Shortest Inspection-Path Queries in Simple Polygons

Christian Knauer, Günter Rote, and Lena Schlipf

Institut für Informatik, Freie Universität Berlin,
Takustraße 9, D–14195 Berlin, Germany.
{knauer,rote,schlipf}@inf.fu-berlin.de

Abstract. We want to preprocess a simple n-vertex polygon P to quickly determine the shortest path p from a fixed source point $s \in P$ to view a set $Q \subseteq P$ of query points (i.e., such that each point $q \in Q$ is visible from some point on the path p). We call such queries *shortest inspection-path queries*. For $|Q| \leq 2$ we describe data structures that answer such queries in logarithmic time. The structures have linear (for $|Q| = 1$) respectively quadratic (for $|Q| = 2$) size and preprocessing time. For $|Q| = 2$ we also present a data structure with linear preprocessing time and space which achieves for any $\epsilon > 0$ a query time of $O(\frac{1}{\epsilon^2} \log n)$ at the expense of providing only a $(1 + \epsilon)$-approximate answer.

Keywords: Computational geometry, Simple polygons, Shortest paths, Visibility.

1 Introduction

Many variations of the problem of computing shortest paths in *simple* polygons have been studied in the past, c.f. [3]. One instance of the problem is to find the shortest path from a given *fixed* source point s in a simple polygon P with n vertices to view a set $Q \subseteq P$ of query points. Our goal in this paper is to preprocess the input (P, s) to answer queries of this type: Given a set $Q \subseteq P$ of query points, find the shortest distance one needs to travel in P from s to see all points in Q, i.e., such that each point in Q is visible from some point on the path travelled. Such queries are called *shortest inspection-path queries*.

In polygons *with holes* the (off-line) problem is clearly NP-hard for $|Q| = \Omega(n)$, by a reduction from TSP. However, in simple polygons the off-line problem can most likely be solved in polynomial time using techniques used to solve the watchman path problem [1].

For $|Q| = 1$ a query can be answered in $O(n)$ time without preprocessing [8], and in $O(\log n)$ time with $O(n^2)$ preprocessing time and space [9]. We improve and simplify the latter result and describe a solution with linear preprocessing time and space that achieves $O(\log n)$ query time. For $|Q| = 2$ we describe a solution with quadratic preprocessing time and space that also achieves logarithmic query time. These results are summarized in the following

Theorem 1. *Let P be a simple polygon with n vertices and let $s \in P$ be a fixed point. We can preprocess (P, s)*

- *in $O(n)$ time into a data structure of $O(n)$ size that can answer shortest inspection-path queries for $Q \subseteq P$ with $|Q| = 1$ in $O(\log n)$ time, and*
- *in $O(n^2)$ time into a data structure of $O(n^2)$ size that can answer shortest inspection-path queries for $Q \subseteq P$ with $|Q| = 2$ in $O(\log n)$ time.*

We will actually prove a stronger result for the case $|Q| = 2$ where the source point s is not fixed but is part of the query.

For the case $|Q| = 2$ (with a fixed starting point s) we also describe a solution with *linear* preprocessing time and space that achieves logarithmic query time but that provides only an approximate answer. For a constant $c > 1$ a *c-approximate shortest inspection-path query* for a set $Q \subseteq P$ of query points asks for (the length of) a path from the fixed source point s to view the set Q that is at most by a factor of c longer than the shortest inspection-path for Q from s. The result is summarized in the following

Theorem 2. *Let P be a simple polygon with n vertices and let $s \in P$ be a fixed point. We can preprocess (P, s) in $O(n)$ time into a data structure of $O(n)$ size that can answer for any $\epsilon > 0$ a $(1 + \epsilon)$-approximate shortest inspection-path query for $Q \subseteq P$ with $|Q| = 2$ in $O(\frac{1}{\epsilon^2} \log n)$ time.*

2 Preliminaries

The visibility polygon of a point $q \in P$ will be denoted by $V(q)$, the shortest path in P between two points $x, y \in P$ by $p(x, y)$ and the shortest path tree from s in P by T_s. For a shortest path $p = p(x, y)$ we denote by $\hat{x}$ (resp. $\hat{y}$) the *first vertex of P on p after x* (resp. the *last vertex of P on p before y*). If we remove $V(q)$ from P, the polygon splits into disconnected regions that we call *invisible regions*. Each such region has exactly one edge in common with $V(q)$. By $P_s(q)$ we denote the region which contains s. The common edge $w(q)$ between $V(q)$ and $P_s(q)$ will be called *the window of q*. Let a and b be the endpoints of $w(q)$, and r be the last common vertex between the two paths $p(s, a)$ and $p(s, b)$ (i.e., the lowest common ancestor $LCA_{T_s}(a, b)$ of a and b in T_s), c.f. Fig. 1. The paths $p(r, a)$ and $p(r, b)$ together with the segment $w(q)$ form the *funnel of q* which will be denoted by $F(q)$, c.f. Fig. 2; the vertex r is called the *root* of the funnel, the segment $w(q)$ is called the *base* of the funnel. Note that the paths $p(r, a)$ and $p(r, b)$ are outward convex. In Section 4 we require a generalization of the notion of a funnel which was introduced in [5]: The *hourglass* between two line segments $l_1, l_2 \subseteq P$ is the boundary of the union of all shortest paths $p(x, y)$ for $x \in l_1$, $y \in l_2$; it will be denoted by $H(l_1, l_2)$. For two paths p, q we denote the concatenation of p and q by $p + q$. Finally, the (Euclidean) length of the path p will be denoted by $|p|$.

In Section 3 we first give a structural characterization of the solution that forms the basis of our approach and then prove the case $|Q| = 1$ of Theorem 1. We consider the case $|Q| = 2$ in Section 4 and provide some conclusions in Section 5.

3 The data structure for one query point

If $Q = \{q\}$ we have to find a point $c \in P$ visible from the query point $q \in P$ that has the shortest distance from s. If q is invisible from s, then s lies in an invisible region (if q is visible from s, then clearly $c = s$). In this case it is easy to see that the point c lies on the window $w(q)$, in particular c is the point on $w(q)$ that has the shortest distance to s, c.f. Fig. 1.

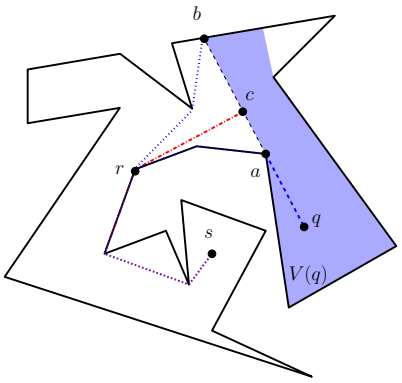

Fig. 1. The window $w(q) = ab$ separates $P_s(q)$ and $V(q)$. The point with shortest distance to s that is visible from q is c. The drawing shows $p(s, a)$, $p(s, b)$ and $p(s, c)$.

We next describe a simple characterization of c given in [9]. To this end let a and b be the endpoints of $w = w(q)$, and r be the lowest common ancestor of a and b in T_s, i.e., the last common vertex between the two paths $p(s, a)$ and $p(s, b)$, c.f. Fig. 1.

The paths $p(r, a), p(r, b)$ together with the segment $w(q) = ab$ form the *funnel* of $w(q)$ which will be denoted by F, c.f. Fig. 2. Note that the paths $p(r, a), p(r, b)$ are outward convex.

Let $a = v_0, v_1, \ldots, r = v_m, \ldots, v_k, v_{k+1} = b$ denote the vertices of the funnel from a to b. F can be decomposed into triangles by extending the edges of F until they intersect $w(q)$. Let x_i denote the intersection point of the extension of the edge $v_i v_{i+1}$ with $w(q)$ (hence, $x_0 = a$ and $x_k = b$). The shortest path from s to points on the segment $x_i x_{i+1}$ passes through v_{i+1} as the last vertex of P. Denote the angles between the extension edges and the window by $\theta_0, \theta_1, \ldots, \theta_k$, i.e., $\theta_i = \angle b x_i v_i$ for $0 \leq i < k$ and $\theta_k = \pi - \angle a b v_k$.

The outward convexity of the paths $p(r, a), p(r, b)$ implies that the sequence $\theta_0, \theta_1, \ldots, \theta_k$ is increasing. As a consequence we can characterize the optimal contact point c in the following way:

1. $\theta_i = \pi/2$ for some $0 \leq i \leq k$. In this case, $c = x_i$.

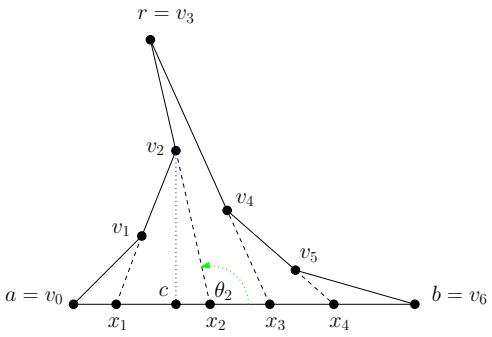

Fig. 2. The funnel F over the window $w(q) = ab$. The optimal point c is the foot of the perpendicular from v_2 to $w(q)$.

2. $\theta_i < \pi/2$ and $\theta_i + 1 > \pi/2$ for some $0 \leq i \leq k$. In this case, c is the foot of the perpendicular from v_{i+1} to $w(q)$.
3. $\theta_0 > \pi/2$. In this case, $c = a$.
4. $\theta_k < \pi/2$. In this case, $c = b$.

We can therefore search for c by looking at the angles θ_i: If $\theta_i > \pi/2$ then c lies left of x_i, whereas if $\theta_i < \pi/2$ then c lies right of x_i.

To answer a query q we will proceed in two steps: First we compute the window $w(q)$ of q along with the funnel root r. Then we compute the optimal point c on $w(q)$.

After the preprocessing phase, the first step and the second step can be done in $O(\log n)$ time.

3.1 Preprocessing phase

1. Store the vertices of P in a circular array A, sorted in clockwise order along the boundary of P.
2. Compute a data structure $\mathcal{D}_1$ that supports $O(\log n)$ time shortest path queries in P between any pair of points $u, t \in P$ [4].
3. Compute a data structure $\mathcal{D}_2$ that supports $O(\log n)$ time ray-shooting queries to P [7].
4. Compute the shortest path tree T_s [5] and preprocess it to support $O(1)$ time LCA-queries [6].

The total preprocessing time and space is $O(n)$.

3.2 Query phase

In the query phase we will check at first if q is visible from s (in this case $c = s$). This can be done in $O(\log n)$ time by shooting a ray from q in the direction of s and testing if the boundary of P is hit before s. In the following we can assume that q is not visible from s.

Computing the funnel. To find $w(q) = ab$ and r in $O(\log n)$ time in the first step of the query phase we proceed as follows: Since the window separating s from $V(q)$ is specified by the last vertex of P on the shortest path from s to q (Fig. 1), we can find $a = \hat{q}$ in $O(\log n)$ time via $\mathcal{D}_1$ (c.f. [3], Corollary 8.4.14). To find b in $O(\log n)$ time we shoot a ray from q in the direction of a. Next, we compute $v_k = \hat{b}$ in $O(\log n)$ time via $\mathcal{D}_1$, and finally, we get the funnel root $r = LCA_{T_s}(a, v_k)$ in $O(1)$ time.

Computing the optimal point on the window. To find the optimal point c on $w(q)$ in $O(\log n)$ time in the second step of the query phase we proceed as follows:

- First we check if $\theta_0 > \pi/2$ or $\theta_k < \pi/2$. In the first case $c = a$, in the second case $c = b$, and in either case we are finished.
- Next we look at the extensions of the edges emanating from the apex $r = v_m$ of the funnel. If $\theta_{m-1} = \pi/2$, or $\theta_m = \pi/2$, or $\theta_{m-1} \leq \pi/2 < \theta_m$, c is the foot of the perpendicular from v_m to $w(q)$ and we are finished.
- If $\theta_{m-1} > \pi/2$, then $\theta_i > \pi/2$ for $m \leq i \leq k$, since the angle sequence is increasing. In particular c is the foot of the perpendicular from some vertex v_i to $w(q)$, where v_i is on the left side $p(r, a)$ of the funnel $F(q)$, i.e., $1 \leq i < m$. To determine for which vertex v_i the perpendicular to $w(q)$ has to be drawn, we would like to perform a binary search on the sequence $v_0, \ldots, v_k$. However this sequence is not directly accessible, so we use the array A instead, and perform a binary search on the interval $[r, a]$ in A (if $r = s$ and s is not a vertex from P, we take the first vertex $\hat{s}$ after s on $p(s, a)$ and search in the interval $[\hat{s}, a]$ instead). For a vertex u in this interval we compute $LCA_{T_s}(u, a)$, which is one of the vertices $v_0, \ldots, v_m$ on the left edge of the funnel, say v_i. By computing the angle θ_i we can decide if the binary search has to continue to the left or to the right of u. After $O(\log n)$ iterations the binary search is narrowed down to an interval between two successive vertices in A. This implies that the point v_i from which the perpendicular to c has to be drawn is also determined.
  Note that for several successive vertices u_j in $[r, a]$ we can get the same vertex v_i as a result of computing $LCA_{T_s}(u_j, a)$. But the number of vertices in $[r, a]$ is $O(n)$ and so still after $O(\log n)$ iteration the binary search is narrowed down to an interval between two successive vertices in A.
- The case that $\theta_{m-1} < \pi/2$ is symmetric to the previous case.

In the end, we can compute the length of the shortest path in constant time from the information stored in T_s. If desired, the shortest path itself can be output in time linear in its length. This concludes the proof of the case $|Q| = 1$ of Theorem 1.

The following result can be obtained immediately from the above reasoning:

Corollary 1. *Let P be a simple polygon with n vertices and $s \in P$. We can compute a data structure of $O(n)$ size in $O(n)$ time to answer queries of the following type in $O(\log n)$ time: Given a query segment $S \subseteq P$, find the shortest distance one needs to travel in P from s to S. If desired, the actual shortest path can be reported in time linear in its length.*

4 The data structure for two query points

There are several cases for the optimal path p if $Q = \{q_1, q_2\}$. First we consider the most general case: The path p first reaches $w_1 = w(q_1)$ to view q_1 where it is *reflected* and then proceeds to $w_2 = w(q_2)$ to see q_2 (or vice versa), c.f. Fig. 6 for an example (we also assume that w_1 and w_2 do not intersect). The full discussion of all cases is given in Section 4.2.

The basic idea[1] behind computing p is to (conceptually) reflect the polygon $P_1 = P_s(q_1)$ at the window w_1. The resulting polygon P_1' is then 'glued' to P_1 along w_1 (and the polygon $P \setminus P_1$ is discarded) to form a (possibly self-overlapping) polygon P_1^*. We then compute the shortest path p^* in P_1^* from s to see q_2', the reflection of q_2 at w_2. To compute (the length of) p^* during a query, we (implicitly) determine the funnel F^* of q_2' in P_1^* and then compute the optimal point on the funnel base w_2' by binary search on its boundaries as in the previous section. We get F^* by combining the funnel $F_1 = F(q_1)$ and the hourglass $H_{12}' = H(w_1, w_2')$ using the technique of [4] (which is essentially a binary search). (The boundaries of) F_1 and H_{12}' (and thus of F^*) will be represented implicitly as paths in precomputed shortest path trees. Note that the reflection at w_1 is not performed explicitly, but 'on demand'.

4.1 Preprocessing phase

1. Compute A, $\mathcal{D}_1$, and $\mathcal{D}_2$ as in Section 3.1.
2. For each vertex v of P compute the shortest path tree T_v and preprocess it to support $O(1)$ time LCA-queries.

The total preprocessing time and space is $O(n^2)$.

4.2 Query phase

In the query phase we will first check if q_1 or q_2 is visible from s. This can be done in $O(\log n)$ time in the same way as in Section 3.2. If both query points are visible from s, the optimal path is the point s. If exactly one of the query points (say q_1) is visible from s the optimal path can be computed in $O(\log n)$ time as in Section 3.2[2]. In the following we will therefore assume that both query points are not visible from s.

As in Section 3.2 the windows $w_1 = a_1b_1$ and $w_2 = a_2b_2$ are computed in $O(\log n)$ time by using $\mathcal{D}_1$. Then there are several cases that may occur.

The windows intersect. In this case the intersection t is computed in $O(1)$ time. Next we identify the endpoint of w_1 which lies inside $P \setminus P_s(q_2)$ and the endpoint of w_2 which lies inside $P \setminus P_s(q_1)$ in $O(\log n)$ time, c.f. Fig. 3. Let

[1] This idea resembles the polygon folding technique of [2].

[2] To this end, we first have to identify the root r of the funnel F_{q_2} and then proceed as in Section 3.2 with T_r playing the role of T_s.

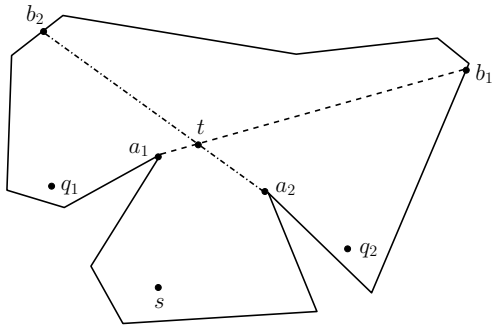

Fig. 3. The point b_1 lies inside $P \setminus P_s(q_2)$ and b_2 lies inside $P \setminus P_s(q_1)$.

$b_1 \in P \setminus P_s(q_2)$ and $b_2 \in P \setminus P_s(q_1)$. Then the optimal path we are looking for, is the shortest of the following three paths:

1. The shortest path from s to $\overline{b_1 t}$
2. The shortest path from s to $\overline{b_2 t}$
3. The shortest path from s that reaches $\overline{a_1 t}$ and $\overline{a_2 t}$

(1) and (2): This two paths can be computed in $O(\log n)$ time (using a variant[2] of Corollary 1).
(3): This path can be computed in the same way as in the general case where $\overline{a_1 t}$ has to be considered as w_1 and $\overline{a_2 t}$ as w_2.

One window lies 'behind' the other. If w_1 lies behind[3] (or *dominates*) w_2 (which can be checked in $O(\log n)$ time) we have to compute the shortest path from s to w_1 which can be done in $O(\log n)$ time (again using the variant[2] of Corollary 1). The case where w_2 dominates w_1 can be handled analogously.

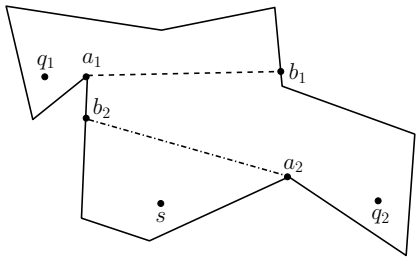

Fig. 4. The window w_1 dominates w_2.

[3] meaning that w_1 lies completely inside $P \setminus P_s(q_2)$

The general case. In this case no query point is visible from s, the windows do not intersect and no window dominates the other. We explain how the shortest path p_1 from s that first visits w_1 (where it is reflected) and then proceeds to w_2 is computed. The path p_2 which is the shortest path from s that first visits w_2 and then proceeds to w_1, can be computed analogously. At the end we determine which of these two paths is shorter; this is the optimal path p we are looking for.

Computing the funnel. To obtain the root r of the funnel F_{q_1} we compute the vertex $\hat{s}_1$ and the vertex $\hat{s}_2$ where $\hat{s}_1$ is the first vertex after s on $p(s, a_1)$ and $\hat{s}_2$ is the first vertex after s on $p(s, b_1)$. Either $\hat{s}_1 \neq \hat{s}_2$ then $r = s$, or $\hat{s}_1 = \hat{s}_2$ then $r = LCA_{T_{\hat{s}_1}}(a_1, b_1)$. In the following let $p(a_1, a_2)$ and $p(b_1, b_2)$ be the sides of the hourglass $H_{12} = H(w_1, w_2)$. We compute F^* by combining the funnel F_1 and the hourglass H'_{12} using the technique of [4]. To this end we need to construct the four common tangents $t_1, \ldots, t_4$ that touch one side from F_1 or H'_{12}. Since we cannot afford to reflect P_1 explicitly at w_1 during a query we compute F^* as the combination of F_1 with H_{12} instead. To this end the 'tangents' $t_1, \ldots, t_4$ have to be 'folded' at w_1, c.f. Fig. 5. Each tangent can be found in $O(\log n)$ time by performing a binary search on the vertices of $p(r, a_1)$, $p(r, b_1)$, $p(a_1, a_2)$ and $p(b_1, b_2)$ [10]. These vertices are not directly accessible but given only implicitly via the trees $T_{\hat{r}}$, $T_{\hat{a}_1}$, and $T_{\hat{b}_1}$ (we can compute $\hat{r}$, $\hat{a}_1$ and $\hat{b}_1$ using $\mathcal{D}_1$ in $O(\log n)$ time). Therefore we have to perform the binary search on the array A instead (as in the previous section).

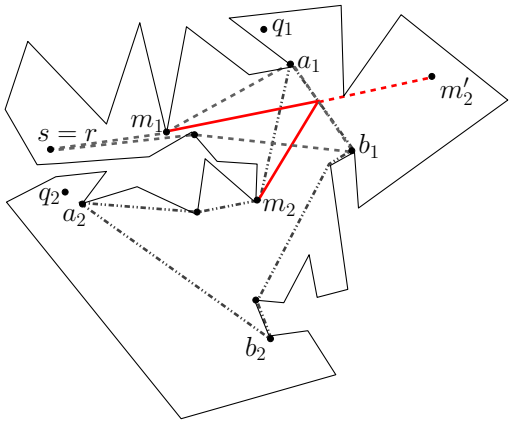

Fig. 5. To obtain m'_2 we reflect m_2 on w_1. We construct the line l_1 between m'_2 and m_1 and reflect l_1 on w_1. Let $m_1 m_2^*$ be the line segment between m_1 and m_2 which is folded at w_1. If l_1 is tangent to $p(r, a_1)$ and the reflection of l_1 is tangent to $p(a_1, a_2)$, $m_1 m_2^*$ is the tangent to $p(r, a_1)$ and $p(a_1, a_2)$.

Let the vertex r^* be the root of F^*. Two cases can occur: Either $r^* \in F_1$ or $r^* \in H_{12}$.

In the example depicted in Fig. 6 we have that $r^* \in F^*$ and the sides of F^* are $p(r^*, a_2)' = p(r^*, m_1) + m_1 m_2^* + p(m_2, a_2)$ and $p(r^*, b_2)' = p(r^*, m_3) + m_3 m_4^* + p(m_4, b_2)$ where m_1 is the last point on $p(r, a_1)$ which belongs also to a side of F^*, m_2 is the first point on $p(a_1, a_2)$ which belongs also to a side of F^* and $m_1 m_2^*$ is the line segement between m_1 and m_2 which is folded at w_1 (clearly $|m_1 m_2^*| = |m_1 m_2'|$ where m_2' is the reflection of m_2 at w_1). The same holds for m_3, m_4 and $m_3 m_4^*$. The points m_1, m_2, m_3 and m_4 are the points on which F_1 and H_{12} are connected, so they can be obtained in the same way as in [4] (note that either $r^* = m_1$, $r^* = m_3$ or $r^* = r$).

If $r^* \in H_{12}$ the sides of F^* are $p(r^*, a_2)$ and $p(r^*, b_2)$ (note that $r^* = m_3 = m_4$).

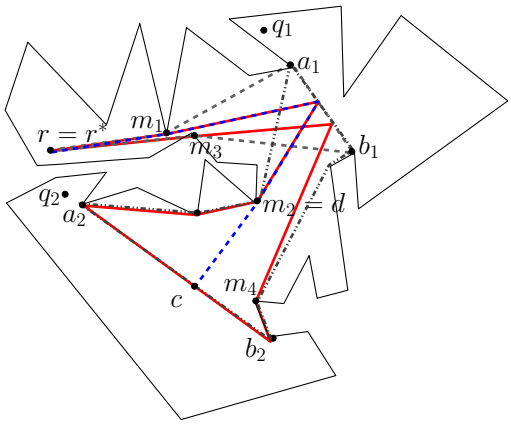

Fig. 6. The drawing shows F^* and the optimal path p from s to view q_1 and q_2.

Computing the optimal point on the window. If $r^* \in H_{12}$, the optimal point c on w_2 can be computed in the same way as in Section 3.2. If $r^* \in F_1$, we additionally have to take the following issues into account:

- If $r^* = s$ and s is not a vertex of P we use $T_{\hat{s}}$ instead of T_s where $\hat{s}$ is the first vertex after s on $p(s, m_1)$ or on $p(s, m_3)$.
- We have to reflect the extension of an edge $v_i v_{i+1}$ in $O(1)$ time on the window w_1 for $v_i v_{i+1} \subseteq p(r^*, m_1)$ or $v_i v_{i+1} \subseteq p(r^*, m_3)$ if we want to compute the angle θ_i.
- If we want to perform a binary search on $p(r^*, b_1)'$ we first have to look at the angle between w_1 and the extension edges incident to m_2. Therefore we have to reflect m_1 on w_1. Via this angle we can decide if we have to perform

a binary search on $p(r^*, m_1)$ or $p(m_2, b_1)$. Note that, if we perform a binary search on $p(m_2, b_1)$, i.e., on $[m_2, b_1]$, we compute $LCA_{T_{m_2}}(u, v_1)$ for a vertex $u \in [m_2, b_1]$.
- If we want to perform a binary search on $p(r^*, b_2)'$ we proceed analogously.

In the end, we can compute the length of the shortest path in constant time from the information stored in the precomputed shortest path trees. E.g., if c is the foot of the perpendicular from $d \in p(m_2, a_2)$, the length of the shortest path is the sum of the lengths of $p(s, m_1)$, $m_1 m_2^*$, $p(m_2, d)$ and dc. If desired, the shortest path itself can be output in time linear in its length.

4.3 Approximate shortest inspection-path queries

We now turn to the proof of Theorem 2. The data structure of Theorem 1 for the case $|Q| = 1$ can be used to answer 3-approximate shortest inspection-path queries for $\{q_1, q_2\} \subseteq P$ as follows: We query the structure to compute (the length of the) path p_1 which is the shortest path from s to view q_1, as well as the path p_2 which is the shortest path from s to view q_2. This requires $O(\log n)$ time. If we walk from s along p_1 then back to s along p_1 and then along p_2 we get the (non-simple) path $\tilde{p}$ which is clearly an inspection path for $\{q_1, q_2\}$. Since the shortest path p_{opt} from s to view q_1 and q_2 is an inspection path for q_1 as well as an inspection path for q_2 we have that $|p_1| \leq |p_{\mathsf{opt}}|$ and $|p_2| \leq |p_{\mathsf{opt}}|$. It follows that $|p_{\mathsf{opt}}| \leq |\tilde{p}| = 2|p_1| + |p_2| \leq 3|p_{\mathsf{opt}}|$, so $\tilde{p}$ is a 3-approximate answer to the shortest inspection-path query for $\{q_1, q_2\}$.

In order to answer $(1+\epsilon)$-approximate shortest inspection-path queries for $\{q_1, q_2\}$ and $\epsilon > 0$ we proceed as follows (the approach is illustrated in Fig. 7):

1. Compute the length of the path $\tilde{p}$ as just described in $O(\log n)$ time.
2. Compute the windows $w_1 = w(q_1)$ and $w_2 = w(q_2)$ as above in $O(\log n)$ time.
3. For $i = 1, 2$ compute $\tilde{w}_i$, the intersection of w_i with the disc of radius $2|\tilde{p}|$ around s. Clearly for any path p' from s that visits points in $w_i \setminus \tilde{w}_i$ we have that $|p'| \geq 2|\tilde{p}| \geq 2|p_{\mathsf{opt}}|$, so p_{opt} can only visit points in $\tilde{w}_i$. By construction $|\tilde{w}_i| \leq 4|\tilde{p}|$.
4. Let $\delta = \epsilon|\tilde{p}|/3$ and compute for $i = 1, 2$ the set $\tilde{W}_i$ of equally spaced points at distance $\delta/3$ on $\tilde{w}_i$ (including the endpoints of $\tilde{w}_i$). The number of points in $\tilde{W}_i$ is $n_i \leq 2 + \lceil \frac{|\tilde{w}_i|}{\delta/3} \rceil = 2 + \lceil \frac{4|\tilde{p}|}{\epsilon|\tilde{p}|/9} \rceil = O(1/\epsilon)$. Observe that the shortest inspection path $\hat{p}$ from s to view $\{q_1, q_2\}$ that is restricted to visit w_i only at the points of $\tilde{W}_i$ has length at most $|p_{\mathsf{opt}}| + \delta = |p_{\mathsf{opt}}| + \epsilon\frac{|\tilde{p}|}{3} \leq (1+\epsilon)|p_{\mathsf{opt}}|$.
5. For each pair of points $\{u, v\}$ with $u \in \tilde{W}_1, v \in \tilde{W}_2$ compute the shortest path from s that visits u and v in $O(\log n)$ time (using the structure of [4]) and return $\hat{p}$ as the shortest such path. The total time for this step is $O(n_1 n_2 \log n) = O(\frac{1}{\epsilon^2} \log n)$.

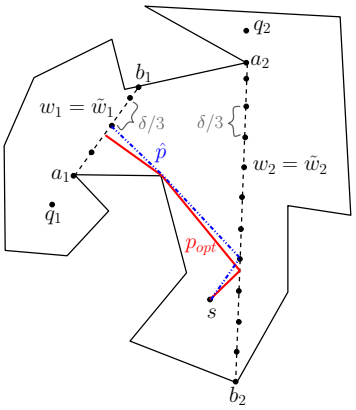

Fig. 7. This drawing shows the sets $\tilde{W}_1$ and $\tilde{W}_2$ as well as the paths $\hat{p}$ and p_{opt}.

5 Conclusion

For a simple n-vertex polygon P with a designated starting point s and a given set $Q \subseteq P$ of query points we described data structures to find the shortest path from s to see all points in Q. For $|Q| = 1$ our approach yields a structure of linear size that can be computed in linear time and achieves logarithmic query time. This significantly improves previous work on the subject [9]. Our approach also seems to be conceptually simpler. For $|Q| = 2$ our approach yields a structure of quadratic size that can be computed in quadratic time and achieves also logarithmic query time, even if the source point is not fixed. We also described a data structure with linear size and preprocessing time that can answer for any $\epsilon > 0$ a $(1+\epsilon)$-approximate query in $O(\frac{1}{\epsilon^2} \log n)$ time.

It remains unclear if the query problem can (efficiently) be generalized to more than two query points, and if it can still be solved efficiently if we consider the case of polygons with holes. As mentioned above the status of the off-line problem in simple polygons is also unknown.

References

1. S. Carlsson and H. Jonsson. Computing a shortest watchman path in a simple polygon in polynomial time. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 122–134. Springer-Verlag, 1995.
2. W. Chin and S. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28:39–44, 1988.
3. S. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, New York, NY, USA, 2007.
4. L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, Oct. 1989.

5. 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*, 2:209–233, 1987.
6. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
7. J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.
8. R. Khosravi and M. Ghodsi. Shortest paths in simple polygons with polygon-meet constraints. *Inf. Process. Lett.*, 91(4):171–176, 2004.
9. R. Khosravi and M. Ghodsi. The fastest way to view a query point in simple polygons. In *Abstracts of the 21st European Workshop on Computational Geometry (EWCG), Eindhoven, Netherlands*, pages 187–190, 2005.
10. M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.