A Simple Output-Sensitive Algorithm for Hidden Surface Removal

MICHA SHARIR
Tel Aviv University and New York University

MARK H. OVERMARS
Utrecht University

We derive a simple output-sensitive algorithm for hidden surface removal in a collection of n triangles in space for which a (partial) depth order is known. If k is the combinatorial complexity of the output *visibility map*, the method runs in time $O(n\sqrt{k}\log n)$. The method is extended to work for other classes of objects as well, sometimes with even improved time bounds. For example, we obtain an algorithm that performs hidden surface removal for n (nonintersecting) balls in time $O(n^{3/2}\log n + k)$.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*geometric algorithms, languages and systems;* I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*hidden line/surface removal*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Hidden line elimination, hidden surface removal, object-space algorithm

1. INTRODUCTION

An important problem in computer graphics is *hidden surface removal*. In a typical setting of the problem we are given a collection of nonintersecting polyhedral objects in 3-space, and a viewing point v, and our goal is to construct the view of the given scene, as seen from v.

Work by M. Sharir has been supported by Office of Naval Research grant N00014-87-K-0129, by National Science Foundation grant CCR-89-01484, and by grants from the U.S.-Israeli Binational Science Foundation, the Israeli National Council for Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences. Work by M. H. Overmars was partially supported by the ESPRIT II Basic Research Actions Program of the EC, under contract No. 3075 (project ALCOM).

Authors' addresses: M. Sharir, School of Mathematical Sciences, Tel Aviv Univ., Tel Aviv, Israel, and Courant Institute of Mathematical Sciences, New York University; M. H. Overmars: Department of Computer Science, Utrecht Univ., P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

Many solutions have been developed to date. Some of them use an "image-space" approach, in which one tries to calculate, for each pixel in the viewed image, which object is visible at that pixel. These techniques generally have hardware implementations, but they can still be slow when the screen size and the number of objects in the scene are both large [22]. Other techniques, like the one presented here, have an "object-space" flavor. That is, they try to obtain a discrete combinatorial representation of the view of the scene, whose complexity does not depend on the screen size, but only on the combinatorial complexity of the scene. Such object space methods are also important, e.g., for hidden line elimination or determining the parts of objects that are lighted by light sources.

Let n denote the number of edges of the given polyhedra. If we project all these edges onto the plane of view we obtain an arrangement of n (usually intersecting) segments. The visible portion of the scene, when projected onto the view plane, yields a polygonal decomposition of the plane into regions, at each of which a connected portion of a single object face is visible, or no object is visible. These regions are bounded by portions of the projected object edges, and the vertices of these regions are either projected object vertices or intersections of projected edges.

These well-known observations lie at the basis of practically all object-space hidden surface removal algorithms. Many of these algorithms simply calculate the entire arrangement of the projected edges, and then determine which features of the arrangement are visible. Crude implementations of this approach run in time $O(n^2)$ [4, 12]. More careful implementations run in time $O((n + I)\log n)$, where I denotes the number of intersections between the projected edges [5]. See also Güting and Ottman [8], Nurmi [14], and Schmitt [21].

The main problem with these solutions is that they are not *output-sensitive*. When the combinatorial complexity of the viewed scene is small, these solutions can be very inefficient. A typical example that is often used to illustrate this issue consists of a large horizontal rectangle, lying below and completely hiding a gridlike pattern of long thin slabs (see Figure 1). In this case $I = \Theta(n^2)$, so any of these algorithms requires at least quadratic time, even though the complexity of the viewed scene is constant!

Several solutions that address the output-sensitivity issue have been proposed. Some of these techniques deal with the restricted case in which the objects are all horizontal axis-parallel rectangles, and lead to fairly efficient output sensitive algorithms [1, 6, 20]. Another output-sensitive algorithm has recently been proposed by Reif and Sen [18], for the special case of a polyhedral terrain (i.e., a piecewise linear surface meeting each vertical line in exactly one point).

However, very little has been done in the general case of arbitrary polyhedral objects. There are two main difficulties that arise in this case. One is the possible lack of order among the object faces, in terms of "nearness" to the viewing point. (It is easy, for example, to construct three nonintersecting triangles, so that each of them hides a portion of the next one in cyclic order.) This is a rather problematic issue even for image-space techniques (see, e.g.,

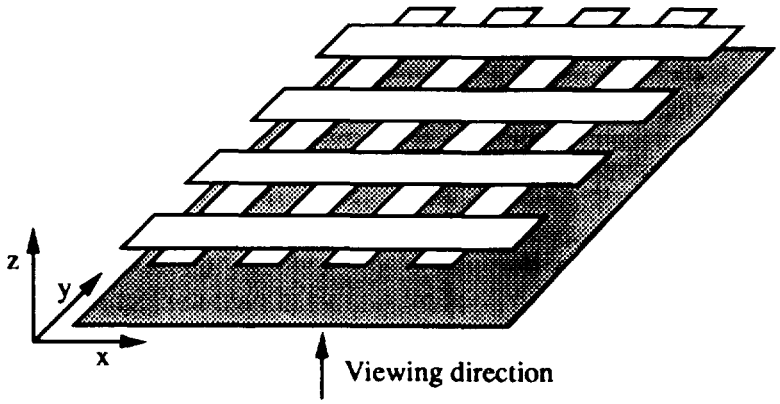

Fig. 1. A scene with a small output size.

Sutherland et al. [22]). To overcome this problem, one usually partitions some of the objects into smaller pieces, by cutting them along appropriate planes, so that the resulting pieces can be ordered by their nearness to the viewing point. However, such partitions can result in an unacceptably high number of "subobjects" (see Paterson and Yao [17] for a recent treatment of this problem). Note that in both cases of axis-parallel rectangles and polyhedral terrains, an ordering of the desired kind is easy to obtain. In this paper we also bypass the ordering problem by assuming that among the given objects no cyclic overlap occurs. In such a case a partial order by nearness to the viewing point exists. We assume the partial order is known. (Computing such an order can in general be time consuming but in many cases is easily obtained.) Such a partial order can then be extended to a total order.

Even when a proper ordering of the objects exists, we still face the difficulty of handling object edges with arbitrary orientations. These difficulties have stalled progress in developing more general output-sensitive solutions. There are only two related works we are aware of. One is by Mulmuley [13] where a randomized "quasi-output-sensitive" solution is obtained; the expected time complexity of this solution is expressed as a sum of weights associated with the intersection points of the projected object edges, where the weight of an intersection is inversely proportional to the number of objects "hiding" that intersection from the viewing point. A second recent work by Schipper and Overmars [19] creates the view of the scene by adding the objects one by one in increasing distance from the viewing point. It uses dynamic partition trees to maintain the boundary of the union of the projected objects so as to facilitate efficient calculation of the intersections of the projections of newly added objects with that boundary. However, the dependence of the complexity of this algorithm on the output size is rather weak; in particular it may run in considerably more than quadratic time if the output size is close to quadratic.

In this paper we present an improved output-sensitive algorithm for the hidden surface removal problem that is conceptually very simple and works

for various classes of objects. The only assumption we have to make is that a (partial) depth order for the objects is known. Let k be the number of vertices in the projected visible scene. The method runs in time $O(n\sqrt{k}\log n)$. Thus, apart from the log n factor, the algorithm is always at most quadratic, and is faster when k is subquadratic. The simplicity of the algorithm also makes it attractive for pragmatic implementation. If we specialize this algorithm to the case of n horizontal discs, we obtain an algorithm whose time complexity is $O(n^{3/2}\log n + k)$. Thus the algorithm becomes optimal if k is sufficiently large. (Note that, also in the case of discs, k can be as large as $\Omega(n^2)$.) Similar performance is obtained for collections of nonintersecting spheres, for collections of nonintersecting isothetic copies of any convex body, and for several other special cases.

The paper is organized as follows. In Section 2 we describe some preliminary results on merging visibility maps. In Section 3 we present our algorithm, restricted to the case of a set of triangles. In Section 4 we extend this to arbitrary polygons and we treat some special cases. The paper is concluded in Section 5, with a discussion of our results and a list of open problems.

2. PRELIMINARIES

Let us first introduce some terminology to define more formally the concepts of visibility and related notions.

Let $\mathbf{T} = \{T_1, \ldots, T_n\}$ be a set of triangles in 3-D space without cyclic overlap. We assume that the triangles are ordered in such a way that for $i < j$ either T_i and T_j are disjoint in the projection or T_i lies (partially) in front of T_j. Let P be the projection plane. Project each triangle T_i perspectively from v on P, obtaining a projected triangle T_i'. Transform P such that it lies on the xy-plane. Finally translate each transformed triangle T_i' to a height i. In this way we obtain a set $\Delta = \{\Delta_1, \ldots, \Delta_n\}$ of n horizontal triangles in 3-D space, so that Δ_i lies in the plane $z = i$. It is easy to verify that the view from v in $\mathbf{T}$ is the same as the view from a viewing point at $z = -\infty$ of Δ. Transforming $\mathbf{T}$ into Δ can easily be done in time $O(n)$ (or $O(n\log n)$ if the depth ordering is not yet known).

The hidden surface removal problem for Δ (and for a viewing point at $z = -\infty$) can be formulated as the problem of partitioning the xy-plane into maximal regions so that for each region R there exists a unique triangle Δ_i (or no triangle at all) such that for all points $(x, y) \in R$, the lowest triangle lying above (x, y) is Δ_i (or no triangle lies above (x, y)). We denote by M the planar map resulting from this partitioning, and call it the *visibility map* of Δ. The combinatorial complexity or *size* of M, denoted by k, is the number of edges and vertices in M. Note that k can be as large as $\Theta(n^2)$ and as small as 6 (when one triangle hides all the others). See Figure 2 for an example of the visibility map we might obtain.

Our algorithm, to be described in the next section, is based on merging visibility maps. For this we need the following result:

LEMMA 2.1. *Let M be a visibility map of size k and let C be an arbitrary polygonal region of size c. The parts of M lying outside C can be determined in*

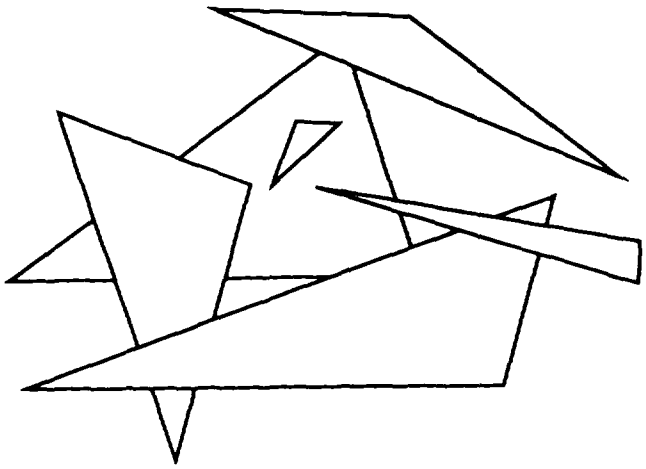

Fig. 2. A visibility map M.

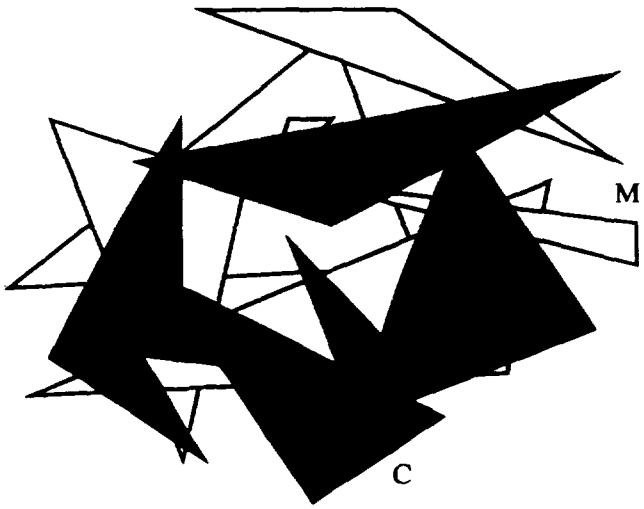

Fig. 3. Intersection between M and C.

time $O((k + c)\log(k + c) + k')$ *where* k' *is the number of intersections between edges of* C *and edges of* M.

PROOF. See Figure 3 for an example. To determine the part of M outside C we have to compute all intersections between edges of M and edges of the boundary of C. Note that the edges in M do not intersect one another and the same applies to the edges of the boundary of C. Hence, we can use the red-blue intersection algorithm of Mairson and Stolfi [10] to find all intersections within the time bound stated. (We could alternatively use the method of

Chazelle and Edelsbrunner [3] but this technique is much more complicated; besides, Mairson and Stolfi's technique can be extended to the case of curved edges, whereas Chazelle and Edelsbrunner's technique cannot.) After we have determined all intersections, the parts of M outside C can easily be computed. □

Remark. Intuitively, think of C as the xy-projection of a polygonal object lying below all triangles from which M is computed and hiding some portions of M. The lemma enables us to compute those portions of M that can be seen, i.e., are not hidden by C.

3. THE ALGORITHM

The main idea of our solution is quite simple. We construct the map M by adding the triangles of Δ from bottom up in an incremental fashion. Rather than adding them one by one, as was done by Schipper and Overmars [19], we add them in groups, where the size of a group depends on the output size of the partial visibility map constructed so far.

In more detail, the algorithm proceeds as follows. Choose an initial constant positive integer parameter c_0 (6 will do). The algorithm then iterates through an incremental round, at which a group of triangles is added to the scene. After the jth iteration, we have processed a collection G_j of the lowest n_j triangles. We denote by M_j the visibility map of this collection. (Note that each edge and vertex of M_j is part of the final visibility map M.) Let k_j denote the combinatorial complexity of M_j, and let $c_j \leq k_j$ denote the number of edges of M_j that bound faces in which no triangle of G_j is visible. In other words, c_j is the size of the *contour* (i.e., boundary) of the union of G_j (projected on the xy-plane). Let C_j denote this contour. Initially, $j = 0$, C_0 and M_0 are empty, but c_0 is the initial chosen parameter.

At the jth iteration, we take the set G_j' of the next $\lceil \sqrt{c_{j-1}} \rceil$ lowest triangles, and construct the visibility map M_j' of G_j', ignoring the rest of the triangles. This can be done in $O(c_{j-1})$ time, using, e.g., the method McKenna [12] and the combinatorial complexity of M_j' is also $O(c_{j-1})$. (Alternatively, one can call the algorithm recursively. In this case the time bound for this step goes up to $O(c_{j-1} \log c_{j-1})$ but this does not influence the total time bound.) Next we clip M_j' against C_{j-1} using the result of Lemma 2.1. The part of M_j' found this way will be part of the final visibility map M so we add it to M_{j-1} to obtain M_j. Finally we merge C_{j-1} with M_j' to obtain C_j and we set c_j to be the size of C_j.

This procedure is continued until all triangles are processed. In Figure 4 a more concise description of the algorithm is given.

It is easy to verify that the algorithm always terminates, and that the final map it produces is the desired visibility map M. Indeed, each c_j is at least 6 (this happens when the contour becomes just a single triangle), so each round "consumes" at least 3 triangles. Correctness follows partially from the fact that, since triangles are processed in the order of their nearness to the viewing point, each feature of any partial map M_j must also be a feature of

1. **Set** $j := 0$; $c_0 := 6$;
2. Set $C_0 = \emptyset$; $M_0 := \emptyset$;
3. **While** Δ is not empty **do**
4. $\quad j := j + 1$;
5. $\quad$ Let $G'_j :=$ next $\lceil \sqrt{c_{j-1}} \rceil$ triangles in Δ; remove them from Δ;
6. $\quad$ Compute the visibility map M'_j of G'_j;
7. $\quad$ Clip M'_j against C_{j-1};
8. $\quad$ Merge the result with M_{j-1} to obtain M_j;
9. $\quad$ Compute C_j from M'_j and C_{j-1};
10. $\quad$ Set $c_j :=$ size of C_j;

Fig. 4. The main algorithm.

the final map M, and each feature of M must arise either as a feature of some M'_j or as an interaction between the contour of some M_{j-1} and the visibility map M'_j.

The time complexity of the algorithm can be estimated as follows. Suppose the algorithm consists of r rounds. The jth round of the algorithm "consumes" $\lceil \sqrt{c_{j-1}} \rceil$ more triangles, so we have

$$\sum_{j=0}^{r-1} \lceil \sqrt{c_j} \rceil = n.$$

(To be precise, $\sum_{j=0}^{r-1} \lceil \sqrt{c_j} \rceil < 2n$ because in the last round not all $\lceil \sqrt{c_{r-1}} \rceil$ triangles might exist.) Step 6 of the algorithm in Figure 4 takes time $O(c_{j-1})$ using, e.g., [12]. By Lemma 2.1, step 7 takes time $O(c_{j-1} \log c_{j-1} + k'_j)$, where k'_j is the number of intersections between edges of M'_j and of C_{j-1}. As argued above, each such intersection must be a feature of the final map M; moreover, no feature of M can arise in this manner in more than one iteration of the while loop. Hence,

$$\sum_{j=1}^{r} k'_j \leq k.$$

Steps 8 and 9 can be carried out in time $O(c_{j-1} + k'_j)$. Hence, the total time for the jth round is $O(c_{j-1} \log c_{j-1} + k'_j)$. The total time complexity of all rounds is

$$\sum_{j=0}^{r-1} O(c_j \log c_j + k'_{j+1}) =$$

$$\sum_{j=0}^{r-1} O(c_j \log n) + \sum_{j=1}^{r} O(k'_j) =$$

$$\sum_{j=0}^{r-1} O\left(\sqrt{c_j}\ \sqrt{c_j}\log n\right) + O(k) =$$

$$\sum_{j=0}^{r-1} O\left(\left\lceil \sqrt{c_j}\right\rceil \sqrt{c}\log n\right) + O(k) =$$

$$O\left(n\sqrt{c}\log n + k\right),$$

where c is the maximal contour size during the construction. Clearly $c \leq k$ and, since $k = O(n^2)$, we also have $k = O(n\sqrt{k})$. Thus we conclude

THEOREM 3.1. *The incremental algorithm outlined above performs hidden surface removal in time $O(n\sqrt{k}\log n)$, where k is the output size. More precisely, the running time of the algorithm is $O(n\sqrt{c}\log n + k)$, where c is the maximum contour size during the execution of the algorithm.*

Remark. Apart for the log n factor, the algorithm is at most quadratic for any value of k, and becomes close to $O(n \log n)$ when k is small. It would be nice to improve the algorithm so that it never becomes more than quadratic, perhaps by employing the technique of Guibas and Seidel [7] for merging planar convex maps. (Our maps are not convex, but the only nonconvex vertices are projections of visible triangle vertices. Can this property be used to improve the performance of the algorithm?)

4. EXTENSIONS AND SPECIAL CASES

We will first show how to extend the method presented in the previous section to polygons other than triangles. A first idea that might come to mind is to subdivide all polygons into triangles. Unfortunately, this might increase the output size considerably (it might go up from $O(n)$ to $\Theta(n^2)$). Hence, we have to be more careful.

So we are given a set $\mathbf{P} = \{P_1, \ldots, P_z\}$ of polygons, ordered in terms of their nearness to the viewing point, as above. Using the same technique as given in Section 2 we can assume that all polygons are parallel to the xy-plane with P_i lying at height $z = i$. Let P_i have n_i vertices, and let $n = \sum_{i=1}^{z} n_i$ denote the total number of vertices. We adapt the algorithm in Figure 4 such that, in line 5, we do not take the next $\lceil \sqrt{c_{j-1}} \rceil$ polygons but a number of next polygons with a total of about $\lceil \sqrt{c_{j-1}} \rceil$ vertices. If this is always possible, it is easy to see that the method runs in the same time bound.

Unfortunately, some polygons might have many vertices, forcing us to take more than the required $\lceil \sqrt{c_{j-1}} \rceil$ vertices. To avoid this we have to split the last polygon to be taken in a round such that the number of vertices is correct. The leftover piece is then treated in the next round. Splitting the polygon can easily be done in time $O(1)$ when the polygon is convex. Otherwise, we can find the correct splitting diagonal in time $O(\log n)$ after an overall $O(n \log n)$ preprocessing (for all polygons), using the polygon cutting theorem of Chazelle [2].

In this way the method clearly works. We only have to show that adding the extra cutting edges does not influence the time bound. A cutting edge e is not part of the actual visibility map but is treated by the algorithm as a polygon boundary. Hence, it will appear in the visibility map. As a result we do spend time computing the visible parts of e. However, if edge e is treated in round j it is easy to see that e is broken up into at most $O(c_{j-1})$ visible pieces. Hence, we spend $O(c_{j-1})$ extra time for this. This does not influence the order of magnitude of the time bound for round j. This leads to the following general theorem:

THEOREM 4.1. *Let V be a set of polygons with a total of n vertices. Let v be a viewpoint and assume the polygons can be ordered by nearness to v. Then the visible parts of the polygons in V can be computed in time $O(n\sqrt{c}\log n + k)$ where c is the maximal contour size during the construction and k is the size of the final visibility map.*

Let us now look at some special cases. If instead of triangles we have a collection of n horizontal discs, or, more generally, of objects ("pseudo-discs") whose xy-projections have the property that each pair of their boundaries intersect in at most two points, then we have the following theorem.

THEOREM 4.2. *The above algorithm, when appropriately modified and applied to a collection of n horizontal discs (or pseudodiscs, in the above sense), performs hidden surface removal in time $O(k + n^{3/2}\log n)$, where k is the output size.*

PROOF. As has been shown by Kedem et al. [9], the contour size in the case of discs (or pseudodiscs) is always $O(n)$. When we apply the preceding algorithm to a collection of discs, we can still use Mairson and Stolfi's algorithm [10] to clip the map M_j' against C_{j-1} since this algorithm can handle curved arcs. The asserted bound is now immediate. □

Many sets of objects behave like pseudodisks. For example, when all objects are nonintersecting isothetic copies of the same convex object (scaling allowed but no rotations) their xy-projections will be pseudodisks. This in particular yields the following corollary:

COROLLARY 4.3. *One can perform hidden surface removal for a collection of n nonintersecting balls, or for a collection of n isothetic nonintersecting copies of an arbitrary convex object, in time $O(k + n^{3/2}\log n)$, where k is the output size.*

Remark. In Theorem 4.2 and Corollary 4.3, of course, we have to assume an appropriate model of computation in which various basic operations, such as finding the intersections of the boundaries of a pair of projected objects, can be performed in constant time.

Another special case are sets of "fat" triangles. A set of triangles is called fat if each triangle has internal angles of at least δ degrees for some constant δ (independent of the size of the set). Matoušek et al. [11] show that the contour of such a set of fat triangles has size $O(n \log n)$. (The constant in the

bound depends on δ.) This immediately leads to the following result:

THEOREM 4.4. *The above algorithm, when applied to a collection of n fat triangles, performs hidden surface removal in time* $O(k + (n \log n)^{3/2})$, *where k is the output size.*

5. CONCLUSION

In this paper we have presented a new, simple, output-sensitize algorithm for hidden surface removal in a set of polygons for which a (partial) depth order is known. The method runs in time $O(n\sqrt{k} \log n)$ where n is the number of vertices in the set of polygons and k is the complexity of the output visibility map. The technique can also be extended to handle curved objects. The only restriction made by the algorithm presented is that no cyclic overlap should occur in the relationship of nearness of the given objects to the viewing point.

In many situations the method is more efficient. In fact, the time complexity depends on the maximal contour size of "prefix subcollections" of the objects during the construction. This is often smaller than k. In particular, for a set of n (nonintersecting) balls, the method runs in time $O(n^{3/2} \log n + k)$.

The results obtained in this paper also raise several related open problems. A first question is of course whether the results can be improved. Using the same idea, it might be possible to remove the $\log n$ factor but for further improvement other techniques seem to be necessary. We give a very complicated technique [16] that obtains a time bound of $O(n^{4/3} \log^{\gamma} n + k^{3/5} n^{4/5+\delta})$ for any $\delta > 0$, where γ is a constant less than 3. Also see Overmars and Sharir [15]. Although this method is theoretically faster for large k, it is too complicated to be practically interesting. Hence, other techniques are required.

Secondly, there is the problem of cyclic overlap. Up to now no output-sensitive methods are known that can deal with cyclic overlap with running time less than $O(n^2)$.

Finally, there is the problem of dynamically maintaining a visibility map when inserting or deleting objects or while moving the point of view. Only in the case of rectangles Bern [1] has given some dynamic results. No efficient results are known for more general objects.

ACKNOWLEDGMENT

We would like to thank Bart Luijten for suggesting the generalization to polygons given in Section 4.

REFERENCES

1. BERN, M. Hidden surface removal for rectangles. *J. Comput. Syst. Sci. 40*, 1 (Feb. 1990), 49-69.
2. CHAZELLE, B. A theorem on polygon cutting with applications. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science* (Chicago, Ill., Oct. 1982), pp. 339-349.
3. CHAZELLE, B., AND EDELSBRUNNER, H. An optimal algorithm for intersecting line segments in the plane. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science* (White Plains, N.Y., Oct. 1988), pp. 590-600.

4. DÉVAI, F. Quadratic bounds for hidden line elimination. In *Proceedings of the 2nd ACM Symposium on Computational Geometry* (White Plains, N.Y., June 1986), pp. 269-275.
5. GOODRICH, M. T. A polygonal approach to hidden line elimination. In *Proceedings of the 25th Allerton Conference on Communication, Control and Computing* (Monticello, Ill., Sept. 1987), pp. 849-858.
6. GOODRICH, M. T., ATALLAH, M. J., AND OVERMARS, M. H. An input-size/output-size trade-off in the time-complexity of rectilinear hidden surface removal. In *Proceedings of the ICALP'90* (Warwick Univ., England, July 1990), Springer-Verlag, Lecture Notes in Computer Science 443, 1990, pp. 689-702.
7. GUIBAS, L., AND SEIDEL, R. Computing convolutions by reciprocal search. In *Proceedings of the 2nd ACM Symposium on Computational Geometry* (White Plains, N.Y., June 1986), pp. 90-99.
8. GÜTING, R. H., AND OTTMAN, T. New algorithms for special cases of the hidden line elimination problem. *Comput. Vision, Graph. Image Processing 40*, 2 (Nov. 1987), 188-204.
9. KEDEM, K., LIVNE, R., PACH, J., AND SHARIR, M. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom. 1*, 1 (1986), 59-71.
10. MAIRSON, H., AND STOLFI, J. Reporting and counting intersections between two sets of line segments. In *Theoretical Foundations of Computer Graphics and CAD*, R. A. Earnshaw, Ed., NATO ASI Series, Vol F-40, Springer Verlag, 1988, pp. 307-326.
11. MATOUŠEK, J., PACH, J., SHARIR, M., SIFRONY, S., AND WELZL, E. Fat triangles determine linearly many holes. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, (San Juan, Oct. 1991).
12. MCKENNA, M. Worst-case optimal hidden surface removal. *ACM Trans. Graph. 6*, 1 (1987), 19-28.
13. MULMULEY, K. An efficient algorithm for hidden surface removal, I. *Comput. Graph. 23*, 3 (July 1989), 379-388.
14. NURMI, O. A fast line-sweep algorithm for hidden line elimination. *BIT 25*, 3 (1985), 466-472.
15. OVERMARS, M. H., AND SHARIR, M. Output-sensitive hidden surface removal. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science* (Research Triangle Park, N.C., Oct. 1989), pp. 598-603.
16. OVERMARS, M. H., AND SHARIR, M. An improved technique for output-sensitive hidden surface removal. Tech. Rept. RUU-CS-89-32, Dept. of Computer Science, Utrecht Univ., 1989.
17. PATERSON, M., AND YAO, F. Binary partitions with applications to hidden surface removal and solid modelling. *Discrete Comput. Geom. 5*, 5 (1990), 485-503.
18. REIF, M., AND SEN, S. An efficient output-sensitive hidden surface removal algorithm and its parallelization, In *Proceedings of the 4th ACM Symposium on Computational Geometry* (Urbana, Ill., June 1988), pp. 193-200.
19. SCHIPPER, H., AND OVERMARS, M. H. Dynamic partition trees. *BIT*, 1991. To appear.
20. PREPARATA, F. P., VITTER, J. S., AND YVINEC, M. Computation of the axial view of a set of isothetic parallelepipeds. *ACM Trans. Graph. 9*, 3 (1990), 278-300.
21. SCHMITT, A. Time and space bounds for hidden line and hidden surface algorithms. In *Eurographics '81*, pp. 43-56.
22. SUTHERLAND, I. E., SPROULL, R. F., AND SCHUMACKER, R. A characterization of ten hidden-surface algorithms. *Comput. Surv. 6*, 1 (Mar. 1974), 1-25.

Received November 1989; revised March 1990; accepted September 1990

Editor: David Dobkin