Finitely-Oriented Shortest Paths in the Presence of Polygonal Obstacles

Gabriele Reich*

Abstract

Given a set of non-intersecting simple polygons, a set of points in the plane, and a set of fixed orientations, we compute a graph such that for any two input points there exists a shortest path in G whose length equals the length of a shortest path between these points in the plane that does not intersect any polygon and that consists only of line segments of the given orientations. This graph may serve as a basis not only to compute shortest paths but also for various other network applications, like e.g. minimum spanning trees or Steiner minimal trees. We show that for f fixed orientations and n input points the size of the graph G is $O(f \cdot n \log n)$ and that G can be computed in time $O(f \cdot n \log n)$.

1 Introduction

Consider a situation in the plane and several network optimization problems for this situation. Although many optimization problems can be solved or approximated directly in geometry, it often will be useful to model the situation in the plane by a weighted, undirected distance graph and to solve or approximate the optimization problem on this graph. For example, minimum spanning trees, one-to-all

*Institut für Informatik, Universität Freiburg, Rheinstr. 10–12, W-7800 Freiburg, Germany, e-mail: reich@informatik.uni-freiburg.de

shortest paths or Steiner minimal tree approximations can be computed quite easily on such graphs. Therefore, in this paper we study the problem of computing a suitable graph from a situation in the plane.

Let the situation in the plane be as follows: given a set of non-intersecting simple polygons, the *obstacles*, a set of points, and a set of fixed orientations. The points may lie on the boundaries of the obstacles but they may also lie in the space of the plane that is not occupied by obstacles. Of course, points are not allowed to lie inside an obstacle. The aim now is to construct a graph G, containing a vertex for each input point, such that for each pair of input points p_i, p_j there exists in G a shortest path between p_i and p_j whose length equals the length of a shortest path between p_i and p_j in the plane, where the path in the plane consists only of line segments following the given orientations and avoiding all obstacles. We call such a graph a *shortest paths preserving* graph (*spp* graph for short).

Shortest paths preserving graphs for the *Manhattan plane*, where only horizontal and vertical line segments to connect the points are allowed, have been described e.g. in [1] and in [5]. In [1], a method has been presented to compute an *spp* graph for polygonal obstacles in the Manhattan plane in time $O(n(\log n)^2)$ for n input points; the algorithm of [5] computes an *spp* graph for orthogonal obstacles in the Manhattan plane in time $O(n \log n)$. First, we will show how the methods of [1, 5] can be generalized, such that an *spp* graph for polygonal obstacles and two fixed orientations can be computed in time $O(n \log n)$. This algorithm will then be used to solve the problem with f fixed orientations in time $O(f \cdot n \log n)$.

In the following section, we give some definitions. The *spp* graph for two orientations is defined in Section 3, its computation is described in Section 4. In Section 5, we generalize the algorithm to f fixed orientations.

2 Definitions

We assume that each obstacle in the input is specified by a list of vertices given in the order in which they are encountered when following the boundary in clockwise direction. For simplicity, let no two obstacles touch each other and let obstacle corner points and other input points be in general position, i.e., no two points share an x- or y-coordinate.

A path $p(p,q)$ between points p and q in the Manhattan plane is a sequence of horizontal and vertical line segments; the length of $p(p,q)$ is the sum of the lengths of all the line segments on $p(p,q)$. A shortest path $sp(p,q)$ is a path with minimum length among all the paths $p(p,q)$. The Manhattan distance or L_1-distance of points p, q in the plane with obstacles is the length of a shortest path between p and q that does not intersect an obstacle.

Let $F = \{\theta_1, \theta_2, \ldots, \theta_f\}$ be a set of orientations in the plane. Each orientation can be represented by the angle of its corresponding straight line with the positive x-axis. For each orientation θ, $0° \leq \theta < 180°$, where $0°$ is the orientation of the positive cartesian x-axis. Let the orientations $\theta_1, \ldots, \theta_f \in F$ be sorted counterclockwise, with θ_1 cyclically following θ_f. Two orientations θ_i, $\theta_{i+1} \in F$ are called *neighboring orientations*, for $1 \leq i \leq f$. A line or a line segment l with orientation $\theta(l)$ is said to be *θ_i-oriented* if $\theta(l) = \theta_i$. The *F-distance* d_F between points p and q is the Euclidean length of a shortest zig-zagging sequence of straight line segments, connecting p and q, where for each of the line segments l, $\theta(l) \in F$. In the plane with obstacles, $d_F(p,q)$ is the length of a sequence of θ_i-oriented line segments, for $\theta_i \in F$, such that no line segment intersects an obstacle.

Two points p and q are called *visible*, if the straight line segment l between p and q does not intersect the interior of an obstacle. If l is θ_i-oriented, then we say that p and q are *θ_i-visible* to each other.

3 Definition of the *spp* graph

Let us first define the *spp* graph G for the case that $f = 2$. Since it is always possible to rotate two fixed orientations θ_1, θ_2 into two orientations θ'_1, θ'_2, such that $\theta'_1 = 90°$ without changing intersections or distances of line segments and points [3], w.l.o.g. we assume that for given orientations θ_1, θ_2, $\theta_1 = 90°$. To simplify the presentation, we also assume that $\theta_2 = 0°$. Of course, this is a restriction of the general case with two arbitrary orientations, but the general case can be handled in the same way as the case that $\theta_1 = 90°$, $\theta_2 = 0°$ by replacing horizontal visibility by θ_2-visibility and L_1-distances by F'-distances, for $F' = \{\theta_1, \theta_2\}$.

Let $G = (V, E)$ be an undirected, connected distance graph with vertex set V, edge set $E \subseteq V \times V$ and distance function $d : E \rightarrow \Re$. Each vertex $v \in V$ represents a point in the plane, i.e., $V \subseteq \Re^2$. The length $d(e)$ of an edge $e = (p, p') \in E$ for $p = (x_p, y_p) \in V$ and $p' = (x_{p'}, y_{p'}) \in V$ is defined as the L_1-distance between p and p', i.e., $d(e) = |x_p - x_{p'}| + |y_p - y_{p'}|$.

Vertex set V consists of the set R_1 of input points and of two sets S_1 and S_2 of additional (Steiner) points to be defined later, i.e., $V = R_1 \cup S_1 \cup S_2$.

S_1 is created by projecting each point $p \in R_1$ horizontally onto obstacle line segments that are horizontally visible from p. Hence, S_1 contains at most $2|R_1|$ vertices. So far, no edges have been created. In contrast to [1], we neither project p also vertically onto obstacle line segments nor include edges connecting p with its projections into E. Note that the general position assumption no longer holds, since for each input point we create (at most) two points with the same y-coordinate as the input point. Of course, it may also happen that a point of S_1 shares its x-coordinate with an input point.

The set S_2 of Steiner points and the edges of G are defined recursively, following [5]. Let $X = x_1, x_2, \ldots, x_k$ be the ordered sequence of different x-values of points in $R = R_1 \cup S_1$. If $k = 1$, then R consists only of points with the same x-coordinate. In that case, no Steiner point or edge is created, i.e., $G = (R, E)$ and $E = \emptyset$. Otherwise, i.e., if $k > 1$, we partition R into two halves as follows. Let $x'_m \notin X$

be the median of the x-values in X. Let $R_{left} = \{(x, y) \in R \,|\, x < x'_m\}$ and $R_{right} = \{(x, y) \in R \,|\, x'_m < x\}$ and let $G_{left} = (V_{left}, E_{left})$ and $G_{right} = (V_{right}, E_{right})$ be the corresponding *spp* graphs for R_{left} and R_{right}. Partitioning lines are indexed in the order in which they are created. More precisely, the line that partitions the original point set R into R_{left} and R_{right} gets index 1, the two lines that partition R_{left} and R_{right} into halves get index 2 and so on.

Now for given G_{left} and G_{right}, G is computed as follows. Create a Steiner point $s' = (x'_m, y)$ for G, whenever a point $p = (x, y) \in R$ is horizontally visible from s', and create a connecting edge $e' = (p, s')$. Note that if there are several points lying on a common horizontal half line to the left of x'_m, i.e., there exist points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $\ldots$, $p_l = (x_l, y_l)$ where $y_1 = y_2 = \ldots = y_l$, $x_1 < x_2 < \ldots < x'_m$, and $p_1, \ldots, p_l$ are horizontally visible from x'_m, then we create only one vertex $s' = (x'_m, y_l)$ and the edge (p_l, s'). Similarly, for points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $\ldots$, $p_r = (x_r, y_r)$ on a common horizontal half line to the right of x'_m, in addition to the vertex $s' = (x'_m, y_1)$ only the edge (s', p_1) is created, where $x'_m < x_1 < \ldots < x_r$. Let S' be the set of all Steiner points s' and let E' be the set of connecting edges. Up to now, when creating the sets S' and E' we act exactly in the same way as described in [5]. Since we are not dealing with orthogonal obstacles but with arbitrary simple polygons as obstacles, we need some more edges and Steiner points.

Let y'' be the y-coordinate where a line segment e of an obstacle boundary intersects the line l, defined by $x = x'_m$. Let p_l be the first point in R_{left} hit on e when following e to the left of l and let p_r be the first such point in R_{right} when following e to the right of l. If there is no point p_l in R_{left} or p_r in R_{right}, then nothing is done; otherwise, we proceed as follows.

Consider Fig. 1 (a) – (d) for the cases that may arise depending on how the obstacle O on whose boundary p_l and p_r lie is located relative to e. In either case, we create a vertex $s'' = (x'_m, y'')$. In cases (a) and (c), we connect s'' with p_l by an edge e'', in cases (b) and (d) e'' connects s'' with p_r. These edges must be created since otherwise it would not be possible to reach from p_l or from p_r the part of l that lies above O (cases (a) and (d)), or to reach from p_r

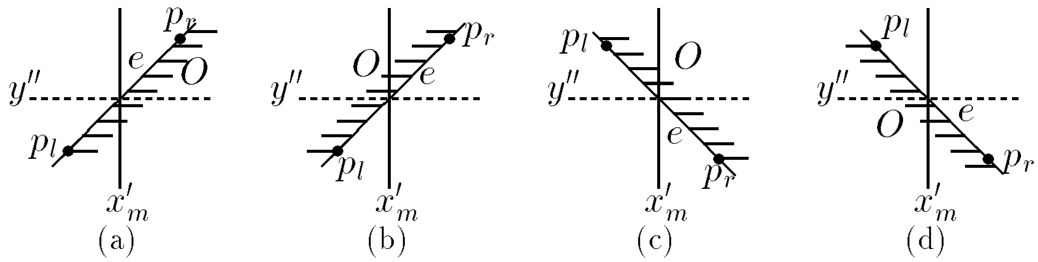


Figure 1:

or from p_l the part of l below O (cases (b) and (c)). On the other hand, the part of l above O can be reached from p_r and from p_l in cases (a) and (d), hence in case (a) no edge (p_r, s''), in case (d) no edge (p_l, s'') is necessary. Analogously, in case (b) no edge (p_l, s'') and in case (c) no edge (p_r, s'') is necessary. Although e'' does not correspond to a vertical or horizontal line segment, its length will be defined by the L_1-distance between its two endpoints s'', p_l or s'', p_r, respectively. Note that by the choice of x'_m there is no corner point of O on l, and hence either the edge (p_l, s'') or the edge (p_r, s'') will be created. We do not connect p_l and s'' or p_r and s'' by horizontally and vertically oriented line segments, since another obstacle O' may lie quite near to O. Then, we would be forced to use a sequence of an arbitrary large number of horizontal and vertical line segments, and as a consequence, we could not bound the number of edges in G. Let S'' be the set of all points s'', let E'' be the set of all edges e'', and let $S_2 = S' \cup S''$.

It remains to connect Steiner vertices lying on l. For each pair $s_i, s_j \in S_2$, we add an edge (s_i, s_j) to E'' whenever s_i and s_j are adjacent on l and (vertically) visible to each other.

Now compose G of G_{left} and G_{right} and the new created sets S_2, E', and E'' such that $G = (V, E)$ with $V = V_{left} \cup V_{right} \cup S_2$ and $E = E_{left} \cup E_{right} \cup E' \cup E''$.

Theorem 3.1: Given a set R_1 of polygon corner points and points in the plane, the graph G as defined above contains at least one shortest path between any pair of points in R_1 whose length equals the L_1-distance of these points in the plane with obstacles.

Proof: In G, the length of any edge $e \in E$ is defined by the L_1-distance of the two endpoints of e. This is even so, if e corresponds to a line segment in the plane that is neither horizontally nor vertically oriented. Also, the straight line segment between the two endpoints of any edge $e \in E$ does not intersect the interior of an obstacle. Hence, each path in G can be identified with a path of equal length in the plane, avoiding all obstacles. To show that for any shortest path between a pair of points in the plane there is a path of the same length in G, we will follow the argumentation of [5].

In [5] it is argued that any shortest path $sp(p,q)$ between points p and q in the plane can be partitioned into $sp(p,r_p)$, $sp(r_p,r_q)$, and $sp(r_q,q)$ with $r_p, r_q \in R_1$ and that a path $p(r_p,r_q)$ of the same length as $sp(r_p,r_q)$ exists in G. Since points of S_1, i.e., projections of input points, are treated like input points, for our purposes it suffices if $r_p, r_q \in R = R_1 \cup S_1$.

So let us consider a shortest path $sp(p,q)$ between points p and q in the plane and let $x = x'_m$ be the partitioning line with smallest index that is intersected by $sp(p,q)$. Assume that $sp(p,q)$ intersects $x = x'_m$ for the first time at a point p'. Let us further assume that $sp(p,q)$ reaches p' from the left. First, we will consider the situation where p' does not lie on an obstacle boundary.

Let $h_{p'}$ be the line segment of $sp(p,q)$ emanating from p' to the left, and let $t_{p'}$ be the left endpoint of $h_{p'}$, i.e., $t_{p'}$ is the turning point where $sp(p,q)$ changes direction for the last time before reaching p' (see Fig. 2 (b)). Consider Fig. 2 (a) – (c) for possible situations. If $t_{p'} = p$ (case (a)), then the edge (p,p') belongs to E, since p and p' are horizontally visible to each other. If an obstacle corner point $c_{p'}$ lies on $h_{p'}$ (case (b)), then the edge $(c_{p'},p')$ exists, and recursion proceeds with $r_p = c_{p'}$. Note that from the fact that the line segment $h_{p'} = (t_{p'},p')$ belongs to $sp(p,q)$ it follows that no obstacle can intersect $h_{p'}$. If none of the cases (a) and (b) occurs, then w.l.o.g. let us assume that the path from p' to p turns downwards at $t_{p'}$. Then we can move $h_{p'}$ downwards either until it hits an obstacle corner point $c_{p''}$ (case (c1)), or until it hits the endpoint of the line segment going down from $t_{p'}$ on $sp(p,p')$ (case (c2)). In either case, we get points $t_{p''}$ and p'', such that we can replace the path $p(t_{p''},p')$, containing point $t_{p'}$, by a path $p(t_{p''},p')$, containing p'', of the same length. In case (c1), $r_p = c_{p''}$; in case (c2), we apply our arguments to $sp(p, p'')$.

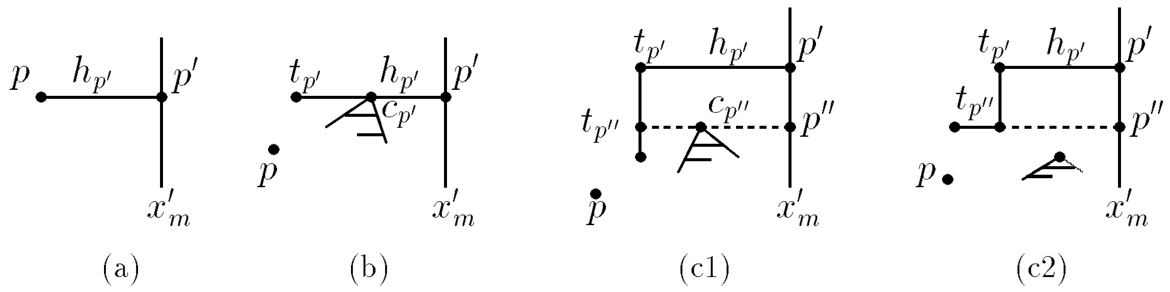


Figure 2:

Let us now consider the situation where p' lies on the boundary of an obstacle O. Then, we either have the situation shown in Fig. 1 (c), i.e., the edge e on which p' lies goes from the upper left to the lower right of p' and the interior of O is above p', or we have the situation shown in Fig. 1 (a), i.e., e goes from the lower left to the upper right with the interior of O below p'. Note that when p' lies on an obstacle boundary then there is no point $r_1 \in R_1$ on $h_{p'}$ (including $t_{p'}$), since any such point would have been projected onto e, thereby creating the vertex p' on the partitioning line $x = x'_m$. But the partitioning line has been chosen such that no point $r \in R$ exists with $x_r = x'_m$. Hence, r_1 cannot exist. Similarly, no point $s_1 \in S_1$ lies on $h_{p'}$, since s_1 must lie on a right boundary of an obstacle and must be caused by a point $r_1 \in R_1$, lying on $h_{p'}$. Obviously, if s_1 lies on the left boundary of an obstacle O, then s_1 cannot lie on $h_{p'}$, since $h_{p'}$ must have ended at the right boundary of O.

Again, we will consider only the case where the path from p' to p turns downwards at $t_{p'}$. Also, let the interior of O be below p' (see Fig. 3 (a)). If the path from p' to p turns upwards at $t_{p'}$ and the interior of O is below p' (case (i)) or if the path turns downwards and O is above p' (case (ii)), then we can proceed as described for cases (a) – (c) of Fig. 2, since in case (i) $h_{p'}$ can be moved upwards, in case (ii) $h_{p'}$ can be moved downwards without intersecting O. The case where the path from p' to p turns upwards at $t_{p'}$ with the interior of O above p' is symmetrical to the situation shown in Fig. 3 (a). Let p'' be the next point of V encountered when following e from p' to the left. Note that it does not matter whether $t_{p'}$ lies to the left or to the right of p''. Since the obstacle O intersects l, a corner point c of O must lie to the left of l. Any such point c causes a Steiner point

s'' on a partitioning line l', whenever the part of the input to which c belongs is combined at l' with another part of the input lying to the right of l'. By definition, the line l, defined by $x = x'_m$, is the partitioning line with smallest index that is intersected by $sp(p,q)$. Therefore, when combining two parts of the input at l, c must belong to the left part. Hence, when creating Steiner vertices on l, a Steiner vertex will be created that is caused by c. Obviously, this vertex is p', which therefore is a vertex of G. Since p'' belongs to V, p' lies to the upper right of p'', and the interior of O is below p' (cf. Fig. 1 (a)), the point p' as well as the edge e joining p'' and p' have been created and inserted into V and E, respectively. Note that p'' may but need not be a corner point of O.

Consider the horizontal half line $h_{p''}$ drawn from p'' to the left. To see that $h_{p''}$ intersects $sp(p,q)$ at a point $t_{p''}$, assume that such a point $t_{p''}$ does not exist. Hence, the path from p' to p which by assumption turns downwards at $t_{p'}$ must turn upwards or end at a point u, where $y_u > y_{p''}$ (see Fig. 3 (b)). Since $sp(p,p')$ is a shortest path, the turn at u must be caused by an obstacle O'. Obviously, a corner point c of O' must lie between u and $t_{p'}$, i.e., $y_u \leq y_c < y_{t_{p'}}$. But such a point c would have been projected onto e, causing a point c', lying between p' and p'' on e. By the choice of p'', such a point c' cannot exist. Hence, the point $t_{p''}$ exists (see Fig. 3 (c)). Note that p'' and $t_{p''}$ are visible to each other, since any obstacle blocking visibility either must have a corner point between $h_{p'}$ and $h_{p''}$, or must intersect $sp(p,p')$ between $t_{p''}$ and p', a contradiction.

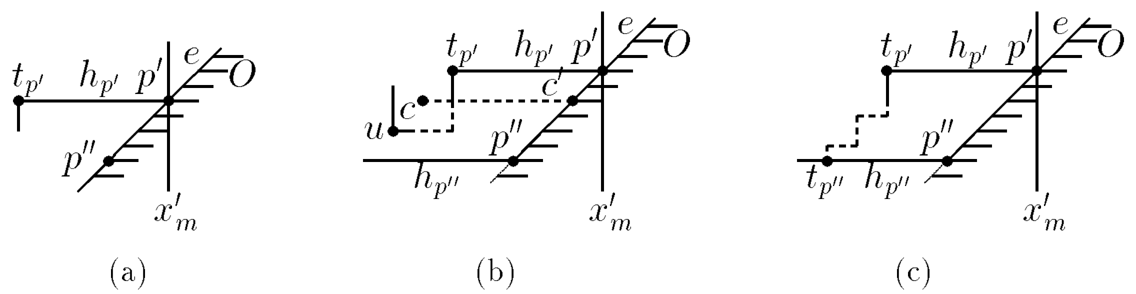
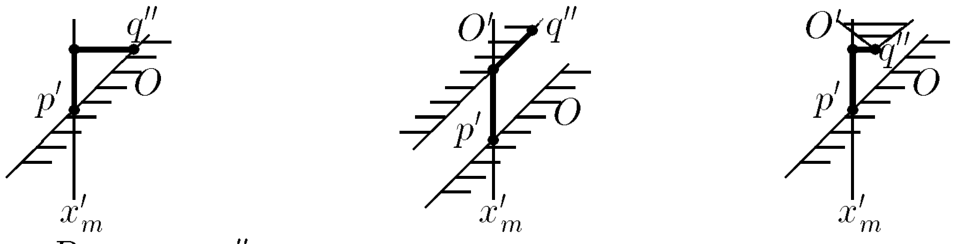


Figure 3:

Since $t_{p''}$ exists, we can replace $sp(p,p')$ by $sp(p,p'')$ and edge (p'',p') where the length of the latter two is the same as the length of $sp(p,p')$. If $p'' \in R$, then we have found a point r_p, otherwise, i.e., $p'' \notin R$, we apply our arguments to $sp(p,p'')$. A point r_q on $sp(p',q)$

can be found in the same way as described for r_p; see Fig. 4 (a) – (c) for possible situations when q'' is located on an obstacle boundary.



if $q'' \in R$ let $r_q = q''$
else continue with $sp(q, q'')$ (a)

same as (a) (b)

let $r_q = q''$ (c)

Figure 4:

From the fact that p' and the edge (p', p'') belong to G it follows that in G a path between r_p and r_q can be found with the same length as a shortest path between r_p and r_q in the Manhattan plane with polygonal obstacles. $\square$

To see that the size of the graph is $O(n \log n)$ for $n = |R_1|$, let us first neglect vertices s''. For each of the n input points we create at most two vertices, when projecting the input points onto obstacle boundaries. Additional vertices are only created on vertical partitioning lines. On a vertical line l partitioning n vertices, a new vertex is created for each of the at most n vertices that are horizontally visible from l. Since each line partitions the point set into halves, we get at most $O(n \log n)$ vertices. It remains to show that on l also at most n vertices s'' are created. But this follows from the definition of vertices s'', since for each of the n points partitioned by l, at most one vertex s'' is created, and since no vertex s'' is created on l for a point not belonging to the n partitioned points. Hence, the number of vertices in G is $O(n \log n)$. By similar arguments it can be shown that the number of edges in G is $O(n \log n)$, too.

4 Construction of the *spp* graph

After defining the *spp* graph G, now we will describe how to compute G efficiently. Following the recursive definition of G, we use a divide-and-conquer approach. Let us briefly recall how G is computed in [5], before describing the differences between our method and the method of [5], that are caused by the polygonal shape of the obstacles. To this end, consider a vertical slice cut off from the input situation. In [5], for a slice sl with left vertical boundary line x'_l and right vertical boundary line x'_r, sets L_{sl}, R_{sl}, B_{sl}, and GB_{sl} are maintained. L_{sl} consists of those points in sl that are horizontally visible from x'_l; R_{sl} is defined analogously to contain points of sl that are visible from x'_r. B_{sl} is the set of open y-intervals where horizontal visibility through sl is blocked by an obstacle, with the additional restriction that for each interval at least one obstacle corner point defining an interval end point lies in sl. Of those y-intervals that are defined by obstacles that also block horizontal visibility through sl but have no corner point in sl, called *globally blocking* obstacles, is taken care of in the set GB_{sl}. GB_{sl} is essential when determining vertical visibility on partitioning lines.

We can carry over the definition of the sets L_{sl} and R_{sl} without modifications to the situation with polygonal obstacles. B_{sl} needs only modified in that not only obstacle corner points but any point on an obstacle boundary can be used to determine an interval where visibility through sl is blocked. To determine vertical visibilities is slightly more complicated, since in our situation the y-interval that is occupied by a globally blocking obstacle remains not the same throughout the slice. But vertical visibilities only must be computed at partitioning lines. Hence, we do not use a single set GB_{sl} for a slice sl, but instead we take care of y-intervals where vertical visibility is blocked in two sets GB^l_{sl} and GB^r_{sl}, one for y-intervals on the left boundary of sl, the other for y-intervals on the right boundary of sl.

In addition to the described sets, for a slice sl we maintain two sets A^l_{sl} and A^r_{sl}, one for the left boundary of sl, one for the right, where we take care of points s'' that must be created because one of the situations shown in Fig. 1 (a) – (d) arises (see Section 3 for the definition of points s''). Let an element of a set A_{sl} be a point $p \notin R$ that lies on a line segment l of an obstacle boundary. With p associated in A_{sl} are the line segment l and a point p' that lies to the right of p on l if $p \in A^l_{sl}$, and that lies to the left of p on l if $p \in A^r_{sl}$.

Before going into the details of the computation of the sets, let us introduce some notation. We say that an obstacle O, where a point on the boundary of O lies in sl, is *local* with respect to sl; an obstacle O', where no point lies on the part of the boundary of O' that belongs to sl, is called *global* with respect to sl. Let us denote the intersection of a line segment l with the vertical line $x = x'$ by $is(l, x')$. For a point $p \in R$, let $up(p)$ be the next line segment encountered when going upwards from p, and let $down(p)$ be the next line segment when going downwards from p. Let $visible(p, x)$ be a function returning true, if horizontal visibility between p and x is not blocked by an obstacle; if visibility is blocked, $visible(p, x)$ returns false.

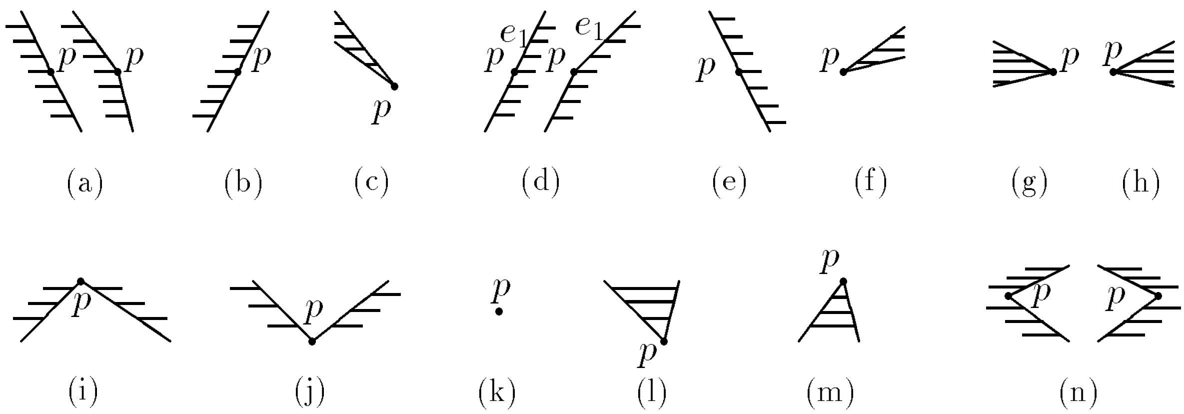


Figure 5:

Let sl be a slice containing a single point $p \in R$. Consider Fig. 5 (a) – (n) for possible situations in sl. We assume that the input points be classified in advance such that for a given point p it is possible to determine in constant time which of the cases (a) – (n) of Fig. 5 arises. We assume further that when given a point $p \in R$ or a line segment l, we can determine in constant time the two line segments meeting at p or the two endpoints of l. By definition, no line segment contains more than two points of R. That is, if by projecting points onto obstacle boundaries several points lying on a common line segment have been created, then any such line segment l is cut into segments $l_1, \ldots, l_k$, such that each of the segments $l_1, \ldots, l_k$ contains exactly two points, namely its two end points. Then, with the values of $up(p)$ and $down(p)$ associated with each point p, $visible(p, x'_l)$ or $visible(p, x'_r)$ and the sets L_{sl}, R_{sl}, B_{sl}, A^l_{sl}, and A^r_{sl} for $sl = (x'_l, x'_r)$ can be computed in constant time. As an example, we will describe the computation of the sets for case (d).

if case (d) then
 if $visible(p, x'_l)$ then $L_{sl} = \{p\}$ else $L_{sl} = \emptyset$;
 $R_{sl} = A^l_{sl} = \emptyset$;
 $A^r_{sl} = \{is(e_1, x'_r)\}$;
 $B_{sl} = \{(\min(y_{is(down(p),x'_l)}, y_{is(down(p),x'_r)}), y_{is(e_1,x'_r)})\}$;

Since the points no longer are in general position, it may happen that sl contains some points with the same x-coordinate. In that case, we compute the sets by considering each point in sl in turn. If we implement all sets as linear linked lists, then the time needed to complete the sets according to all points of sl is linear in the number of points in sl.

After describing the computation of the various sets for a slice containing only points with the same x-coordinate, let us now look at the merge of two slices. The merge of two slices can be done in the same way as in the situation with orthogonal obstacles, except that we must take the points in A^l and A^r into consideration and adjust these sets to the boundaries of the new slice. Let the two slices to be merged be $sl_{left} = (x'_l, x'_m)$ and $sl_{right} = (x'_m, x'_r)$ and let the resulting slice be $sl = (x'_l, x'_r)$.

$merge(sl_{left}, sl_{right})$:
 $A^l_{sl} = A^l_{sl_{left}}$;
 $A^r_{sl} = A^r_{sl_{right}}$;
 for each $p \in R_{sl_{left}} \cup L_{sl_{right}}$
  add a vertex $v = (x'_m, y_p)$ and an edge (v, p) to G;
 for each $p \in A^r_{sl_{left}} \cup A^l_{sl_{right}}$
 {let p' and l be the point and the line segment associated with p
  in $A^r_{sl_{left}}$ or $A^l_{sl_{right}}$}
  add a vertex $v = p$ and an edge (p', v) to G,
   where the length of (p', v) is the L_1-distance of v and p';
  if $p \in A^r_{sl_{left}}$ and l intersects x'_r then
   add a point $p'' = is(l, x'_r)$ to A^r_{sl};
   associate p and l with p'';
  if $p \in A^l_{sl_{right}}$ and l intersects x'_l then
   add a point $p'' = is(l, x'_l)$ to A^l_{sl};
   associate p and l with p'';

$L_{sl} = L_{sl_{left}} \cup (L_{sl_{right}} \setminus B_{sl_{left}})$;
$R_{sl} = R_{sl_{right}} \cup (R_{sl_{left}} \setminus B_{sl_{right}})$;
$B_{sl} = B_{sl_{left}} \cup B_{sl_{right}}$;
connect visible pairs of vertices on x'_m.

Note that by definition l ends at the next point of R on the obstacle boundary. Hence, if l intersects line $x = x'_m$ or $x = x'_r$ then there is no point on l between p and p''.

It remains to describe how visible pairs of vertices v on the line $x = x'_m$ can be connected. To connect visible vertices, we must know where obstacles block vertical visibility on the line $x = x'_m$. Obstacles that are local within sl easily can be determined by inspecting $R_{sl_{left}}$, $L_{sl_{right}}$, $A^r_{sl_{left}}$, and $A^l_{sl_{right}}$. But from the definition of global obstacles it follows immediately that they cannot be computed from the information available in sl, since no corner point or other point on the boundary of a global obstacle lies in sl. Hence, we must take care of global obstacles when partitioning the input. Analogously to the situation with orthogonal obstacles, a polygonal global obstacle O for a slice sl' is created, when a slice sl is cut into slices sl' and sl'', such that two boundary line segments of O with the interior of O in between them extend fully through sl' and there is at least one point on the boundary of O in sl''. Hence, when partitioning a slice sl into slices sl_{left} and sl_{right}, we inspect the points in sl_{left} whether they lie on the boundary of an obstacle that becomes global for sl_{right}, and we inspect the points in sl_{right} whether they give rise to a global obstacle in sl_{left}. Thereby we create the sets $GB^l_{sl_{left}}$ and $GB^r_{sl_{left}}$ for sl_{left}, and $GB^l_{sl_{right}}$ and $GB^r_{sl_{right}}$ for sl_{right}.

For a slice $sl = (x'_l, x'_r)$, there may be as much as $O(|R_1|)$ global obstacles. Of course, we cannot afford to take care of each of these explicitly, and it can be seen that this is not necessary, either. If there are two or more global obstacles $O_1, \ldots, O_k$ in sl, numbered from bottom to top, with no point in between them, then vertical visibility is blocked from O_1 to O_k, and it does not matter how many obstacles cause the blocking. Hence, it suffices to make two large intervals, one on x'_l, the other on x'_r, for all of the k obstacles together, instead of storing k intervals on either boundary of sl. It follows that the number of intervals in GB^l_{sl} and GB^r_{sl} will be linear in the number of points in sl.

With the information about globally blocked intervals on partitioning lines available, we can complete the merge of two slices as follows:

for each pair (v, v') adjacent in y on $x = x'_m$
determine by inspecting $R_{sl_{left}}$, $L_{sl_{right}}$, $A^r_{sl_{left}}$, $A^l_{sl_{right}}$, $GB^r_{sl_{left}}$, and $GB^l_{sl_{right}}$
whether v and v' are visible to each other;
if so, add an edge (v, v') to G.

Let us now consider the runtime of our algorithm.

As already mentioned, we implement the various sets as linear linked lists, and therefore, their computation for a slice sl takes time linear in the number of points in sl. We sort the linear lists in increasing y-order to ensure that the set union and set difference operations during the merge of slices can be performed by scanning in parallel through the involved lists. The time needed for the set union and set difference operations then will be linear in the number of list elements, which itself is linear in the number of points in the involved slices. Note that the sets created for a slice sl with n points sharing the same x-coordinate are already sorted in y-value, if the points in sl are considered in increasing y-order. It follows that the input points must be sorted in y-value.

As already described, sets $GB^l_{sl'}$ and $GB^r_{sl'}$ for a slice sl', created by partitioning a slice sl into sl' and sl'', can be determined by inspecting the points in sl''. To get the sets $GB^l_{sl'}$ and $GB^r_{sl'}$ sorted in y-order without explicitly sorting, here it is also essential that the points in sl'' are given in increasing y-order. Adjacent global obstacles with no point in between in sl' can be determined and collapsed in time linear in the number of points in sl by scanning in parallel through $GB^l_{sl'}$, $GB^r_{sl'}$ and through the set of y-ordered points in sl'.

When partitioning a slice sl into sl_{left} and sl_{right}, the x- and y-ordered sequences of the points in R for sl_{left} and sl_{right} and the (y-ordered) sets $GB^l_{sl_{left}}$, $GB^r_{sl_{left}}$, $GB^l_{sl_{right}}$, and $GB^r_{sl_{right}}$ must be computed. Given the points in sl sorted by x- and by y-value, all this can be done in time linear in the number of points in sl. The median x'_m for partitioning can be found in constant time for an x-ordered point set.

From the facts that each partitioning and each merge takes time linear in the number of involved points, that the sets L_{sl}, R_{sl}, B_{sl}, A^l_{sl}, and A^r_{sl} for a slice sl, containing only points with the same x-coordinate, also can be computed in time linear in the number of points in sl, and that the depth of the recursion is bounded by $O(\log n)$ for $n = |R_1|$, it follows that — apart from the necessary preprocessing — the shortest paths preserving graph G can be computed in time $O(n \log n)$. It remains to analyze the preprocessing.

First of all, we have to determine projections of input points onto obstacle boundaries. After sorting the input points in increasing y-order, this can be done in time $O(n \log n)$ for $n = |R_1|$ by sweeping a horizontal scan line over the plane, see e.g. [2]. Now we must classify the at most $3n$ points according to the cases (a) – (n) of Fig. 5, and determine for each point p $up(p)$ and $down(p)$. This again can be done in time $O(n \log n)$ by first sorting the input points in increasing x-order and then by using a scan line algorithm, where here the scan line proceeds from the left to the right. Altogether, the whole preprocessing can be completed in time $O(n \log n)$. Hence, G can be computed in time $O(n \log n)$.

Let X_{sl} be the set of points in sl sorted in x-value, and let Y_{sl} be this set sorted in y-value. Then, the following algorithm which is quite similar to the algorithm presented in [5] computes an *spp* graph for n points in time $O(n \log n)$.

Algorithm *spp* graph $(X_{sl}, Y_{sl}, GB^l_{sl}, GB^r_{sl}, x'_l, x'_r)$
{computes an *spp* graph for points X_{sl} in slice $sl = (x'_l, x'_r)$}

If X_{sl} contains only points with the same x-coordinate
then compute L_{sl}, R_{sl}, B_{sl}, A^l_{sl}, A^r_{sl} directly;
else partition slice sl at vertical line x'_m into slices
$sl_{left} = (x'_l, x'_m)$ and $sl_{right} = (x'_m, x'_r)$;
partition X_{sl} into $X_{sl_{left}}$ and $X_{sl_{right}}$;
partition Y_{sl} into $Y_{sl_{left}}$ and $Y_{sl_{right}}$;
compute $GB^l_{sl_{left}}$ and $GB^r_{sl_{left}}$ from GB^l_{sl} and $Y_{sl_{right}}$;
compute $GB^l_{sl_{right}}$ and $GB^r_{sl_{right}}$ from GB^r_{sl} and $Y_{sl_{left}}$;
spp graph $(X_{sl_{left}}, Y_{sl_{left}}, GB^l_{sl_{left}}, GB^r_{sl_{left}}, x'_l, x'_m)$;

spp graph $(X_{sl_{right}}, Y_{sl_{right}}, GB^{l}_{sl_{right}}, GB^{r}_{sl_{right}}, x'_m, x'_r)$;
$merge(sl_{left}, sl_{right})$.

5 The *spp* graph for f orientations

In the previous sections, we have shown how to compute an *spp* graph in time $O(n \log n)$ if the set F of given orientations solely consists of two orientations. Let us now return to the more general situation where $F = \{\theta_1, \theta_2, \ldots, \theta_f\}$.

We start with orientations θ_1 and θ_2. Let θ_1 be the orientation in which the partitioning of the input points is done, and let θ_2 determine the (θ_2-) visibility between points in V and partitioning lines as well as the direction, in which input points are projected onto obstacle boundaries. Then we construct graph G as described in Section 3. We proceed with orientations θ_2 and θ_3, then θ_3 and θ_4 and so on until θ_f determines partitioning lines and θ_1 visibility. Of course, we do not compute f distinct graphs, but instead when creating new Steiner vertices and edges for pairs of orientations θ_i and θ_{i+1}, $2 \leq i \leq f$, we insert these vertices and edges into the already existing graph G. Note that we do not create extra vertices when two line segments represented by edges e_1 and e_2 in G intersect each other in the plane. Hence, G merely consists of f edge-disjoint subgraphs having a set of basic vertices, namely the n input points, in common.

From the construction and the number f of orientations it follows that graph G has size $O(f \cdot n \log n)$ and also can be computed in time $O(f \cdot n \log n)$.

Theorem 5.1: Given a set of input points and a set F of fixed orientations, graph G constructed as described before contains for each pair of input points a shortest path whose length equals the F-distance of these points in the plane with obstacles.

Proof: For any pair p, q of input points in the plane where the straight line segment l connecting p and q does not intersect an obstacle and $\theta(l) \notin F$, the shortest path in the given orientations is a staircase, consisting of line segments with orientations θ_i and θ_{i+1}, where $\theta_i < \theta(l) < \theta_{i+1}$ (see e.g. [4]). In the plane with obstacles, any shortest path between points p and q consists of several staircases, each consisting of line segments with neighboring orientations.

From Theorem 3.1 and the construction of G it is obvious that in G for each pair of input points a shortest path consisting of edges with neighboring orientations can be found. It remains to be shown that two staircases with different orientations always can be joined at one of the input points. To this end assume that a shortest path between points p and q contains a path (p', q') consisting of two staircases st_1 and st_2, connected at a point r, where st_1 and st_2 consist of line segments with together at least three orientations. Assume further that no input point lies on st_1 and st_2 (see Fig. 6). But then it is possible to change st_1 and st_2 such that they contain an input point and that the path (p', q') becomes shorter (see the dashed lines in Fig. 6), a contradiction. □

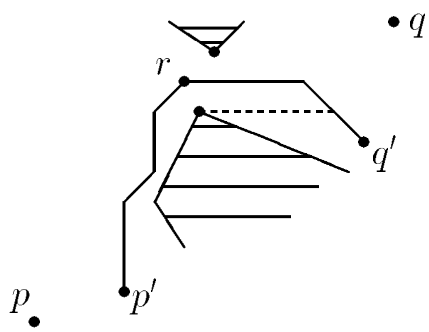


Figure 6:

6 Conclusion

We have presented a method to compute a shortest paths preserving graph for a set of points and a set of polygonal obstacles in the plane, where only fixed orientations may be used. For f fixed orientations, this graph can be computed by applying f times an algorithm that computes a shortest paths preserving graph for two fixed orientations. We have shown that for n obstacle corner and other input points, the graph has $O(f \cdot n \log n)$ vertices and edges and can be computed in time $O(f \cdot n \log n)$.

Acknowledgement

Thanks are due to P. Widmayer for fruitful discussions and to B. Nilsson for comments on an earlier draft of this paper.

References

[1] K.L. Clarkson, S. Kapoor, P.M. Vaidya: Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time, Proc. 3rd Annual Symposium on Computational Geometry, 1987, 251–257.

[2] F.P. Preparata, M.I. Shamos: Computational Geometry: An Introduction, Springer, New York, 1990.

[3] G.J.E. Rawlins: Explorations in restricted orientation geometry, Ph.D. Thesis, Tech. Rep. No. CS-87-57, Department of Computer Science, University of Waterloo, Canada, 1987.

[4] P. Widmayer, Y.F. Wu, C.K. Wong: On some distance problems in fixed orientations, SIAM J. Computing, Vol. 16, 1987, 728–746.

[5] P. Widmayer: On shortest paths in VLSI design, Tech. Report, Institut für Informatik, Universität Freiburg, Germany, 1990, also to appear in: Annals of Operations Research, 1991.