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

Shortest Paths with Single-Point Visibility Constraints

Ramtin Khosravi
Computer Engineering Department, Sharif University of Technology
P.O. Box: 11365-9415, Tehran, Iran
ramtin@mehr.sharif.edu

Mohammad Ghodsi
Computer Engineering Department, Sharif University of Technology
P.O. Box: 11365-9415, Tehran, Iran
ghodsi@sharif.edu

This paper studies the problem of finding the shortest path between two points in presence of single-point visibility constraints. In this type of constraints, there should be at least one point on the output path from which a fixed viewpoint is visible. The problem is studied in various domains including simple polygons, polygonal domains, polyhedral surfaces. The method is based on partitioning the boundary of the visibility region to a number of intervals according to shortest path structure of their points to both source and destination. The result for the two dimensional domains is worst-case optimal.

Keywords: Computational Geometry; Shortest Paths; Visibility Constraints; Shortest Path Maps.

1. Introduction

Finding the shortest path between two points is a basic problem in computational geometry and has many applications in different areas such as motion planning and navigation. The problem has been studied over various geometric domains such as simple polygons [1], polygonal domains [2,3], polyhedral surfaces [4,5,6]. Also, several variations exist depending on the metric used for computing distances, and different constraints applied to the solution path. Examples of such restrictions are curvature constraints [7] or altitude constraints [8]. The visibility constraints are less studied so far. In this type of constraints, the path is required to satisfy some visibility properties, e.g. entire or part of the path should be visible from a given viewpoint. Applications for this constraint are mainly in communication systems where direct visibility is needed or guarding problems. An example of a motion planning problem combined with visibility constraints can be found in [9] in which the problem of locating a continuously-moving target using a group of guards moving inside a simple polygon is studied.

Single-point visibility constraint requires the path to include at least one point from which a given viewpoint is visible. In this paper, we study the problem of finding shortest paths with single-point visibility constraint in several domains: simple polygons, polygonal domains, and polyhedral surfaces. The algorithms proposed for the first two cases are worst-case optimal and have $O(n)$ and $O(n \log n)$ time bounds respectively. The case of polyhedral surfaces is more complex and our algorithm for this domain runs in $O(n^2 \log n)$ and $O(n^3 \log n)$ time for convex and non-convex cases respectively. To the best of our knowledge, no other result has been available on this problem.

If we have to visit multiple viewpoints during motion along the path, we face a related problem which is *TSP with Neighborhoods* in which multiple polygonal regions, called neighborhoods, are given and the goal is to find a tour that visits each neighborhood. The problem is NP-hard [10], and several approximation algorithms have been presented for different cases [11,12,13].

The approaches used in this paper to solve the problem for different domains have similar structure, so we formulate it in a generic form in section 2, and then discuss issues specific to two dimensional domains (i.e., simple polygons and polygonal domains) in section 3, and the case of polyhedral surfaces is studied in section 4.

2. Definitions and the General Approach

Here we consider the general approach for solving the problem of finding the shortest path between two points with the constraint that at least one point of the path is visible from a given viewpoint.

Let $\mathcal{P}$ be the geometric domain of the problem under consideration. We consider $\mathcal{P}$ as a set of points, and $\mathcal{V}_p$ as the *visibility region* of the viewpoint $p \in \mathcal{P}$. In all domains considered in this paper, the boundary of $\mathcal{V}_p$ consists of a number of line segments. The set $\mathcal{P} - \mathcal{V}_p$ consists of a number of connected sets of points, which we call *invisible regions* of the domain. Our goal is to compute the shortest path between given source point s and destination point t such that the intersection of the path and $\mathcal{V}_p$ is non-empty. We call such a path *p-visible.*

Let q be the first intersection point between $\mathcal{V}_p$ and the shortest p-visible path between s and t when walking along the path from s and t. Obviously, q lies somewhere on the boundary of $\mathcal{V}_p$, and the subpaths from s to q and from q to t are locally optimal. So, our problem reduces to finding the point q defined above.

The main idea of the algorithm is to partition the boundary of $\mathcal{V}_p$ into a set of intervals such that for each interval I, we can easily find the point $q(I) \in I$ whose total shortest distance to s and t is minimum among all points on I. Then q is the one with the minimum total shortest distance among all such points.

Partitioning the boundary of $\mathcal{V}_p$ is based on the combinatorial structure of the shortest paths between s (or t) and the points on the boundary. An interval in this partition is a connected set of points such that the shortest paths between all points in this interval and s have the same combinatorial structure, while the same is true

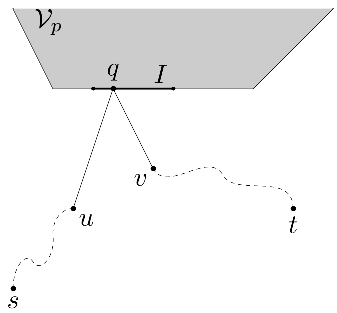

Fig. 1. In two dimensional domains, a linearity interval I (shown with heavy solid line) is a connected subset of the boundary of $\mathcal{V}_p$ such that the shortest path from s (resp. t) to its points passes u (resp. v) as the last vertex. $q \in I$ is the point with minimum total shortest path distance to s and t.

for the shortest paths to t. In the domains we will consider, these intervals have the property that the shortest distance function from their points to both s and t is linear, so we call them *linearity intervals* or simply *L-intervals* with respect to the pair (s, t), denoted by $L_{s,t}$ (Fig. 1).

To find $L_{s,t}$ we find the linearity intervals with respect to s (L_s) and t (L_t) separately, then merge the endpoints of the intervals in the two sets to obtain the endpoints of the intervals in $L_{s,t}$. Intervals in L_s (resp. L_t) can be found by intersecting the boundary of $\mathcal{V}_p$ with the edges of the *shortest path map* of s (resp. t), although computing the entire set of intervals may not be necessary.

Based on the preceding discussion, we sketch the generic algorithm for computing the shortest p-visible path between s and t as the following:

(1) Computer the boundary of $\mathcal{V}_p$ (or the relevant portion of it).
(2) Compute the set of L-intervals with respect to s and t on the boundary of $\mathcal{V}_p$ (or the relevant subset of it) and call it $L_{s,t}$.
(3) For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t.
(4) Let q be the point with the minimum total distance from s and t among $\{q(I) : I \in L_{s,t}\}$.
(5) The shortest p-visible path between s and t is the shortest path between s and q followed by the shortest path between q and t.

The first two steps in the above algorithm depends on the specific geometric domain of the problem, which we consider in succeeding sections.

3. Two Dimensional Cases

In two dimensions, we consider two geometric domains for the problem: simple polygons and polygonal domains. For the polygonal domain case, we assume that the domain is bounded by a given simple polygon called the boundary and there are a number of (simple) polygonal obstacles inside it. By *free space* we mean the set of points inside or on the boundary of the simple polygon minus the interior of the obstacles. The *shortest path map* of the source point x, denoted by $SPM(x)$, is a linear-size subdivision of the free space into regions (*cells*) such that the shortest path to the points in one cell has the same combinatorial structure, i.e., has the same sequence of obstacle vertices along the path. The last obstacle vertex along the path to the points in a cell is called the *root* of that cell. Each cell is starshaped with respect to its root, which lies on the boundary of the cell. The boundaries of cells consist of portions of the obstacle edges, extension segments (extensions of visibility graph edges incident on the root), and bisector curves. The bisector curves are, in general, hyperbolic arcs that are the locus of points that have shortest distance from two roots.

For a general polygonal domain, the shortest path map can be computed using the algorithm of Hershberger and Suri [3] in worst-case optimal time $O(n \log n)$ using $O(n \log n)$ space. The subdivision can be used to answer single-source shortest path queries using classic point-location algorithms in logarithmic time.

For simple polygons, many shortest path problems have linear algorithms due on the fact that there is exactly one "taut-string" between any two points in a simple polygon that can be found using dual of a triangulation for the polygon, which is a tree. In the method presented by Guibas et al. [1], one can construct the shortest path map of a given source point using a DFS traversal of the mentioned tree. The method maintains a funnel-like structure during this traversal to construct the shortest path map. We will consider how to use this structure to find the L-intervals on the relevant portion of the visibility polygon.

3.1. *Simple Polygons*

In this case, $\mathcal{P}$ is supposed to be a simple polygon with n edges. The visibility region of a point p in a simple polygon is also a simple polygon. Edges of this polygon are extensions of segments connecting p to vertices of $\mathcal{P}$ visible from p. Hence, each edge of $\mathcal{V}_p$ decomposes $\mathcal{P}$ into two parts, one contains $\mathcal{V}_p$ and the other is completely invisible from p. We name the latter an *invisible region* of $\mathcal{P}$ with respect to the viewpoint p.

If either s or t lies inside or on the boundary of the visibility polygon of p, the Euclidean shortest path between s and t is p-visible. The same is true if s and t lie in two different invisible regions with respect to p. This is due to the fact that the only way for the path to exit from an invisible region is to cross a segment common to $\mathcal{V}_p$, since other edges of the region are part of the boundary of $\mathcal{P}$. So we can assume that the points s and t are in one invisible region with respect to p. We call this invisible region $\bar{V}$. Note that testing above conditions can be done in $O(n)$ total time.

Based on the above assumption, we can restrict the computations to the relevant

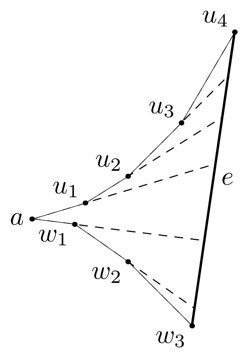

Fig. 2. L-intervals on the edge e obtained by running shortest path algorithm in the polygon $\bar{V}$.

portion of the boundary of $\mathcal{V}_p$ which is a single edge e common to $\mathcal{V}_p$ and $\bar{V}$. It can be easily verified that in this case, the shortest p-visible path does not enter $\mathcal{V}_p$ and just *touches* $\mathcal{V}_p$ and returns. The reason is obvious, since we can take a shortcut along e between the point that the path enters $\mathcal{V}_p$ and the point that it exits from. So, our problem is to find L-intervals on the edge e.

Since the shortest path does not exit $\bar{V}$, we can find the L-intervals on e easily by using the shortest path algorithm of [1] for constructing the shortest path map of a simple polygon. If we consider $\bar{V}$ as the input polygon to the mentioned algorithm with s as the source point, we first find a triangulation of $\bar{V}$ and then its dual which is a tree rooted at the triangle containing s. During the DFS traversal, we finally visit the boundary edge e (since e belongs to the boundary of $\bar{V}$, it is an edge of the triangulation). Assuming the funnel-like structure maintained during the traversal has the form $[u_l, u_{l-1}, \ldots, u_1, a, w1, \ldots, w_k]$ at that time with $a = u_0 = w_0$ as its cusp (thus $e = u_l w_k$), the rays emanating from u_i (resp. from w_i) and passing through u_{i+1} (resp. through w_{i+1}) partition e into $k + l - 1$ intervals (Fig. 2). Each interval has the property that the last polygon vertex on the shortest paths from s to all of its points is the same. These intervals are L-intervals on e with respect to s.

Computing the set $L_{s,t}$ for this case is easy now. We have to compute L_s and L_t using the method mentioned above, and a merging process is required to obtain $L_{s,t}$ from the computed intervals. Assuming the shortest Euclidean distances from s and t to all vertices of $\mathcal{P}$ have already been computed, we have the following lemma:

Lemma 1. *Let I be an L-interval with respect to the pair (s, t). One can find the point $q(I)$ on I with minimum total distance to s and t in constant time.*

Proof. I has the property that the last vertices of the shortest paths from s and t to an arbitrary point on I are the same for all points of I. Let u (resp. v) be the last vertex on shortest paths form s (resp. t) to the points of I. To find $q(I)$, we can reflect v about the line supporting I and connect the reflected point to u. If the segment obtained in this way intersects I, the intersection point will be $q(I)$. Otherwise, $q(I)$ will be one of the endpoints of I depending on which side of I lies the intersection point between the segment and the line supporting I. □

Based on the above discussions, the following steps are taken to compute the shortest point-visible path in a simple polygon:

Algorithm. *Computing Shortest Point-Visible Paths in Simple Polygons*

(1) If p is visible from either s or t, report the shortest Euclidean path between s and t.
(2) Compute the visibility polygon of p ($\mathcal{V}_p$).
(3) Compute the invisible regions in which s and t lie. (May be two different regions).
(4) If the invisible regions of s and t are different, report the shortest Euclidean path between s and t, otherwise, name the invisible region in which both s and t lie, $\bar{V}$, and the segment common to $\mathcal{V}_p$ and $\bar{V}$, e.
(5) Run the shortest path algorithm in the simple polygon $\bar{V}$ twice, assuming s and t as the source point each time. As the result of this step, the following information is generated:
    (a) Shortest Euclidean distances from both s and t to every vertex of $\mathcal{P}$, and
    (b) L-intervals with respect to both s and t on the edge e (L_s and L_t respectively).
(6) Compute L-intervals with respect to the pair (s, t), named $L_{s,t}$, by merging endpoints in L_s and L_t.
(7) For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t. Let q be the point among all $q(I)$ that has the minimum total distance.
(8) Report the shortest path from s to q appended by the shortest path from q to t.

The following theorem summarizes the result for the case of simple polygons and provides a time bound on the algorithm.

Theorem 1. *Given a pair of points s and t and a viewpoint p inside a simple polygon, the shortest p-visible path from s to t inside the polygon can be found in $O(n)$ time and $O(n)$ space.*

Proof. The check in step 1 of the algorithm above can be easily done in linear time. The visibility polygon computation in step 2 can be done using the linear time algorithm of Lee [14,15]. To compute the invisible region in which s (resp. t) lies (step 3), we can traverse the boundary of the invisible region, starting from the intersection of the ps (resp. pt) directed half-line and the boundary of $\mathcal{P}$. This can

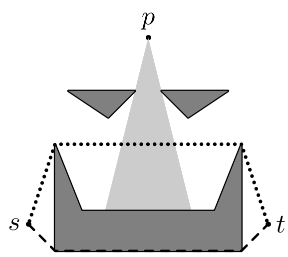

Fig. 3. Part of a polygonal domain with obstacles shown as dark shaded polygons. The shortest p-visible path between s and t (shown with dots) crosses part of $\mathcal{V}_p$ (light-shaded) while s and t are in one invisible region and the Euclidean shortest path between them (dashed line) does not cross $\mathcal{V}_p$.

be done in linear time assuming we have the vertices of the polygons $\mathcal{P}$ in order. Step 5 is calling the shortest path algorithm of Guibas et al. [1] twice and can be done in $O(n)$ time. Since the funnel structure is stored in a *finger search tree* [16] which is based on the B-tree data structure, we can obtain the sorted list of intervals in L_s and L_t both in $O(n)$ time. Therefore, $L_{s,t}$ can be computed in linear time (Step 6). Finally, the minimum point q can be computed in $O(n)$ time according to lemma 1. Each step of the algorithm uses at most $O(n)$ space, hence the overall algorithm needs linear time and space. □

3.2. *Polygonal Domains*

In this case, $\mathcal{P}$ is supposed to be a polygonal domain (i.e. a polygon with holes) with total number of n edges. The problem is to find the shortest obstacle-avoiding path between two given points s and t such that there is at least one point on the path from which a given viewpoint p is visible. As stated earlier, the main challenge in a particular domain is to find the linearity intervals efficiently. Taking the same approach as the case of simple polygon, we must find the invisible region in which s and t lie ($\bar{V}$), and run a shortest path algorithm (such as [3]) to construct the shortest path map of $\bar{V}$ and take the intervals made on the boundary between $\bar{V}$ and $\mathcal{V}_p$ as the linear intervals corresponding to the source of the map. However, unlike the case of simple polygons, the boundary between $\bar{V}$ and $\mathcal{V}_p$ may consists of more than one edge. So the shortest p-visible path between two points s and t may enter $\mathcal{V}_p$ from one edge and exit from another edge, while s and t both lie in one invisible region $\bar{V}$ and the shortest path between them lies completely inside $\bar{V}$ (Fig. 3).

In general, if we intersect the boundary of the visibility region of a viewpoint with the edges of the shortest path map of a source point, the boundary of the visibility polygon is partitioned into $O(n^2)$ segments, hence the total number of $L_{s,t}$-intervals on the boundary of $\mathcal{V}_p$ will be $O(n^2)$. Computation of these intervals, when done using known algorithms for finding segment intersections such as [17] will take $O(n^2 \log n)$ time. To reduce this to subquadratic time order, we define the notion of *visibility gates* as follows.

Consider the set Π of shortest p-visible paths from s to all points in the free space. For each such path π, let q_π be the first p-visible point when walking along π from s to the destination (i.e., q_π is the point whose shortest distance to s is minimum among all p-visible points on π). The set $\{q_\pi | \pi \in \Pi\}$ forms a set of segments on the boundary of $\mathcal{V}_p$ such that every shortest p-visible path starting from s, has a point in common with one of these segments. We call such a segment, a *visibility gate* of $\mathcal{V}_p$ with respect to s. Note that in a p-visible path like π, the subpath from s to q_π is optimal in the Euclidean shortest path metric, and the same is true for the subpath from q_π to the destination point. The next two lemmas bounds the number of visibility gates and the time and space required to compute them.

Lemma 2. *The number of visibility gates with respect to a source point s is $O(n)$.*

Proof. Consider a cell f of the shortest path map of the source point s with root r. If we connect r to vertices of $\mathcal{V}_p$ lying in f and extend the segment to touch the boundary of f, we obtain a number of slabs all having the root r in common. Each slab is intersected with a number of $\mathcal{V}_p$ edges (possibly zero), but contains no vertex from it in its inside (Figure 4). We claim that in each slab, only one intersecting segment, the closest to r, is part of a visibility gate. To see this, consider a point $x \in f \cap \mathcal{V}_p$ and suppose it belongs to a visibility gate. Since x belongs to a visibility gate, there must be a path π such that x is the first p-visible point on it when walking from s to the destination. As noted earlier, the subpath of π from s to x is the shortest Euclidean path between the two points, so it has r as its last vertex and the segment rx as its last edge. rx intersects the first segment crossing the slab, so x is not the first p-visible point along the path, which is a contradiction. So, the number of gates is bounded by total number of slabs in all cells of the shortest path map. Since each $\mathcal{V}_p$ vertex adds at most one slab to the cell containing it, $\mathcal{V}_p$ has $O(n)$ vertices, and shortest path map has $O(n)$ cells, this bound is $O(n)$. □

Lemma 3. *Visibility gates of s can be computed in $O(n \log n)$ time and space.*

Proof. To find the visibility gates of s, we consider the interior of $\mathcal{V}_p$ as an obstacle and run the standard shortest path algorithm with source s to generate the shortest path map of $\mathcal{P} - \mathcal{V}_p$ (namely $SPM_{\bar{\mathcal{V}}_p}$). As a result of generating this map, the boundary of $\mathcal{V}_p$ is partitioned into a number of segments, each on the boundary of one $SPM_{\bar{\mathcal{V}}_p}$ cell. Some of these segments belong to cells whose roots are vertices of $\mathcal{V}_p$ and others belongs to cells whose roots are vertices of $\mathcal{P}$. The first type of segments should be ignored since they are generated by the assumption that the interior of $\mathcal{V}_p$ belongs to obstacles. The second type of segments are visibility gates of s. This is true due the fact that the shortest path from s to a point on a visibility gate does not enter $\mathcal{V}_p$. □

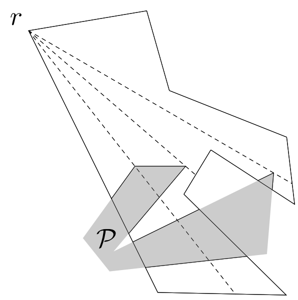

Fig. 4. Proof of Lemma 2.

The notion of visibility gates helps in reducing the number of L-intervals which are to be considered to compute the desired path.

Lemma 4. *To compute the shortest p-visible path between s and t, it is sufficient to include L-intervals on the visibility gates of s in the set $L_{s,t}$. Furthermore, there are $O(n)$ intervals on these gates that should be considered.*

Proof. Suppose π be the shortest p-visible path between s and t. If we consider q_π as the first p-visible point on π when walking from s along π, q_π belongs to a visibility gate of s. Since the subpaths from s to q_π and from q_π to t are shortest paths, it is sufficient to partition all visibility gates of s according to the combinatorial structure of the shortest paths to t. From the definition of visibility gates, it is obvious that any shortest p-visible path between s and t crosses (or touches) exactly one visibility gate of s. So if we consider the shortest Euclidean paths from t to points on the visibility gates of s, they have no point in common with gates of s other than the endpoint. Hence, we can compute the relevant L-intervals by considering visibility gates of s as obstacles of zero thickness, and using the Euclidean shortest path algorithm. Intervals generated on the gates associated with a vertex of $\mathcal{P}$ are L-intervals that are to be considered when computing the desired path. Similar to the computation of gates, we ignore those intervals with roots other than $\mathcal{P}$ vertices. Since there are $O(n)$ gates, the map generated by the shortest path algorithm will have linear size, hence the generated L-intervals is also linear. $\square$

Based on the notion of visibility gates, we propose the algorithm for the case of polygonal domains as follows.

Algorithm. *Computing Shortest Point-Visible Paths in Polygonal Domains*

(1) If p is visible from either s or t, report the shortest Euclidean path between s and t.
(2) Compute the visibilty polygon of p ($\mathcal{V}_p$).

(3) Compute the invisible regions in which s and t lie. (May be two different regions).
(4) If the invisible regions of s and t are different, report the shortest Euclidean path between s and t, otherwise, name the invisible region in which both s and t lie, $\bar{V}$.
(5) Find the visibility gates on the boundary of $\mathcal{V}_p$ with respect to the source point s.
(6) Run the shortest path algorithm in the polygonal domain $\mathcal{P}$ twice, assuming s and t as the source point each time, to find the Shortest Euclidean distance from both s and t to every vertex of $\mathcal{P}$.
(7) Compute L-intervals on the visibility gates (the set $L_{s,t}$).
(8) For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t. Let q be the point among all $q(I)$ that has the minimum total distance.
(9) Report the shortest path from s to q appended by the shortest path from q to t.

The following theorem summarizes the result for the case of polygonal domains and provides a time bound on the algorithm.

Theorem 2. *Given a pair of points s and t and a viewpoint p inside a polygonal domain, the shortest p-visible path from s to t inside the free space can be found in $O(n \log n)$ time and $O(n \log n)$ space.*

Proof. The check in step 1 of the algorithm above can easily be done in linear time. The visibility polygon computation in step 2 can be done using the algorithm of Asano et al. [18] which requires $O(n \log h)$ time (h is the number of obstacles in the domain which is $O(n)$). To compute the invisible region in which s (resp. t) lies (step 3), we use the same method as the simple polygon case. For the next three steps (5, 6, and 7), we can use the shortest path algorithm of Hershberger and Suri [3] which takes $O(n \log n)$ time (see lemmas 3 and 4). Note that in step 7, we have to sort L_s and L_t and merge the two sets which requires $O(n \log n)$ time itself. Finally, the minimum point q can be computed in $O(n)$ time, since the size of $L_{s,t}$ is $O(n)$, and the property stated in lemma 1 holds in this case too. □

4. Polyhedral Surfaces

In this section, we consider the geometric domain of polyhedral surfaces, i.e. $\mathcal{P}$ is the surface of a polyhedron. In the case of convex polyhedra, $\mathcal{V}_p$ is a connected region whose boundary consists of $O(n)$ edges of $\mathcal{P}$. In non-convex case, $\mathcal{V}_p$ is a set of possibly disconnected regions of total complexity of $O(n^2)$. As the complexity of the visibility region of a point on a (possibly non-convex) polyhedron is quadratic in size of the polyhedron and the algorithms for finding the visibility map are superquadratic in general [19], we assume the visibility region of the point to be seen

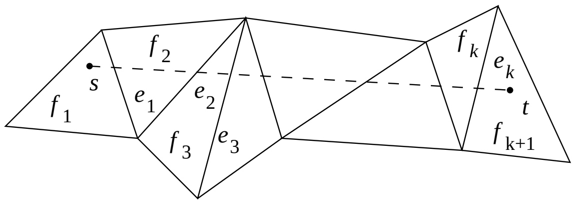

Fig. 5. Shortest path from s to t unfolded along its edge sequence

is determined through a preprocessing stage and focus on the problem of finding the shortest path meeting the visibility region.

The problem of finding the shortest path between two points on the surface of a polyhedron is well studied. Especially [5] presents a method for building a subdivision of the surface which can be used for finding the shortest paths from a fixed source to a given query point efficiently. The subdivision can be built in $O(n^2)$ time. The best known algorithm for finding shortest paths on polyhedral surfaces is [6] which finds the shortest path in $O(n \log^2 n)$ using *wavefront propagation* method.

4.1. *Shortest Paths on a Polyhedron*

We briefly review the terminology which is borrowed from [4]. Two faces f and f' are said to be *edge-adjacent* if they share a common edge e. A *sequence of edge-adjacent faces* is a list of one or more faces $\mathcal{F} = (f_1, f_2, \dots, f_{k+1})$ such that face f_i is edge-adjacent to face f_{i+1} (sharing common edge e_i). We refer to the (possibly empty) list of edges $\mathcal{E} = (e_1, e_2, \dots, e_k)$ as an *edge sequence* and to the vertex of face f_1 which is opposite e_1 as the *root* of $\mathcal{E}$.

Each face has a two dimensional coordinate system associated with it. If faces f and f' are edge-adjacent sharing edge e, we define the *planar unfolding* of face f' onto face f as the image of points of f' when rotated about the line through e into the plane of f such that the points of f' fall on the *opposite* side of e to points of f. Extending this notation, we say that we *unfold* an edge sequence $\mathcal{E} = (e_1, e_2, \dots, e_k)$ as follows: Rotate f_1 around e_1 until its plane coincides with that of f_2, rotate f_1 and f_2 around e_2 until their plane coincides with that of f_3, continue in this way until all faces $(f_1, f_2, \dots, f_k)$ lie in the plane of f_{k+1}.

A path on $\mathcal{P}$ is called *geodesic* if it is locally optimal and cannot be shortened by small perturbations. We say a path π *connects* edge sequence $\mathcal{E} = (e_1, e_2, \dots, e_k)$ if π consists of segments which join interior points of $e_1, e_2, \dots, e_k$ (in that order).

The following lemmas express some facts about geodesic paths:

Lemma 5 (Mitchell, Mount and Papadimitriou [4]). *If π is a geodesic path which connects the edge sequence $\mathcal{E}$, then the planar unfolding of π along $\mathcal{E}$ is a straight line segment.*

Lemma 6 (Sharir and Schorr [20]). *If $\mathcal{P}$ is a convex polyhedron then the shortest*

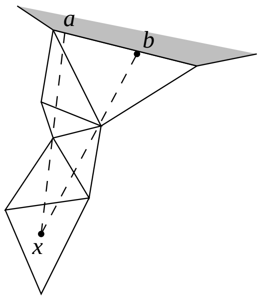

Fig. 6. The segment $\overline{ab}$ is an L-interval with respect to x. The shaded area is part of $\mathcal{V}_p$.

path between any two points on $\mathcal{P}$ cannot pass through a vertex of $\mathcal{P}$ except at its end-points.

Lemma 7 (Mitchell, Mount and Papadimitriou [4]). *The general form of a geodesic path is a path which goes through an alternating sequence of vertices and (possibly empty) edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of the path passing through a vertex is greater than or equal to π. The general form of an optimal path is the same as that of a geodesic path, except that no edge can appear in more than one edge sequence and each edge sequence must be simple.*

Lemma 8 (Mitchell, Mount and Papadimitriou [4]). *If $\pi(x)$ and $\pi(y)$ are optimal paths from s to points x and y, then they can intersect only at vertices of $\mathcal{P}$, and if they do at v, then the subpath of $\pi(x)$ from s to v has the same length as that subpath of $\pi(y)$ from s to v.*

Figure 5 shows a shortest path π from s to t unfolded along the edge sequence connected by it $(e_1, e_2, \ldots, e_k)$.

4.2. *Computing Shortest p-Visible paths*

Consider a maximal set of points on the boundary of $\mathcal{V}_p$ whose shortest paths to a point x connect the same edge sequence. Such an interval is an L-interval with respect to x (Figure 6). Clearly the set of all L-intervals with respect to a point x is a partition the boundary (Figure 7).

To compute the set of L-intervals with respect to a point x on the surface we use a subdivision presented in [5]. To obtain the subdivision, we cut the surface of $\mathcal{P}$ along the shortest paths from x to all of the vertices of $\mathcal{P}$. The resulting surface can be laid out on a common plane. The layout obtained in this manner is called the *inward layout* of $\mathcal{P}$ and is a star-shaped polygon [21]. The vertices of this polygon are the vertices of $\mathcal{P}$ and *images of the source point x* under different unfoldings and the edges are shortest paths from the source to the vertices of $\mathcal{P}$. A subdivision

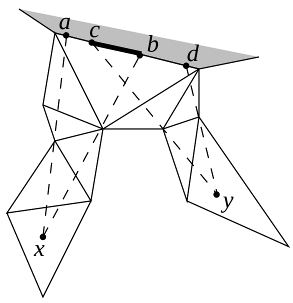

Fig. 7. $\overline{cb}$ is an L-interval with respect to the pair x and y.

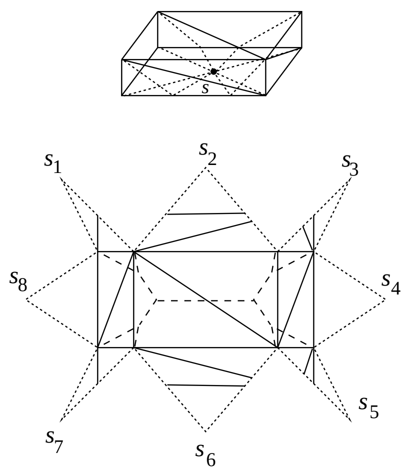

Fig. 8. The inward layout of a box. Solid lines are the edges of the subdivision that are part of polyhedron edges. Dashed lines are edges of the Voronoi diagram (ridges) and dotted lines are shortest paths from images of the source to vertices (cuts).

of the inward layout can be obtained by constructing the Voronoi diagram on the layout with respect to the images of the source point (Figure 8). This subdivision has the property that The points in the same region are closer to the corresponding image of the source than to other images, and their shortest paths from the source pass the same edge sequence. It is clear that the L-intervals with respect to point x are the intervals formed by intersection of the boundary of $\mathcal{V}_p$ and the subdivision mentioned above considering x as the source point.

For the non-convex case, the inward layout may overlap itself. The algorithm for computing Voronoi diagram, in this case, is slightly different and is given in [5]. In this case an L-interval with respect to x has the property that there is a vertex v of the polyhedron which every shortest path from x to a point on the interval

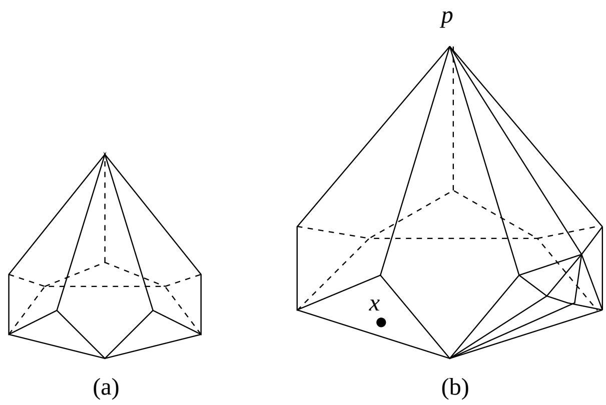

Fig. 9. (a) A diamond polyhedron which have $O(n)$ *sides*
(b) Diamond polyhedron with a number of faces added to it.

passes through v as the last vertex, and the edge sequence from v to points on the interval is the same. As the first part of the path is fixed among the points on interval, given an L-interval I with respect to s and t we can still find the point $q(I)$ whose total distance from s and t is minimum in constant time provided that during computing intervals we store the distance to pseudo-source associated with the interval.

To bound the number of L-intervals, observe that the subdivision has three kinds of edges: shortest paths to vertices (cuts), edges of the Voronoi diagram (ridges), and part of the edges of $\mathcal{P}$ that are obtained by intersecting ridges with the edges of $\mathcal{P}$. The number of cuts and ridges is $O(n)$, but there may be $O(n^2)$ edges of the third kind. In convex case, $\mathcal{V}_p$ consists of one or more faces of the polyhedron, so its boundary consists of polyhedron edges. Hence the number of L-intervals will be $O(n^2)$ in this case. For the non-convex case, $\mathcal{V}_p$ may be disconnected, with total complexity of $O(n^2)$ edges (not necessarily part of $\mathcal{P}$ edges). In this case, we do not have to keep track of intersections between $\mathcal{V}_p$ edges with those of $\mathcal{P}$. L-intervals on the boundary of $\mathcal{V}_p$ can be found by intersecting $\mathcal{V}_p$ edges with cuts and ridges, yielding to maximum of $O(n^3)$ intervals. Note that it can be shown that these bounds are tight (Fig. 9 shows a convex case).

Based on the above observations, we propose the algorithm for the case of polyhedral surfaces as follows. As we mentioned before, the algorithm in this case focuses on finding the shortest path assuming the visibility region is already computed.

Algorithm. *Computing Shortest Point-Visible Paths on Polyhedral Surfaces*

(1) If p is visible from either s or t, report the shortest Euclidean path between s and t.
(2) Run the shortest path algorithm on the surface $\mathcal{P}$ twice, assuming s and t as

the source point each time, to find the Shortest Euclidean distance from both s and t to every vertex of $\mathcal{P}$. As a result of this step, subdivisions of the surface are built with respect to s and t.

(3) Compute the L-intervals with respect to s and t and the set $L_{s,t}$.

(4) For each interval $I \in L_{s,t}$ find the point $q(I)$ which has the minimum total distance from s and t. Let q be the point among all $q(I)$ that has the minimum total distance.

(5) Report the shortest path from s to q appended by the shortest path from q to t.

The following theorem summarizes the result for the case of polyhedral surfaces and provides a time bound on the algorithm.

Theorem 3. *Given a pair of points s and t, a viewpoint p, and the visibility region of the viewpoint $\mathcal{V}_p$ on the surface of a polyhedron, the shortest p-visible path from s to t can be found in $O(n^2 \log n)$ time for the convex case and $O(n^3 \log n)$ for the non-convex case.*

Proof. The check in step 1 of the algorithm above can be easily done in time proportional to the size of the visibility region which is less than the time bounds stated for the whole algorithm. Using the algorithm of Chen and Han [5], we can compute the subdivisions with respect to s and t in time $O(n^2)$ for both cases (step 2). In step 3, we first compute L-intervals with respect to s and t separately. This taks $O(n^2 \log n)$ for the convex case and $O(n^3 \log n)$ for the non-convex case using algorithms for the segment intersection problem, such as [17]. This comes from $O(n^2)$ (resp. $O(n^3)$) bound for the number of L-intervals mentioned before for the convex (resp. non-convex) case. To find the set $L_{s,t}$, we must sort L_s and L_t and merge the end-point of the intervals. This can be done in the same time bound as finding L_s and L_t. Finally, the minimum point q can be computed in $O(n^2)$ (resp. $O(n^3)$) time for the convex (resp. non-convex) case. It can be easily verified that the space complexity of the algorithm is $O(n^2)$ (resp. $O(n^3)$). □

5. Conclusion

We studied the problem of finding the shortest path between two points with single-point visibility constraint in various domains. The time and space complexity of the proposed algorithms are summarized in Table 1.

Time bounds for the first two cases (two-dimensional ones) are the same as the bounds for the standard shortest path problem (without constraints), so these bounds cannot be improved any further. The case of polyhedral surfaces may be improved though, due to the subquadratic algorithm of Kapoor [6] for finding shortest paths on polyhedral surfaces that uses the wavefront propagation method.

There may be several extensions that can be considered. One extension is to use other metrics for distance computations (e.g. link-distance or weighted region)

Table 1. Time and space complexity of the proposed algorithms.

Domain	**Time**	**Space**
Simple Polygon	$O(n)$	$O(n)$
Polygonal Domain	$O(n \log n)$	$O(n \log n)$
Convex Polyhedral Surface	$O(n^2 \log n)$	$O(n^2)$
Non-convex Polyhedral Surface	$O(n^3 \log n)$	$O(n^3)$

while having the visibility constraints. For some cases, it is possible to use the same framework as we used in the current paper, i.e. constructing shortest path maps according to the metric used and finding the linearity intervals on the boundary. However, it is possible that for some metric, there may be more specific and more efficient algorithms.

Another extension is to constrain the path to meet an arbitrary general region, not necessarily a visibility region. Generalization of our algorithm to this case is straightforward for some domains (e.g. polygonal domains), since we have not used special properties about visibility regions in those domains. Other domains (e.g. simple polygons) require more study.

Acknowledgements

We would like to thank Mana Taghdiri for her contribution to the polyhedral surfaces case.

References

1. L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica* **2** (1987) 209–233.
2. J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.* **6** (1996) 309–332.
3. John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. Manuscript, Washington University (1995).
4. J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.* **16** (1987) 647–668.
5. J. Chen and Y. Han. Shortest paths on a polyhedron. *Internat. J. Comput. Geom. Appl.* **6** (1996) 127–144.
6. S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 32th Annu. ACM Sympos. Theory Comput.* (1999) 770–779.
7. Jean-Daniel Boissonnat, André Cérézo, and Juliette Leblond. Shortest paths of bounded curvature in the plane. *Internat. J. Intell. Syst.* **10** (1994) 1–16.
8. M. De Berg and M. Van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica* **18** (1997) 306–323.
9. A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms (SODA'2000)* (2000) 927–936.
10. J. Gudmundsson and C. Levcopoulos. Hardness result for TSP with neighborhoods.

Technical Report LU-CS-TR:2000-216, Department of Computer Science, Lund University (Sweden, 2000).

11. C. Mata and J. S. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.* (1995) 360–369.
12. Joachim Gudmundsson and Christos Levcopoulos. A fast approximation algorithm for TSP with neighborhoods and red-blue separation. *Lecture Notes in Computer Science* **1627** (1999) 473–482.
13. Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. In *Symposium on Discrete Algorithms* (2001) 38–46.
14. D. T. Lee. Visibility of a simple polygon. *Comput. Vision Graph. Image Process.* **22** (1983) 207–221.
15. B. Joe and R. B. Simpson. Correction to Lee's visibility polygon algorithm. *BIT* **27** (1987) 458–473.
16. Leonidas J. Guibas, E. McCreight, M. Plass, and J. Roberts. A new representation for linear lists. In *Proc. 9th Annu. ACM Sympos. Theory Comput.* (1977) 49–60.
17. B. Chazelle and H. Edlelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM* **39** (1992) 1–54.
18. Ta. Asano, Te. Asano, L.J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica* **1** (1986) 49–63.
19. J. O'Rourke. Visibility. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 25 (CRC Press LLC, 1997) 467–480.
20. M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.* **15** (1986) 193–215.
21. Pankaj K. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon. Star unfolding of a polytope with applications. Technical Report 031, Dept. Comput. Sci., Smith College, (Northampton, MA, 1993).