shaped layout is shown in Figure 4.1.

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

Theorem 2: *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 (Figure 4.2). 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 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 is a cycle in $\mathcal{D}$, this cycle cuts the surface of the polyhedron into two parts. There exists a vertex D of the polyhedron in the part which does not contain the source point S. Thus the shortest path from S to D intersects 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. □

We have not been able to prove or disprove the proposition that the layout obtained here do not overlap. As will be seen, this proposition is not essential to our scheme.

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 Figure 4.2.

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

In a layout the vertices of the polyhedron except the source can be arranged into a circular order. If we connect the adjacent vertices in the circular order by a straight line segment. These line segments form a closed curve. 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.

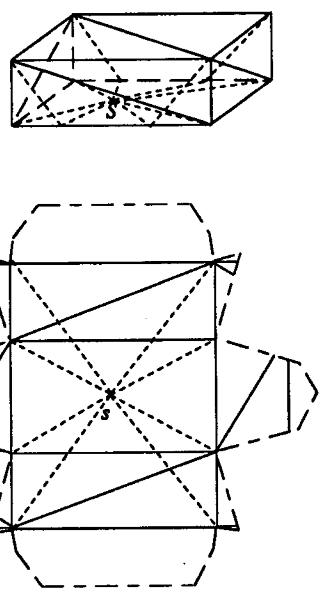


Figure 4.1

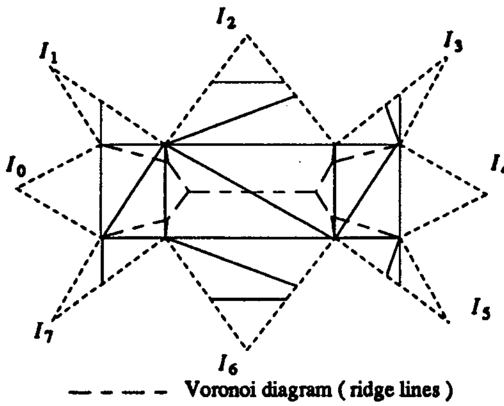


Figure 4.2

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.

After the submission of our paper, we learned that

Agarwal *et al.* [AAOS] made similar observations about the inward layout.

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 3: *The inward layout of a polyhedron can be computed in $O(n^2)$ 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 sequence.

It can be shown that, 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 (Figure 4.2). $O(n \log n)$ time and $O(n)$ space is sufficient for computing the Voronoi diagram [SH]. In the next section we will outline the construction of the Voronoi diagram for the more complicated case of nonconvex polyhedron.

The subdivision of the surface of the polyhedron can be obtained by finding the intersection of the faces of 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. We have

Theorem 4: *The subdivision of the surface of a polyhedron can be computed in $O(n^2)$ 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 uses $O(n)$ space except the storing of the regions in the subdivision.

The subdivision can be used to answer queries of shortest path [MMP] [Mol] [SS]. 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 [MMP] [Mol] [SS]. The point location can be done in time $O(\log n)$ by known point location techniques [Ki] [Pr]. 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.

Theorem 5: *The subdivision of a polyhedron can be built in time $O(n^2)$, after which 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.*

Our recent work [CH] 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, path 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 is discussed in this section.

If a polyhedron is nonconvex, it is possible for a shortest path to pass some vertices [MMP]. Suppose V is the last vertex on the shortest path from source S to 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. A sequence subtree rooted at V can grow since there are shortest paths pass V. We use two kinds of node in a sequence tree. A node is either a *vertex node* which is a pair $n = (V, \delta)$ or an *edge node* which is a quadruple $n = (e, I, Proj_e^I, \delta)$, δ is the shortest distance known so far from S to a pseudo-source V, 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, while a vertex node can have children with number at most twice 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. However, the number of leaves at any time during execution of Algorithm 2 is still $O(n)$. Thus $O(n^2)$ time and $O(n)$ space is sufficient for building a sequence tree for the polyhedron.

In the convex case a ridge is defined as a point R to which there are more than one shortest paths from the source [SS]. This definition is not suitable in the nonconvex case. We define the ridge paths as loci of the Voronoi diagram constructed on the inward layout of the polyhedron from the images of the source and pseudo-sources.

The inward layout is a polygon which may overlap with itself. This happens when there are some *concave* vertices. A *concave* vertex is a vertex the sum of the angles formed around it is larger than 2π. (Figure 5.1) A path for two points on the inward layout is restrained within the layout polygon. Shortest distance of two points on the layout is the minimum length of the paths between the two points. After unfolding, the source as well as the pseudo-sources have a total of $O(n)$ images on the layout. Let us call them image source without discrimination.

The algorithm for computing the Voronoi diagram on the inward layout can be obtained by modifying the algorithm for traditional Voronoi diagram [SH] [Sh]. Here we only give a sketch of the modified algorithm. A circular order of the image sources 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, S, the set of image sources is divided into two subset L and R, each containing $|S|/2$ 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 $O(|S|)$ to form the Voronoi diagram $\mathcal{V}(S)$ for the entire set as in [SH]. Splitting the problem recursively will give an $O(n \log n)$ algorithm.

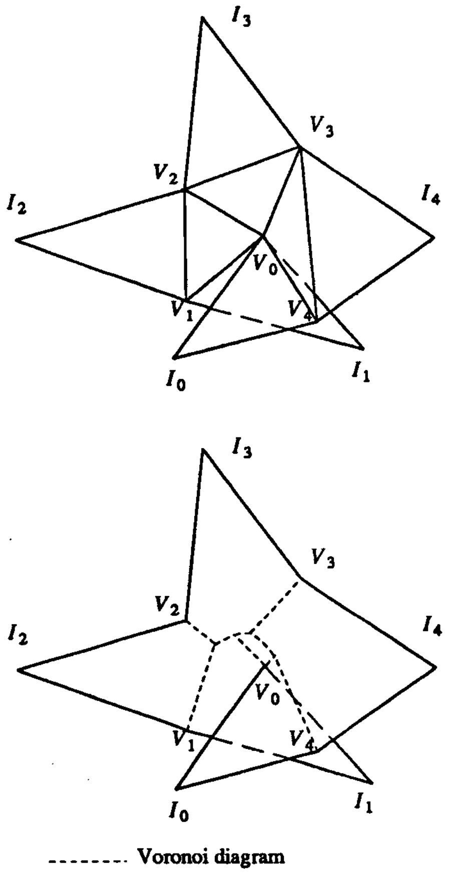


Figure 5.1

The merge is done similar to that in [SH]. 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 image source I_L in L and the leftmost image source I_R in R. Since I_L and I_R have different initial distance, the actual poly-hyperline always 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 outline in [SH]. The poly-hyperline is to stop at an edge of 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.

Having overcome all the complications, the shortest path problem on nonconvex polyhedron can be solved in the same time and space as we do in 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 T on the surface may be determined merely by locating T in the subdivision. We expect that the non-Dijkstra approach could be applied to other versions of the shortest path problem.

REFERENCES

[AAOS] P. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon, *Star unfolding of a polytope with applications (extented abstract)* Tec. Rep. 90.2.13, February 1990.

[Ca] J. Canny, *A New algebraic method for robot motion Planning and Real Geometry*, Proc. 1987 IEEE FOCS, 39-48.

[CH] J. Chen and Y. Han, *Shortest paths on a polyhedron, Part II: storing shortest paths*, TR No. 161-90, Computer Sci. Dept., University of Kentucky, Lexington, Kentucky, February, 1990.

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

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

[GJPT] M. R. Garey, D. S. Johnson, F. P. Preparata and R. E. Tarjan, *Triangulating a simple polygon*, Inform. Process. Lett., 7(1978), pp. 175-179.

[HCT] Yie-Huei Hwang, Ruei-Chuan Chang and Hung-Yi Tu, *Finding all shortest path edge sequences on a convex polyhedron*, LNCS 382, Algorithms and Data Structures Workshop, Ottawa, Canada. 251-266 (Aug. 1989).

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

[Mi1] J. S. B. Mitchell, *Shortest paths in the plane in the presence of obstacles*, Manuscript, Dept. of Operations Research, Stanford Univ., Stanford, CA, 1984.

[Mi2] J. S. B. Mitchell, *Planning shortest paths*, Ph.D. thesis, Dept. of Operation Research, Stanford, CA, August, 1986.

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

[Mo1] D. M. Mount, *On finding shortest paths on convex polyhedra*, Technical Report 1495, Department of Computer Science, University of Maryland, Baltimore, MD, 1985.

[Mo2] 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.

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

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

[RS1] 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.

[RS2] J. Reif, J. A. Storer, *3-Dimensional shortest paths in the presence of polyhedral obstacles*, Proc. 1988, Foundations of Computer Sci. 85-92.

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

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

[SS] M. Sharir and A. Schorr, *On shortest paths in polyhedral spaces*, SIAM J. COMPUT. 15(1986), pp. 193-215.