Maintaining Visibility Information of Planar Point Sets with a Moving Viewpoint

Olivier Devillers* Vida Dujmović† Hazel Everett‡ Samuel Hornus§ Sue Whitesides¶ Steve Wismath‖

Abstract

Given a set of n points in the plane, we consider the problem of computing the circular ordering of the points about a viewpoint q and efficiently maintaining this ordering information as q moves.

1 Introduction

Let $P = \{p_0, p_1, \ldots, p_{n-1}\}$ be a set of points in the plane. A point p is *visible* from a viewpoint q if the line segment pq contains no other point of P. If p coincides with q then p is not visible. The *view* of P from a viewpoint q is the clockwise circular ordering of all points in P that are visible from q. Given P, the view from q is easily computed in $O(n \log n)$ time.

In this paper we are interested in maintaining the view when q moves. More precisely, we would like to answer the following *moving viewpoint* queries efficiently: Given a trajectory σ, report, as q moves along σ, all of the different views in the order that they are seen by q. We assume that σ is given *on-line*. We don't know ahead of time the entire trajectory; we would like to compute the new views as the trajectory becomes known to us. For simplicity, here we assume the trajectory is a single segment σ, that does not pass through any points of P, and we assume no three points of P are collinear.

This problem is motivated by the graphics problem of maintaining the view during an interactive walk-through of a 3D scene. Here we give an efficient solution for a 2D scene consisting of points.

Our main result is:

Theorem 1 *Let P be a set of n points in the plane. After $O(n \log n)$ preprocessing time and using $O(n)$ space, moving viewpoint queries along a specified line segment σ can be answered in $O(k \log n + s)$ amortized or $O(k \log^2 n + s)$ worst-case time where k is the number of different views seen from σ and s is the size of the output.*

Notice that, if each time the view changes the entire view is output, then $s = \Theta(kn)$; if the output is restricted to the two points that must be swapped then $s = \Theta(k)$.

Similar problems have been considered in both on-line and off-line mode for objects other than points. Ghali and Stewart [3, 4] developed an on-line algorithm for maintaining the view of a set of *line segments*. Pocchiola [7] presented a solution to the off-line version of the problem for a set of n objects. Roughly speaking, if these two results are scaled down to n points then the space requirements are $O(n^2)$ since they concentrate on cases with many occlusions among the given objects. The algorithm of Ghali and Stewart [3] is however similar to ours and they have implemented the algorithm. See [5] for an overview of other related results.

2 Preliminaries

We start with some simple properties pertaining to the view of a moving point.

Lemma 2 *Let q and q' be two points such that segment $\overline{qq'}$ intersects line $\overline{ab}$, $a, b \in P$, and no other line through two points of P. If segment $\overline{qq'}$ intersects segment $\overline{ab}$ then the view from q is the same as the view from q'. Otherwise the views are the same except that a and b are inverted. (Refer to Figure 1.)*

The *chipped line* through two points a and b is the line defined by a and b minus the (closed) line segment $\overline{ab}$. The points a and b are not on the chipped line. The following observation states that if q moves across the line $\overline{ab}$ then the view changes if only if q does not cross segment $\overline{ab}$.

The *CL-arrangement* induced by a set of points is constructed by drawing a chipped line through each pair of points. The cells of this arrangement are vertices, open edges and open faces. See Figure 2. The following lemma follows from Lemma 2.

Lemma 3 *Any two points in the same cell of a CL-arrangement have the same view.*

*INRIA Sophia-Antipolis, Olivier.Devillers@sophia.inria.fr
†Carleton University, vida@scs.carleton.ca
‡LORIA University Nancy 2, everett@loria.fr
§INRIA Rhône-Alpes, Samuel.Hornus@inria.fr
¶McGill University, sue@cs.mcgill.ca
‖University of Lethbridge, wismath@cs.uleth.ca

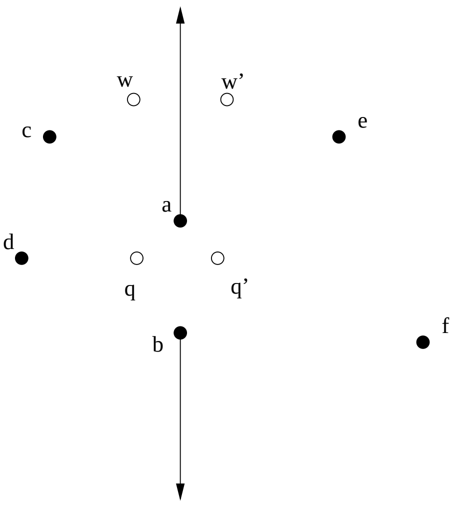


Figure 1: The view from q and q' is (a, e, f, b, d, c) from w it is (e, f, a, b, d, c) and from w' it is (e, f, b, a, d, c).

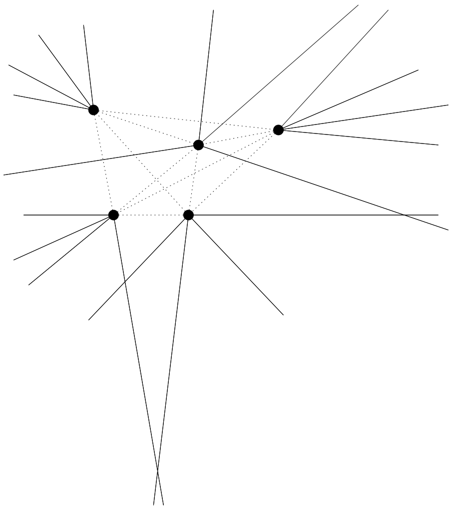


Figure 2: A CL-arrangement

Lemma 3 leads to the following straightforward solution to our problem. In a preprocessing step, compute the CL-arrangement of P. There are $O(n^4)$ cells in this arrangement which can be computed in $O(n^4)$ time [2]. Preprocess the arrangement for point location queries. To answer a moving viewpoint query for a segment $\sigma = \overline{qr}$, first compute the view from q by sorting the points around q. Find the cell C containing q using the point location structure. Shoot a ray from q along $\overline{qr}$ to find the first intersection point of $\overline{qr}$ with C; say $\overline{qr}$ intersects C at q' on an edge defined by the line through $\overline{cd}$. Compute the view from this point q' by removing all the points on line $\overline{cd}$ except for those visible to the viewpoint. Compute the view from a point in the neighboring cell of C into which the viewpoint moves. Repeat this process until the viewpoint arrives at the cell containing r.

Clearly the problem with this approach is the $O(n^4)$ space and preprocessing time needed to compute the CL-arrangement. We show how to reduce this significantly in section 3.

Another, less naive approach to solve the moving viewpoint query proceeds as follows: Recall that our goal is to maintain the circularly ordered set of points around q. The idea is to sort them first (as a preprocess), then, for each of the n pairs of circularly consecutive points, insert an event in an event-queue that tells when the swapping of 2 consecutive points should happen (if ever). The update is as follows: (A) Pop an event. (B) Update the circular ordering by swapping two consecutive elements. (C) Insert up to 3 new events in the queue, while deleting 2 old events. Special care is required when the viewpoint q moves out of the convex hull of P, but this does not increase the complexity. Overall, for a given line segment trajectory, the complexity is $O(n \log n)$ preprocessing time, and then $O(n \log n + k \log n + s)$ time. The $O(n \log n)$ term is required because, as the new trajectory is known, it is necessary to empty the event-queue, compute the event times for n pairs of points, and insert $O(n)$ events in the queue.

In the next section, we show how to remove this $O(n \log n)$ term.

3 Proof of Theorem 1

Rather than computing the entire arrangement, we will compute the cells of the arrangement only when we need them. Since the size of each cell is $O(n)$, it is the ability to perform updates efficiently without computing the entire CL-arrangement that is critical.

It will be convenient to rid ourselves of the non-convex cells of the CL-arrangement which, as the following lemma indicates, can be done by adding to the arrangement the edges of the convex hull of P. For example ray shooting is more easily performed in convex regions.

Lemma 4 *A vertex of a CL-arrangement is reflex in some cell if and only if it is a point on the convex hull.*

Proof. First notice that a reflex vertex must be one of the n points; indeed, all other arrangement vertices are at intersections of lines. Let p be a point corresponding to a reflex vertex. Now, since p is reflex in some cell, all of the $n-1$ rays emanating from p lie on one half-plane defined by a line through p and thus all of the other $n-1$ other points lie in the other half-plane which is exactly the condition for p to be on the convex hull. □

In the following, we assume that the algorithm is operating in the CL-arrangement which has been augmented to include the edges of the convex hull of p.

The following lemma will be useful for efficiently computing a cell given a view.

Lemma 5 *The edges of a fully-dimensional cell C of the CL-arrangement are determined by consecutive points of the view from any point q in C.*

Proof. Let s be a point on an edge of C and let a and b be the points generating the line containing s. We will distinguish the cases where s is on a chipped line or a convex hull edge.

Case 1: Chipped line.

Without loss of generality, say s is on the ray emanating from b. It suffices to show that a and b are consecutive points of the view of q. Suppose not. Then there is a point x of P in the wedge aqb. We first show that x is not in triangle aqb. See Figure 3. If it was then the ray emanating from x in direction $\overline{ax}$ intersects the segment $\overline{qs}$, which is entirely inside C by convexity, at a point p. But then the points b, x and a are ordered around any point on the segment $\overline{qp}$ in the order bxa and about any point on segment $\overline{ps}$ in the order bax which contradicts Lemma 3. Similarly, there is no point x anywhere else in the wedge aqb since then the ray emanating from x in direction $\overline{xb}$ intersects the segment $\overline{qs}$. See Figure 4.

Case 2: Convex hull edge.

Then s is on the segment $\overline{ab}$. First if q is inside the convex hull and if a and b are not consecutive then there is a point x in the wedge, and since $\overline{ab}$ is a convex hull edge, x belongs to triangle aqb. A ray from x crosses segment $\overline{qs}$ which should be inside C giving a contradiction; this ray is issued either from b as in Figure 5 or from a depending on the location of x with respect to segment $\overline{qs}$. If q is outside the convex hull, then either all the points are in the wedge aqb (outside the triangle) and a and b are consecutive, or a ray from b separates q and a giving a contradiction as in Figure 6 (or a ray from a separates q from b). $\square$

Lemma 5 implies that, in order to compute a cell it suffices to know the view from a point in that cell; each pair of consecutive points in the circular ordering contributes a half-plane, the intersection of which gives the cell. Now from Lemma 2 we know that to compute a cell from a neighboring cell, it suffices to delete two half-planes and insert two new ones. These operations can be accomplished in $O(\log n)$ amortized or $O(\log^2 n)$ worst-case time [1, 6].

The algorithm, which implies Theorem 1, is given in Figure 7. The input to the algorithm is a point set P and a query segment $\overline{qr}$. The algorithm is similar to the less efficient version suggested in the previous section, however note that the space requirements are contained to $O(n)$ as the point q' moves from q to r encountering the cells of the CL-arrangement without first computing the arrangement.

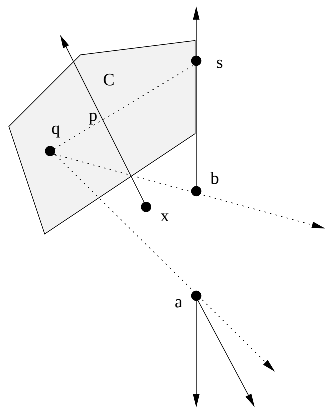


Figure 3: If point s is on chipped line $\overline{ab}$, there is no point in triangle aqb.

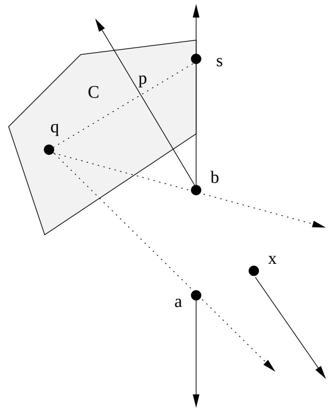


Figure 4: If point s is on chipped line $\overline{ab}$, there is no point in wedge aqb.

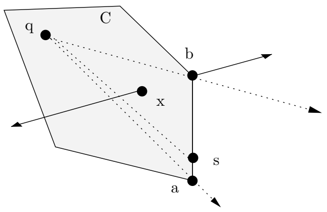


Figure 5: If point s is on convex hull edge $\overline{ab}$, there is no point in triangle aqb.

4 Extensions

The same approach works when the trajectory σ is not a line segment, the only difference being in the resulting complexity.

Theorem 6 *Let P be a set of n points in the plane. After $O(n \log n)$ preprocessing time, moving viewpoint queries along a specified curve σ can be answered in $O(f(n)+k\log n+s)$ amortized or $O(f(n)+k\log^2 n+s)$ worst-case time, where k is the number of different views seen from σ, s is the size of the output, and $f(n)$ is the*

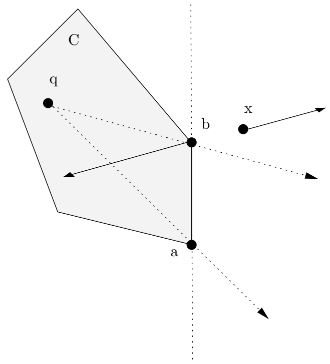


Figure 6: If point s is on convex hull edge $\overline{ab}$, there is no point outside wedge aqb.

Algorithm: Moving_Point_Query $(P, \overline{qr})$

1. Set q':=q.
2. Construct the view around q. – $O(n \log n)$.
3. Construct the cell C containing q. – $O(n \log n)$.
4. While $q' < r$ do
   - Shoot a ray from q' in the direction $\overline{qr}$ hitting C at point q''. – $O(\log n)$.
   - q':=q''
   - Update and output the view at q'. – $O(\log n)$.
   - Update the cell C. – $O(\log n)$ or $O(\log^2 n)$.

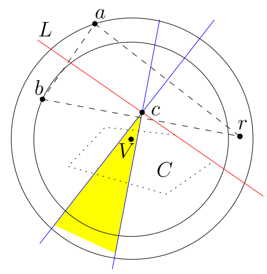
Figure 7: The algorithm and analysis

time required to perform ray-shooting in a convex n-gon along curves of the same type as σ.

Suppose as q moves from s to t along segment $\overline{st}$, the view at s is known, and that only the view at t is required, and not the intermediate views. The view at t can be computed by running our algorithm and a sorting algorithm in parallel. Then the time to compute the view at t is $\min\{O(n \log n), O(m \log n)\}$ where m is the number of cells that are traversed.

Finally, if P is the set of endpoints of a set of disjoint line segments, then the actual view of the segments can be obtained easily from the view of the points in linear time, thus making our algorithm useful in the case when few segments are occluded.

Distance-sort maintenance. Interestingly, the same algorithm can be used to maintain the set of points in P sorted according to their distance to a moving query point q. In this ordering, two consecutive points are swapped when q crosses their bisector. Thus, the CL-arrangement is replaced by an arrangement of the $n(n-1)$ bisectors called a *B-arrangement.* We present only a critical lemma to support our claim.



Lemma 7 *Each edge of a cell in the B-arrangement is supported by the bisector of two consecutive points in the distance-ordering.*

Proof. Let line L be the bisector defined by points a and b supporting one edge of a cell C containing the query point v. Assume that, in the distance-ordering, there is a point r between a and b. Then the circumcircle of a, b and r has center c, the point of intersection of the 3 bisectors defined by (a,b) (b,r) and (r,a). We have $\overline{vb} < \overline{vr} < \overline{va}$. Therefore v lies in the yellow shaded region. Hence the line L cannot support an edge of C, which is a contradiction. $\square$

By redefining the viewpoint query of theorem 1 to report changes in the distance-ordering, we obtain the same complexity result.

5 Conclusion

We present here the first solution to the moving viewpoint query problem among points in 2D with provable worst-case and amortized complexity. Solving the problem in 3D remains a difficult open problem.

References

[1] G. Brodal and R. Jacob, Dynamic planar convex hull, *Proc. 43rd Annual Symposium on Foundations of Computer Science*, 2002, 617-626.

[2] H. Edelsbrunner and L. Guibas. Topologically Sweeping an arrangement, *J. of Computer and System Sciences* 38, 165-194, 1989

[3] S. Ghali and J. Stewart. Incremental Update of the Visibility Map as Seen by a Moving Viewpoint in Two Dimensions, *Eurographics 7th Workshop on Computer Animation and Simulation*, 1996.

[4] S. Ghali and J. Stewart. Maintenance of the Set of Segments Visible from a Moving Viewpoint in Two Dimensions, *Proc. of ACM Symposium on Computational Geometry*, V3–V4, May 1996.

[5] O. Hall-Holt. *Kinetic Visibility.* PhD Dissertation, Stanford University, 2002.

[6] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166-204, 1981.

[7] M. Pocchiola. Graphics in Flatland revisited. *Proc. 2nd Scand. Workshop Algorithm Theory*, LNCS Vol 447, Springer-Verlag 85–96, 1990.