Maintaining Visibility of a Polygon with a Moving Point of View*

Danny Z. Chen† *Ovidiu Daescu*‡

Abstract

The following problem is studied in this paper: Given a scene with an n-vertex simple polygon and a trajectory path in the plane, construct a data structure for reporting the perspective view from a moving point along the trajectory. We present conceptually simple algorithms for the cases of this problem in which the trajectory path consists of several line segments or of a conic curve that contains the polygon. Our algorithms take $O(n \log n)$ time and $O(n)$ space. We also prove that the problem of reporting perspective views from successive points along a trajectory path takes $\Omega(n \log n)$ time in the worst case in the algebraic computation tree model. Our data structure reports the view from any query point on the trajectory in $O(k + \log n)$ time for a view of size k.

1 Introduction

In this paper, we study the following problem: Given a scene with an n-vertex simple polygon P and a trajectory path in the plane, report the perspective view from each of a sequence of successive points on the trajectory. We present conceptually simple and optimal algorithms for the cases of this problem in which the trajectory path consists of several line segments or of a conic curve that contains the polygon. Intuitively, the polygon P could represent the "opaque" walls of a building.

The general problem of computing visibility information of a geometric scene along a trajectory arises in several application areas, such as computer animation, computer vision, image compression, and flight simulation. Bern *et al.* [2] and Mulmuley [11] studied a case of this problem in which the trajectory is a straight line and the scene consists of a set of polygons in the 3-dimensional (3-D) space. Lenhof and Smid [10] considered a related problem of computing the visibility map for a 3-D scene consisting of disjoint spheres from a moving point on a circle at infinity.

Note that the visibility from points along a trajectory path can be captured by a sequence of "similar" views in the sense that one view differs topologically from the next view only slightly. That is, at certain points along the path, topology changes occur to the view (e.g., a newly seen object emerges in the view or a previously visible object disappears from the view). We call these points on the trajectory its *critical points*. The solutions of Bern *et al.* [2] and Mulmuley [11], for a more complicated 3-D version of the problem, use persistent data structures [5, 13] to capture the topological changes of the visibility from points along the trajectory. However, their data structures support only ray shooting queries and it is not immediately clear how these data structures can be modified to report the complete view from a query point in an output-sensitive manner.

We also use a persistent data structure to maintain visibility from points along a trajectory for a simpler (planar) scene. Our data structure requires $O(n \log n)$ time and $O(n)$ space to construct. This data structure reports the view from any query point on the trajectory in an output-sensitive fashion (i.e., in $O(k + \log n)$ time for a view of size k). Further, we prove that reporting all critical points in the order along the trajectory (at which topological changes of the visibility occur) requires $\Omega(n \log n)$ time in the worst case in the algebraic computation tree model. Hence we solve optimally the problem of reporting the view from a moving point along a trajectory path. Our algorithms are quite simple conceptually.

Note that the visibility of a moving point along a trajectory path is closely related to the weak visibility of the path. With the boundary of the polygon P being the only "opaque" object, a point p is said to be *weakly visible* from the trajectory if and only if p is visible from some point (depending on p) on the trajectory. Many algorithms have been developed for computing the weak visibility of a polygon from a line segment [4, 6–9, 14]. Our solutions will make substantial use of several structures of the weak visibility of P from the trajectory.

The rest of the paper consists of 3 sections. Section 2

*This research was supported in part by the National Science Foundation under Grant CCR-9623585.

†Dept. of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. chen@cse.nd.edu.

‡Dept. of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. odaescu@bach.helios.nd.edu.

reviews some notations and preliminary results needed by our algorithms. Sections 3 describes in detail our algorithm for the case with the trajectory path being a line segment inside P. Section 4 extends our solution to several other cases of the problem.

2 Preliminary

Let P be the input simple polygon, specified by the list of vertices $V = (v_1, v_2, \ldots, v_n)$ along the boundary $bd(P)$ of P clockwise. Without loss of generality (WLOG), we assume that no three distinct vertices of P are collinear. We denote the x (resp., y) coordinate of a point p in the plane by $x(p)$ (resp., $y(p)$).

It is well known that a triangulation of P can be obtained in linear time [3]. In [7], it was proved that the weakly visible polygon of a triangulated polygon P from a line segment l that does not cross the boundary $bd(P)$ of P can be computed in linear time. We will denote the weakly visible polygon of P from l as $Vis(P,l)$. Consider the case in which the segment l is the input trajectory. Observe that for each point x on l, the visible portion of P from x, denoted by $Vis(P,x)$, is completely contained in $Vis(P,l)$ (i.e., $Vis(P,x) \subseteq Vis(P,l)$). Hence the visibility of a moving point along the segment l is concerned only with points in $Vis(P,l)$. WLOG, we assume that for a segment l in P, we have computed $Vis(P,l)$ (in linear time). From now on, our discussion will focus on $Vis(P,l)$. We denote $Vis(P,l)$ still by P and denote the number of vertices of P as $|P| = n$.

Consider $Vis(P,x)$ for a given point $x \in l$. $Vis(P,x)$ is said to be *star-shaped* and it is well known that when traversing the boundary of $Vis(P,x)$, its vertices and edges are visited in the sorted order by their polar angles with respect to x [12]. The edges e of $Vis(P,x)$ can be classified into two types: (i) e is either part of an edge of P, or (ii) e is not on $bd(P)$ except its end vertices. We call the edges of the first type *real edges* (because they are part of some edges of P), and the edges of the second type *false edges*. It is easy to see that the vertices of $Vis(P,x)$ are all on $bd(P)$.

Let l be the line segment from which weak visibility of the input polygon was computed, and let a and b be the endpoints of l. Suppose a point x traverses along l from a to b. At certain "moments" of this traversing, the visibility of P from the point x, $Vis(P,x)$, may change its topology as some edge of $Vis(P,x)$ becomes no longer visible from x or a "new" edge becomes visible from x. These special "moments" correspond to certain points on l, called *critical points*. One of our main tasks is to characterize the critical points on l. This characterization is done by exploiting the relation between a weakly visible polygon and certain shortest paths inside it.

It is known [1] that the shortest path tree inside P rooted at an end point s of l can be computed in linear time (since P is weakly visible from l). The algorithm for computing such a shortest path tree is based on the outward convexity of the shortest paths to s whose vertices are also vertices of P. Figure 1 gives an example of such an outward convex path from v to b.

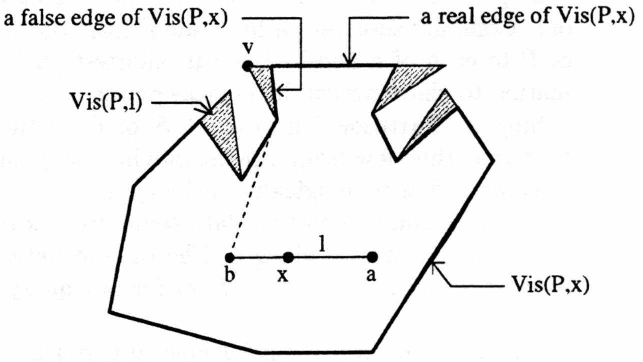

Figure 1: Illustrating $Vis(P,l)$, $Vis(P,x)$, and a shortest v-to-b path in P.

To maintain the visibility information for points on the segment l, we make use of persistent data structures [5,13]. Ordinary data structures are "ephemeral" because any change to the structure destroys the old version. In contrast, a "persistent" data structure allows accesses to the current version as well as to the old versions that had occurred in a past time. A time in our problem corresponds to a point on the segment l visited during the traversing of l. Techniques for designing persistent versions of binary search trees that support logarithmic time operations of search, insertion, and deletion were presented in [5,13]. In particular, a persistent red-black tree was used in [13] to maintain a sorted list. Let m be the total number of update operations on the list. The persistent data structure in [13] implements update and search queries (even in the past) in $O(\log m)$ time each. The space used by this data structure is $O(1)$ per update operation amortizedly (and hence $O(m)$ total space in the worst case). We will reduce the computation for maintaining the visibility from points along the segment l to a sequence of $O(n)$ update operations (i.e., insertion and deletion) that can be handled by the data structure in [13].

3 The Basic Algorithm

In this section, we present an algorithm for the basic case with the trajectory path being the line segment l inside P. This algorithm, nevertheless, illustrates our key ideas. The algorithm takes $O(n \log n)$ time and

$O(n)$ space to build the persistent data structure for our visibility problem. We also prove that sorting all critical points in the order along the trajectory requires $\Omega(n \log n)$ time in the worst case in the algebraic computation tree model. This implies that we solve optimally the problem of reporting the view from a moving point along a trajectory path. Our algorithm consists of the following three steps:

Step 1: Let a and b be the starting and ending points of l. Compute shortest paths inside P from the vertices of P to each of a and b. Use this shortest path information to characterize the critical points of l.

Step 2: Partition l into a set S of $O(n)$ intervals such that the view from a point moving along such an interval of S is topologically unchanged.

Step 3: Build a persistent data structure to store the visibility information along l. The data structure will support a fast report of $Vis(P, x)$ for any query point $x \in l$.

We first show in Section 3.1 how to perform Steps 1 and 2. In Section 3.2, we present the persistent data structure that stores the visibility information along l (Step 3). We prove in Section 3.3 an $\Omega(n\log n)$ worst case lower bound in the algebraic computation tree model for sorting the critical points along l.

3.1 Computing the Topological Changes in Visibility

Since we assumed (WLOG) that the polygon P is weakly visible from the segment l, the shortest paths in P from all vertices of P to the endpoints of l can be computed in $O(n)$ time [1]. Hence we only need to discuss the characterization of the critical points of l. We will first show how to identify the critical points on l and how to use them to obtain a partition S of l. We will then prove that $\Omega(n\log n)$ is a worst case lower bound in the algebraic computation tree model for sorting the critical points along l.

Let x be a point moving on l by starting at its end point a and consider how $Vis(P, x)$ changes correspondingly (see Figure 2). Recall that the boundary $bd(Vis(P, x))$ consists of real edges and false edges, as defined in Section 2. Suppose that when x is moving towards a position $x' \in l$, the topology of $Vis(P, x)$ remains unchanged. But nevertheless, other changes can occur to some real and false edges of $Vis(P, x)$ during this movement of x to x', as follows. The changes to a real edge e of $Vis(P, x)$ may be that e becomes longer or shorter, and one or even both of its end vertices change their actual locations on $bd(P)$. Note that a false edge e^* of $Vis(P, x)$ is always collinear with x, and that the end vertex of e^* that is closer to x than the other end vertex of e^*, called the *closer vertex* $u_c(e^*)$ of e^*, is always a vertex of P. The change to a false edge e^* of $Vis(P, x)$ may be that the end vertex of e^*

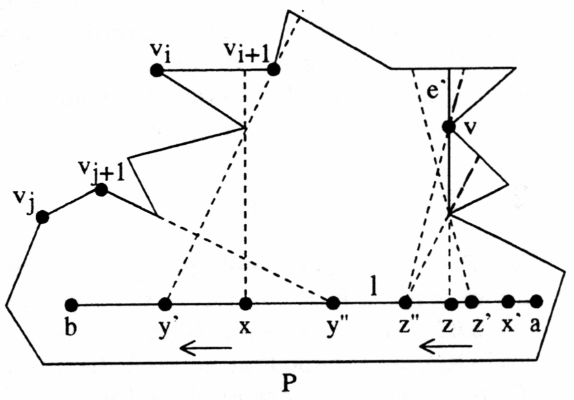

Figure 2: Illustrating various types of critical points on the trajectory l.

that is not $u_c(e^*)$, called the *farther vertex* $u_f(e^*)$ of e^*, changes its actual location on $bd(P)$ (the location of $u_c(e^*)$ remains the same). The changes to each real edge e are in fact caused by the changes to one or two false edges of $Vis(P, x)$ whose farther vertex defines an end vertex of e.

Now consider a real edge e and a false edge e' of $Vis(P, x)$ such that the farther vertex $u_f(e')$ of e' defines an end vertex of e. Let e be contained by the edge $e_i = \overline{v_i v_{i+1}}$ of P and WLOG let the farther vertex $u_f(e')$ of e' falls in the interior of e_i. Suppose the point x passes x' and continues moving to the end point b of l. Then $u_f(e')$ also continues to move towards the vertex v_{i+1} of P, maybe until the edge e' encounters a vertex v of P. There are two possible cases for e' encountering a vertex v of P: (i) $v = v_{i+1}$, and (ii) $v \neq v_{i+1}$. When $v \neq v_{i+1}$, v must occur in the interior of e'.

In case (i) (i.e., $v = v_{i+1}$), a topological change may (but does not have to) occur to $Vis(P, x)$ when $u_f(e')$ is at v_{i+1}. This is the situation if at this "moment" of x, the real edge e "shrinks" into a single vertex of $Vis(P, x)$. An example of this situation is given in Figure 2 for x being at the position y'. Furthermore, if the vertex v_{i+1} of P was also the closer vertex of another false edge e'' of $Vis(P, x)$ just before x moved onto the position y', then when x is at y', the two false edges e' and e'' of $Vis(P, x)$ "merge" into a single false edge of $Vis(P, x)$, also a topological change to $Vis(P, x)$. If the real edge e does not shrink into a single vertex of $Vis(P, x)$, then when x continues to move such that $u_f(e')$ passes v_{i+1}, a topological change occurs to $Vis(P, x)$. This is the situation when a "new" real edge that begins at v_{i+1} appears in $Vis(P, x)$. An example of this situation is given in Figure 2 for x being at the position y'' (illustrated with the edge $e_j = \overline{v_j v_{j+1}}$ of P).

In case (ii) (i.e., $v \neq v_{i+1}$), the false edge e' of $Vis(P,x)$ is "split" into two false edges of $Vis(P,x)$ (one has v as its closer vertex), a topological change to $Vis(P,x)$. An example of this situation is given in Figure 2 for x being at the position z (e.g., the false edge of $Vis(P,x)$ at the position z' results into two false edges at z''). If x continues to move and passes z, then a "new" real edge of $Vis(P,x)$ that begins with v emerges in $Vis(P,x)$, also a topological change to $Vis(P,x)$.

From the above discussion, it is clear that a topological change to $Vis(P,x)$ occurs when the point x becomes collinear with and is visible from two different vertices v and w of P simultaneously such that v and w are on the same side of l. The following lemmas summarize the useful geometric structures of this situation.

Lemma 1 *Let a point x be moving along the segment l. A topological change to $Vis(P,x)$ occurs or is about to occur if and only if x is at a position that is collinear with and is visible from two different vertices v and w of P simultaneously such that v and w are on the same side of l. Such a position is a* critical point *of l.*

Proof. Followed from the above discussion. ■

Lemma 2 *Let a and b be the two end points of l. Let v and w be two different vertices of P such that (1) they are on the same side of l, and (2) there is a critical point x of l that is collinear with and is visible to both of them simultaneously. Then the segment $\overline{vw}$ is an edge on a shortest path from one of v and w to one of a and b in P.*

Proof. Easy and omitted. ■

The characterization of the critical points of l follows from Lemmas 1 and 2. To identify the critical points, we do the following. For each vertex v of P, let $vv(a)$ (resp., $vv(b)$) be the first edge on the v-to-a (resp., v-to-b) shortest path inside P. Let the ray originating from v and along $vv(a)$ (resp., $vv(b)$) cross the segment l at a point $c_v(a)$ (resp., $c_v(b)$). Then when a point x travels along l from a to b, $c_v(a)$ (resp., $c_v(b)$) is the first (resp., last) position at which v is visible from x, and v is visible from x while x is in the interval $\overline{c_v(a)c_v(b)}$ on l. The set C of critical points of l is then $\{c_v(a), c_v(b) \mid v \text{ is a vertex of } P\}$. It is clear that $|C| = O(n)$ and that given the shortest paths from the vertices of P to a and b, C can be easily obtained in $O(n)$ time.

The set S of intervals on l is obtained by first sorting in $O(n\log n)$ time the points of C along l and then using these sorted points to divide l into intervals. If several critical points coincide at some position of l, then we break the tie arbitrarily and let the interval between two such points be of zero length.

It is easy to see that the computation in this subsection takes $O(n \log n)$ time and $O(n)$ space.

3.2 Building the Persistent Data Structure

There are two facts that enable us to use the persistent data structure of [13] to maintain visibility information from points along the segment l.

1. The order of the vertices and edges of $Vis(P,x)$ based on their polar angles with respect to any point $x \in l$ is consistent with the order of the vertices and edges of P along $bd(P)$.

2. Each critical point in C can be associated with $O(1)$ operations on $Vis(P,x)$ such as "delete an edge" (e.g., when a real edge shrinks into a vertex or two false edges merge into one) and "insert an edge" (e.g., when a "new" real edge emerges or a false edge is "split" into two false edges). For example, when an existing false edge e' is "split" into two false edges because a vertex v of P touches the interior of e' (with one of the two false edges having v as its closer vertex), what we can actually do is the following $O(1)$ operations on $Vis(P,x)$: (a) delete e' (since its farther vertex changes), (b) insert each of the two "new" false edges, and (3) insert a "new" real edge that begins at the vertex v of P. If we associate the two collinear vertices v and w of P with their corresponding critical point p on l, then it is easy to decide what operations to perform on $Vis(P,p)$ based on the position of p and the edges of P adjacent to each of v and w.

From the above facts, we maintain $Vis(P,x)$ as a (circular) list of real and false edges sorted by their polar angles with respect to x. In the list $Vis(P,x)$, we represent each real edge e by the "name" e_i if e is contained by the edge e_i of P, and represent each false edge by its closer vertex. The following facts are also useful:

- In the sorted list $Vis(P,x)$, two different false edges cannot be consecutive to each other unless they are collinear. If two real edges are consecutive to each other in the list, then the two edges are also adjacent to each other on $bd(P)$ (and hence their common vertex in P belongs to $Vis(P,x)$).

The above facts imply that there is a vertex of P in every (at most) two edges of $Vis(P,x)$. These vertices of P in $Vis(P,x)$ enable us to perform a binary search in $Vis(P,x)$ based on a polar angle with respect to x. This property also enables us to compute, in $O(|Vis(P,x)|)$ time, the actual edges of $Vis(P,x)$ once we are given these vertices of P that are on $Vis(P,x)$, the ordered list $Vis(P,x)$ containing the "names" of its edges, and the position of x.

We are now ready to describe the construction of our persistent data structure based on the technique of [13]. First we compute $Vis(P, a)$ from the starting point a of l; this takes $O(n)$ time [12]. Then using the sorted sequence C of critical points along l, we perform appropriate insertion/deletion operations on $Vis(P, x)$ at each critical point, corresponding to the traveling of a point x along l. The insertion/deletion of an edge in $Vis(P, x)$ is guided by binary search based on the angular information of an appropriate vertex of P with respect to x.

Our construction of the persistent data structure requires $O(n\log n)$ time and $O(n)$ space, because totally $O(n)$ update operations are done (for $O(n)$ critical points, each causing $O(1)$ such operations). This data structure enables us to do binary search in $Vis(P, x)$ (based on angular information) in $O(\log n)$ time. In fact, it allows us to compute all edges of $Vis(P, x)$ between two polar angles with respect to x in $O(k+\log n)$ time, where k is the number of edges reported [13]. Hence with this data structure, we can report $Vis(P, x)$ for any query point $x \in l$ in an output-sensitive $O(|Vis(P, x)| + \log n)$ time.

3.3 Lower Bound of Sorting the Critical Points along a Trajectory

Since P is a weakly visible polygon from the segment l and since the critical points of l are obtained by "projecting" onto l the vertices of P along their first edges on the shortest paths to the end points a and b of l, one could suspect that it might be possible to sort the set of critical points along l in less than $O(n\log n)$ time. But the following lemma shows that this is not the case.

Lemma 3 *The problem of reporting all the critical points in their order along the segment l requires $\Omega(n\log n)$ time in the worst case in the algebraic computation tree model.*

Proof. The lemma is proved by reducing, in linear time, the problem of sorting arbitrary integers to that of reporting all the critical points in their order along l. Note that from Yao's results in [15], sorting n integers in an arbitrary range requires $\Omega(n\log n)$ time in the worst case in the algebraic computation tree model. Further, note that it is easy to reduce, in linear time, the problem of sorting n *arbitrary* integers to the case of sorting n *positive* integers in an arbitrary range. Actually, we reduce the problem of sorting n positive integers (in an arbitrary range) to that of reporting the critical points along l. Also, it is sufficient to consider only those critical points that are generated by using the shortest paths in P from the vertices of P to the *starting* end point a of l.

Our reduction works as follows. Given a set A_I of n arbitrary positive integers $a_1, a_2, \ldots, a_n$, we first create a polygonal chain $P_I = (w_1, w_2, \ldots, w_{2n})$ of $2n$ vertices that is monotone to the x-axis. The vertices w_{2i-1} of P_I are at points $(i, n-i+1)$ in the plane, $i = 1, 2, \ldots, n$. Each vertex w_{2i} of P_I is at the point in the first quadrant of the point w_{2i-1} such that the length of the edge $\overline{w_{2i-1}w_{2i}}$ of P_I is 0.5 and the slope of $\overline{w_{2i-1}w_{2i}}$ is a_i (see Figure 3). P_I so obtained is clearly monotone to the

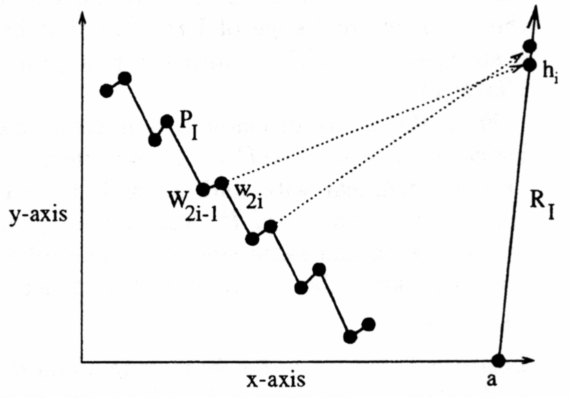

Figure 3: Reducing the integer sorting problem to that of sorting critical points along R_I.

x axis. Now consider an almost vertical ray R_I (slant slightly to the right) that is to the right of P_I and that originates from a point a on the x-axis (we will choose a later). Observe that P_I is weakly visible from R_I (provided that R_I is long enough) since the ray from every vertex w_{2i-1} and along the edge $\overline{w_{2i-1}w_{2i}}$ of P_I hits a point h_i on R_I. Also, note that h_i is the critical point on R_I for the vertex w_{2i-1} of P_I generated by the edge $\overline{w_{2i-1}w_{2i}}$ on the shortest w_{2i-1}-to-a path that is inside the region enclosed by R_I and P_I. Observe that if R_I is sufficiently far to the right of P_I (because of its starting point a), then the following will hold: For any $i, j \in \{1, 2, \ldots, n\}$ with $i < j$, (i) $a_i \geq a_j$ if and only if $y(h_i) > y(h_j)$, and (ii) $a_i < a_j$ if and only if $y(h_i) < y(h_j)$. But the sorted order of the critical points h_i based on their y coordinates is the same as their order along R_I. Therefore, for an appropriate ray R_I, the sorted sequence of A_I can be obtained in $O(n)$ time from the sorted sequence of the critical points h_i (for the vertices w_{2i-1} of P_I) along R_I.

The only thing left is to show how to choose the ray R_I (i.e., choose the starting point a for R_I). Find the smallest integer in A_I; let it be a_i. Let L be the line passing the vertex of P_I at the point $(n, 1)$ and with slope a_i, and L' be the line passing the vertex at the point $(1, n)$ and with slope $a_i - 1$ ($a_i - 1$ is the largest integer that is smaller than a_i). Then the intersection point of the ray from each vertex w_{2k-1} of P_I and along the edge $\overline{w_{2k-1}w_{2k}}$ with the ray from any other vertex w_{2j-1} and along the edge $\overline{w_{2j-1}w_{2j}}$ is to the left of the intersection point of L and L'. The starting point a of R_I can be obtained by finding the intersection point of

L and L' and then projecting this point vertically onto the x axis. Note that it is easy to construct a weakly visible polygon P from R_I with the chain P_I as part of $bd(P)$. This finishes the linear time reduction. ∎

The above lemma immediately implies that the problem of reporting all critical points in the order along a trajectory path requires $\Omega(n \log n)$ time in the worst case in the algebraic computation tree model. Since we can use the $O(n\log n)$ time algorithm in this section to solve the case of the problem with the trajectory being a line segment, our solution for this case is optimal.

4 Extensions

We can easily extend our algorithm in the previous section to several other cases of maintaining visibility information of a polygon P along a trajectory path. Note that in this section, we *do not* assume that P is weakly visible from the trajectory path.

The first case is that the trajectory path consists of $h > 1$ line segments inside the polygon P. In this case, a direct application of our algorithm in Section 3 to each of the h segments on the path results in a solution that requires $O(hn\log n)$ time and $O(hn)$ space. The case in which the h-segment trajectory path is outside P can also be handled in a similar way.

Another case is that the trajectory path is a conic curve that contains the polygon P. We sketch below only the algorithm for the case in which the trajectory path is a circle C containing P (other conic curves can be processed in a similar manner).

We first compute the convex hull $CH(P)$ of P in linear time [12]. Note that the circle C contains $CH(P)$ because C contains P. Now consider each connected region that is enclosed between the convex hull $CH(P)$ and $bd(P)$. Let R be such a region. Then R forms a simple polygon and the boundary $bd(R)$ of R consists of exactly one edge from $CH(P)$ and the rest of edges from $bd(P)$. We denote the edge of R from $CH(P)$ by $e(R)$. Note that each edge of $bd(P)$ belongs to at most one such region. We next compute, for each such region R, its weakly visible portion $Vis(R, e(R))$ from the edge $e(R)$. Then we apply our algorithm in Section 3 to $Vis(R, e(R))$ and $e(R)$ for each region R, with one modification. The modification is that instead of obtaining critical points by "projecting" vertices of $Vis(R, e(R))$ onto the edge $e(R)$, the "projections" are onto C. It is easy to show that there are $O(n)$ critical points on C (this is the same as the case with the trajectory being a segment). Then the rest of the algorithm follows as in Section 3. The complexity bounds of the algorithm for the conic curve case are $O(n\log n)$ time and $O(n)$ space.

References

[1] D. Avis and G.T. Toussaint. An optimal algorithm for determining the visibility polygon from an edge. *IEEE Trans. Comput.*, pages 910–914, C-30 (12), 1981.

[2] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. *Algorithmica*, 11:360–378, 1994.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, pages 485–524, 1991.

[4] B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4:551–581, 1989.

[5] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Communications of the ACM*, 5:109–121, 1986.

[6] H. ElGindy. Hierarchical decomposition of polygon with applications. Ph.D. thesis, McGill University, 1985.

[7] L. Guibas, J. Hershberger, D. Leven, M.E. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple poligons. *Algorithmica*, 2:209–233, 1987.

[8] P.J. Heffernan and J.S.B. Mitchell. Structured visibility profiles with applications to problems in simple polygons. *Proc. 6th Anual ACM Symp. Computational Geometry*, pages 53–62, 1990.

[9] D.T. Lee and A.K. Lin. Computing the visibility polygon from an edge. *Computer Vision, Graphics, and Image Processing*, 34:1–19, 1986.

[10] H.P. Lenhof and M. Smid. Maintaining the visibility map of spheres while moving the viewpoint on a circle at infinity. *MPI-I*, pages 92–102, 1992.

[11] K. Mulmuley. Hidden surface removal with respect to a moving view point. *Proc. 23rd ACM Symp. on Theory of Computing*, pages 512–522, 1991.

[12] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[13] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[14] G.T. Toussaint. A linear-time algorithm for solving the strong hidden-line problem in simple polygon. *Pattern Recognition letters*, 4:449–451, 1986.

[15] A.C.-C. Yao. Lower bounds for algebraic computation trees with integer inputs. *SIAM Journal on Computing*, pages 655–668, 1991.