Efficient Hidden Surface Removal for Objects with Small Union Size*

Matthew J. Katz[†] Mark H. Overmars[‡] Micha Sharir[§]

Abstract

Let S be a set of n non-intersecting objects in space for which we want to determine the portions visible from some viewing point. We assume that the objects are ordered by depth from the viewing point (e.g., they are all horizontal and are viewed from infinity from above). In this paper we give an algorithm that computes the visible portions in time $O((U(n)+k)\log^2 n)$, where $U(n')$ is a super-additive bound on the maximal complexity of the union of (the projections on a viewing plane of) any n' objects from the family under consideration, and k is the complexity of the resulting visibility map. The algorithm uses $O(U(n)\log n)$ working storage. The algorithm is useful when the objects are "fat" in the sense that the union of the projection of any subset of them has small (i.e., subquadratic) complexity. We present three applications of this general technique: (i) For disks (or balls in space) we have $U(n) = O(n)$, thus the visibility map can be computed in time $O((n+k)\log^2 n)$. (ii) For 'fat' triangles (where each internal angle is at least some fixed θ degrees) we have $U(n) = O(n \log\log n)$ and the algorithm runs in time $O((n\log\log n+k)\log^2 n)$. (iii) The method also applies to computing the visibility map for a polyhedral terrain viewed from a fixed point, and yields an $O((n\alpha(n) + k)\log n)$ algorithm.

*Work by Mark Overmars has been partially supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM) and by the Dutch Organisation for Scientific Research (N.W.O.). Work on this paper by Matthew Katz and Micha Sharir has been supported by a Grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development. Work by Micha Sharir has also been supported by Office of Naval Research Grant N00014-90-J-1284, by National Science Foundation Grant CCR-89-01484, and by grants from the U.S.-Israeli Binational Science Foundation, and the Fund for Basic Research administered by the Israeli Academy of Sciences.

[†]School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel.

[‡]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

[§]School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA.

Figure 1: The visibility map of six triangles.

1 Introduction

In the past few years much attention has been given in computational geometry to the *hidden surface removal problem*, one of the central problems in computer graphics. In a typical setting of the problem we are given a collection of n non-intersecting polyhedral or other 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.

Most solutions to the problem as applied in graphics 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 (see e.g. [28]).

Recently a considerable effort has been made to obtain efficient "object-space" methods that try to compute 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. This view consists of a subdivision of the viewing plane into maximal connected regions in each of which (some portion of) a single object can be seen, or no object is seen. The obtained subdivision is called the *visibility map* of the given collection of objects. See figure 1 for an example.

A major challenge in this direction is to obtain *output-sensitive* algorithms, namely algorithms whose running time depends on the actual combinatorial complexity, k, of the visibility map, so that if k is small the algorithms will run more efficiently. Early object-space methods have a running time of $O(n^2)$, independent of the complexity of the resulting visibility map [9, 17]. Other implementations run in time $O((n+I)\log n)$, where I denotes the number of intersections between the projected edges [10, 12, 19, 27], which may also be insensitive to the output size (there are easy examples where $I = \Theta(n^2)$ but k is a constant). Another recent technique [18] uses a randomized incremental approach, leading to expected running time that is expressed as a weighted sum over the I intersection points; however, this technique is also not output-sensitive.

The most general output-sensitive hidden surface removal method to date (for polyhedral objects) is due to de Berg at al. [4]. Using the recent data structure of Agarwal and Matoušek [1], the method actually runs in time $O(n^{2/3+\epsilon}k^{2/3})$ for any arbitrarily small $\epsilon > 0$. This method works for sets of triangles in space with possible cyclic overlap (i.e. no depth order needs to exist). However, the method is rather complicated. A simpler method was proposed by Overmars and Sharir [20] (see also [26]). It computes the view of a set of horizontal triangles (or other flat objects with a simple shape), as seen from above, in time $O(n\sqrt{k}\log n)$. Although these methods are output-sensitive, the running time is still quite high. Better results have been obtained for special cases, like axis-parallel rectangles [2, 11, 24], c-oriented polyhedra [5, 12], polyhedral terrains [25], and unit disks

[21]. In these cases, the running time of the improved algorithms is $O((n + k)\text{polylog}\, n)$.

In this paper we develop a new technique for output-sensitive hidden surface removal. The technique is fairly general and simple, but its efficiency shows up when the objects have the property that the union of the projections on the viewing plane of any subcollection of j of them has small combinatorial complexity (by 'small' we mean $o(j^2)$, and typically close to linear in j). We refer to objects with this property as being "fat". Let $U(n)$ be a bound on the maximum combinatorial complexity of the union of the projections of any n objects from such a family, and suppose that $U(n)$ is *super-additive*, i.e., $U(n_1) + U(n_2) \leq U(n_1 + n_2)$. We show that the view of n such fat objects can be computed in time $O((U(n)+k)\log^2 n)$, using $O(U(n)\log n)$ working storage. The method is simple and, hence, potentially practical.

We present three applications of the technique:

- If the given objects are horizontal disks (or, for that matter, pairwise disjoint balls) viewed from any fixed point, then $U(n) = O(n)$ [14]. In this case our technique yields an algorithm with running time $O((n+k)\log^2 n)$.

- If the given objects are horizontal 'fat' triangles, namely triangles whose angles are all at least some fixed angle θ, which are viewed from any fixed point, then $U(n) = O(n \log\log n)$ [16]. In this case our technique yields an algorithm with running time $O((n \log\log n + k)\log^2 n)$.

- Finally we consider the case of viewing a polyhedral terrain from any fixed point. Here one has $U(n) = O(n\alpha(n))$, where $\alpha(n)$ is the extremely slowly growing inverse of Ackermann's function [8]. In this case our technique yields an algorithm with running time $O((n\alpha(n) + k)\log n)$. (The simpler structure of the visibility map in this case facilitates a saving of a $\log n$ factor in the time bound.)

Like most of the results on output-sensitive hidden surface removal (except for the very recent methods in [4, 5]), our technique assumes a depth order among the viewed objects, which is easy to compute and which excludes cyclic overlaps among them. Problems in which such an order is not available or does not exist are much harder to handle, especially if comparable efficiency is being thought (see e.g. [6, 7, 22] for the extra techniques that may be required).

The paper is organized as follows. In Section 2 we describe the algorithm. In Section 3 we analyze its run-time and show how to improve the storage to the bound given above. In Section 4 we present the applications listed above. The paper is concluded in Section 5 with a discussion of our results and some open problems.

A preliminary version of this paper appeared in [13]. That paper also presents a second, more complicated method, that yields exactly the same performance bounds.

Figure 2: The tree $\mathcal{T}$ with the regions U_δ and V_δ (shaded).

2 The algorithm

We first present a simpler version of the method where we do not optimize the working storage. This version is really simple — it involves two divide-and-conquer passes over the objects ordered by depth from the viewing point. At each recursive call we compute the union, intersection, or difference of two planar regions, using standard line-sweeping methods. In this version the working storage is $O((U(n) + k) \log n)$. Optimizing the storage requires a more careful handling of the recursive process.

As a first step the method sorts the objects by depth order and stores them in this order in the leaves of a balanced binary tree $\mathcal{T}$, the nearest object in the leftmost leaf. For each node δ of $\mathcal{T}$ we compute the following two maps:

- U_δ — the union of the projections of the objects in the subtree $\mathcal{T}_\delta$ of $\mathcal{T}$ rooted at δ.
- V_δ — the visible portions of U_δ, i.e. the subset of U_δ consisting of those points that are not contained in the projection of any nearer object (stored in $\mathcal{T}$ to the left of δ).

Both U_δ and V_δ are planar regions, possibly with holes. Their boundary consists of portions of projected edges of the original objects. Clearly $V_\delta \subseteq U_\delta$. In following the description of the algorithm, it is helpful to visualize U_δ as a new nominal object obtained by "squashing" all objects stored below δ onto some common in-between plane and gluing them together. V_δ can be thought of as the portions of the new object that are visible in the standard sense. See figure 2 for the tree we obtain for the example in figure 1. At each node U_δ is drawn. The shaded part at each node is V_δ.

Once we have computed V_δ for each node δ in the tree we are done, because for each leaf δ, V_δ consists precisely of those parts of the object stored in this leaf that are visible. So reporting V_δ for all leaves gives the entire visibility map (those V_δ's can easily be glued together in a final step of the algorithm to obtain the global visibility map).

Computing U_δ for all nodes is quite easy. We do this in a bottom-up manner by first computing the unions for all leaves (being the objects themselves) and then merging unions towards the root, using the fact that

$$U_\delta = U_{\mathrm{lson}(\delta)} \cup U_{\mathrm{rson}(\delta)}.$$

Merging two unions is done by computing all intersections between their boundaries. Note that any such intersection point is necessarily a vertex of the overall union. This can be done using e.g. the red-blue intersection algorithm of Mairson and Stolfi [15] in time $O((u_{\text{lson}(\delta)} + u_{\text{rson}(\delta)}) \log n + u_\delta)$, where u_δ denotes the complexity of U_δ. For our purpose we can as well use the standard intersection algorithm by Bentley and Ottmann [3] (see also [23]) without increasing the overall asymptotic time complexity.

After computing the union U_δ at each node, we compute V_δ for all nodes in a top-down manner, starting at the root and working our way down the tree. The method is based on the following lemma:

Lemma 2.1 *The maps V_δ stored at nodes δ satisfy the following equations:*

$$V_{\text{root}} = U_{\text{root}}$$

$$V_{\text{lson}(\delta)} = V_\delta \cap U_{\text{lson}(\delta)}$$

$$V_{\text{rson}(\delta)} = V_\delta - U_{\text{lson}(\delta)} \ .$$

Proof. The first equation is easy, because the whole union of the set of objects is obviously visible in the sense defined above — there is no nearer object to hide it. The second equation follows from the fact that $U_{\text{lson}(\delta)}$ can only be covered by objects that also cover U_δ. Moreover, $U_{\text{lson}(\delta)}$ is a subset of U_δ. V_δ can be interpreted as the window through which we can see U_δ and, hence, the portions of $U_{\text{lson}(\delta)}$ that can be seen are exactly those that lie inside V_δ. The third equation follows from the fact that $V_\delta - U_{\text{lson}(\delta)}$ consists of those points of $U_{\text{rson}(\delta)}$ that are not hidden by objects stored to the left of δ or below lson(δ); by definition, these points constitute $V_{\text{rson}(\delta)}$. □

We apply this lemma to compute the regions V_δ, starting at the root and working our way down. To compute $V_{\text{lson}(\delta)}$ (resp. $V_{\text{rson}(\delta)}$) we simply compute the intersection (resp. difference) of V_δ and $U_{\text{lson}(\delta)}$ using any of the techniques above, say the red-blue intersection algorithm of [15]. This takes time $O((u_{\text{lson}(\delta)} + v_\delta) \log n + v_{\text{lson}(\delta)})$, where v_δ denotes the complexity of V_δ. (Note that in both cases any intersection between the boundaries of V_δ and $U_{\text{lson}(\delta)}$ must be a vertex of the resulting intersection or difference.)

This concludes the description of (the simpler version of) the algorithm. In the following section we will slightly modify the algorithm so as to reduce its working storage. After we have computed the regions V_δ at all nodes of the tree, we simply collect (and properly glue) the regions computed at the leaves, to construct the whole visibility map. Note that the algorithm is very simple and only requires as a subroutine an implementation of the red-blue intersection algorithm (or some other intersection algorithm like the one in [3]), suitable for computing unions, intersections, and differences between two regions in the plane.

3 Analysis of the algorithm

It immediately follows from the above description that the total time required for the algorithm, after the initial sorting and construction of the tree (which requires time $O(n \log n)$), is bounded by

$$\sum_{\delta} O((u_\delta + v_\delta) \log n) = O(\log n) \cdot (\sum_{\delta} u_\delta + \sum_{\delta} v_\delta) . \tag{1}$$

So we have to estimate both $\sum_\delta u_\delta$ and $\sum_\delta v_\delta$. As indicated in the introduction, we assume that the objects involved are "fat" in the sense that the complexity of the union of (the xy-projections of) any subset of n' objects is bounded by (the subquadratic function) $U(n')$ which we also assume to be super-additive. Now let n_δ denote the number of objects in the subtree rooted at δ. Then clearly

$$\begin{aligned} \sum_{\delta} u_\delta \quad \leq \quad & \sum_{\delta} U(n_\delta) = \sum_{d=0}^{\log n} \sum_{\delta \text{ at depth } d} U(n_\delta) = \\ & \sum_{d=0}^{\log n} O(U(n)) = O(U(n) \log n) . \end{aligned} \tag{2}$$

Estimating v_δ is slightly more complicated. The bound is based on the following lemma:

Lemma 3.1 *Any vertex of V_δ is a vertex of $V_{\delta'}$ for some leaf δ' in the subtree rooted at δ.*

Proof. V_δ has four different types of vertices: visible vertices of U_δ, visible intersections between the boundaries of U_δ and the projection of a nearer object, visible vertices of nearer objects that lie inside U_δ, and visible intersections between the (projections of the) boundaries of two nearer objects, which lie inside U_δ. All of these are obviously vertices of the final visibility map. It remains to show that there exists an object stored in the subtree rooted at δ, such that the intersection shows up as a vertex of the individual visibility map of the object. This claim is immediate for vertices of the first or second type, because each of them is either an original vertex of an object stored below δ, or the intersection of the boundary of such an object with the boundary of another higher object. For a vertex v of the third or fourth type, note that U_δ must be visible on some side of v in a sufficiently small neighborhood, which means that an object stored below δ is visible there. Hence v is a vertex of $V_{\delta'}$ for the leaf δ' that stores this object. $\square$

As stated above, the collection of maps V_δ over all leaves forms together the full visibility map. Moreover, as in the proof of the preceding lemma, it is easily verified that each vertex of the map can appear in at most two 'leaf-regions' V_δ. As a result we have:

$$\sum_{\delta \text{ a leaf}} v_\delta = O(k) \ .$$

It follows from the above lemma that the overall complexity of the maps V_δ on each level of the tree is also $O(k)$. Hence,

$$\sum_\delta v_\delta = \sum_{d=0}^{\log n} \sum_{\delta \text{ at depth } d} v_\delta = \sum_{d=0}^{\log n} O(k) = O(k \log n) \ . \tag{3}$$

This leads to the following result:

Proposition 3.2 *Given a set of n non-intersecting objects, such that the union of the projections on a viewing plane of any n' of them has complexity $U(n')$, where $U(n')$ is super-additive (and hopefully subquadratic), the visibility map of the objects can be computed in time $O((U(n) + k)\log^2 n)$.*

Proof. This follows immediately from equation (1), plugging in the results of equations (2) and (3). □

Remark. As noted earlier, this technique is rather general — it does not require the objects to be polyhedral, and it only requires a (known) depth ordering of the objects relative to the viewing point. It also applies when $U(n)$ is large, up to quadratic, except that the result is then much less significant.

It remains to analyze the amount of working storage required by the algorithm. Unfortunately, using the method as described above, the amount of required working storage becomes $O((U(n) + k)\log n)$. To reduce this we have to modify the method slightly.

First we construct the whole tree, together with the U_δ's for all nodes. All U_δ's at any particular level of the tree use $O(U(n))$ overall storage, so the total tree uses so far $O(U(n)\log n)$ storage. Next we recursively traverse the tree in preorder, computing the V_δ for all nodes, in the following way:

- if δ is a leaf, output V_δ; otherwise,
- compute $V_{\mathrm{lson}(\delta)}$ from $U_{\mathrm{lson}(\delta)}$ and V_δ;
- recursively treat the left subtree;
- remove $V_{\mathrm{lson}(\delta)}$ (it is no longer required);
- compute $V_{\mathrm{rson}(\delta)}$ from $U_{\mathrm{lson}(\delta)}$ and V_δ;
- recursively treat the right subtree;
- remove $V_{\mathrm{rson}(\delta)}$.

As a result, at any time during the algorithm we only store the regions V_δ along a single path of the tree, i.e., for at most $O(\log n)$ nodes. It remains to bound the size of one V_δ. Let U be the union of the projections of all the objects that lie nearer than U_δ (i.e. objects that are stored in the tree to the left of the subtree rooted at δ). Any vertex of V_δ is either a vertex of U_δ, or a vertex of U, or an intersection point between the boundaries of U_δ and U, and, hence, a vertex of $U_\delta \cup U$. The total number of these vertices is clearly bounded by $O(U(n))$. This leads to our main result:

Theorem 3.3 *Given a set of n non-intersecting objects in space and a viewing point z (that may be at infinity), such that there exists a known (and easily computable) depth ordering of the objects with respect to z, and such that the union of the projections of any n' of the objects on a viewing plane has complexity $U(n')$, where $U(n')$ is super-additive (and subquadratic), then the visibility map, as seen from z, can be computed in time $O((U(n)+k)\log^2 n)$, using $O(U(n)\log n)$ working storage.*

4 Applications

In this section we present the three applications mentioned in the introduction. In the first application we have a set of non-intersecting balls in space viewed from any fixed point. Computing the view of such a set can be reduced to computing the view from above of a set of horizontal disks. The best known result for output-sensitive hidden surface removal in such a set is due to Sharir and Overmars [26] who give a method that runs in time $O(n\sqrt{n}\log n + k)$. In the special case of unit disks considered in [21] a method is given that runs in time $O((n+k)\log^2 n)$. Here we apply our technique to obtain the same improved running time for the case of disks (or balls) of arbitrary radii.

To apply our method we need a bound on the union of a set of n (arbitrary) disks in the plane. It is well-known [14] that such a union has linear complexity, i.e., $U(n) = O(n)$. Now applying Theorem 3.3 we obtain:

Theorem 4.1 *Given a set of n non-intersecting balls in space, the view of this set from any fixed point can be computed in time $O((n+k)\log^2 n)$, using $O(n\log n)$ storage.*

Note that the bound $U(n) = O(n)$ applies also to *pseudodisks*, i.e., planar regions with the property that the boundaries of any pair of them intersect in at most 2 points. Hence the preceding theorem can be extended to the case of objects whose projections on the viewing plane behave like pseudodisks, assuming the shape of each object is not too complicated.

As an application of this extension, consider the case of a set of n non-intersecting convex homothetic objects (i.e., objects that are translated and scaled

Figure 3: A polyhedral terrain.

copies of a fixed convex object). Here again, the boundaries of the projections of any pair of the objects, in any view, intersect at most twice, so that the union has linear size. The depth ordering can be computed as in the case of balls or disks. Hence we have:

Theorem 4.2 *Given a set of n non-intersecting convex homothetic objects in space, the view of this set from any point can be computed in time $O((n+k)\log^2 n)$, using $O(n \log n)$ storage.*

Next consider a set of horizontal 'fat' triangles viewed from any fixed point. A set of triangles is called fat when there exists some positive constant θ such that any internal angle of the triangles is at least θ. For such a set of triangles it is proven by Matoušek et al. [16] that the union has complexity at most $O(n \log\log n)$. Note that the projections of a set of fat triangles need not in general be fat, but it is still the case that the union of any subfamily of n' of these projections has complexity $O(n' \log\log n')$. (To see this, project the triangles towards the viewing point, but make the viewing plane horizontal.) Hence, we can apply Theorem 3.3 to obtain the following result:

Theorem 4.3 *Given a set of n horizontal fat triangles, the view of this set from any fixed point can be computed in time $O((n \log\log n + k)\log^2 n)$, using $O(n \log n \log\log n)$ storage.*

Finally consider the case of a polyhedral terrain Σ with n faces, viewed from some fixed point a lying above it. A *polyhedral terrain* is the graph of a piecewise linear continuous function $z = \Sigma(x, y)$ (see figure 3). It has been shown in [8] that the faces of Σ can be ordered by depth with respect to a (although it might be necessary to cut some faces of Σ to ensure that the resulting order is indeed acyclic). Cole and Sharir [8] give an efficient technique for implicitly computing the visibility map. Reif and Sen [25] give an output-sensitive construction of the map that runs in time $O((n+k)\log n \log\log n)$. Their technique, which is based on dynamic ray-shooting in monotone polygonal chains, is fairly complicated. Using our much simpler algorithm we can obtain faster solutions.

To apply our technique, imagine that we replace Σ by a collection of semi-unbounded vertical prisms, each consisting of all points lying below a face of Σ. Obviously, the visibility map from a does not change by this transformation. The prisms have the fatness property, since the union of the projections of any n' of them has complexity $U(n') = O(n'\alpha(n'))$ (see [8] for details). We can thus apply Theorem 3.3 to the modified scene. In this case we can even improve the bound on the running time by a factor of $\log n$. Indeed, the regions U_δ and V_δ are all monotone polygons, and it is easily checked that each of the Boolean operations on them performed by the algorithm can be done in linear time. We thus have:

Theorem 4.4 *The visibility map of a polyhedral terrain consisting of* n *faces, viewed from some fixed point above it, can be computed in time* $O((n\alpha(n) + k)\log n)$ *and working storage* $O(n\alpha(n)\log n)$.

5 Conclusion

In this paper we have presented a new method for computing the visibility map of a set of non-intersecting objects in 3-space. It runs in time $O((U(n)+k)\log^2 n)$ and uses $O(U(n)\log n)$ working storage, where $U(n')$ is the maximum complexity of the union of the projections on a viewing plane of any subset of n' of the objects, and k is the complexity of the output visibility map. The method is quite simple, applies to general scenes of polyhedral or other objects, where a depth ordering of the objects is available, and is efficient whenever $U(n)$ is small. This is the case for sets of fat objects like disks (balls), fat triangles, homothets, and polyhedral terrains. This condition might also occur for many sets of non-fat objects. It is also worth noting that for any set of objects $U(n) = O(n + I)$ where I is the number of intersections in the projection. Hence, even for non-fat objects, the time bound is never worse than $O((n + I)\log^2 n)$ which is only a factor $\log n$ worse than the techniques in [27]. Although we did not exploit this observation, it is interesting to note that our technique also applies when the objects can be split into a small number of subfamilies so that within each subfamily the union complexity is small. An example where this observation can be applied is the case of axis-parallel horizontal rectangles (see [21] for details), although the resulting algorithm would be inferior to the best known solutions for this case.

In the preliminary version [13] of the paper, we also presented an alternative technique. Roughly speaking, it sweeps over all nodes of the tree $\mathcal{T}$ simultaneously, maintaining the cross sections of all the sets U_δ, V_δ with the sweepline. As things stand now, the alternative technique is considerably more complicated than the one given here, and yields exactly the same performance bounds. Still, there might be cases where the other technique becomes more advantageous. We refer the reader to [13] for more details.

Of course, the main open problem that remains is to find an output-sensitive algorithm that is efficient for general objects in space. Another open problem is to improve still further our technique. For instance, can the running time be reduced to $O((U(n) + k)\log n)$ (as in the case of polyhedral terrains)?

References

[1] P.K. Agarwal and J. Matoušek, Ray shooting and parametric search, *manuscript*, 1991.

[2] M. Bern, Hidden surface removal for rectangles, *J. Comp. Syst. Sciences* **40** (1990), 49–69.

[3] J.L. Bentley and T.A. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. Computers* **28** (1979), 643–647.

[4] M. de Berg, D. Halperin, M.H. Overmars, J. Snoeyink and M. van Kreveld, Efficient ray shooting and hidden surface removal, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 21–30.

[5] M.T. de Berg and M.H. Overmars, Hidden surface removal for axis-parallel polyhedra, *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 252–261.

[6] M.T. de Berg, M.H. Overmars and O. Schwarzkopf, Computing and verifying depth orders, *manuscript*, 1991.

[7] B. Chazelle, H. Edelsbrunner, L. Guibas, R. Pollack, R. Seidel, M. Sharir and J. Snoeyink, Counting and cutting cycles of lines and rods in space, *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 242–251.

[8] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symbolic Computation* **7** (1989), 11–30.

[9] F. Dévai, Quadratic bounds for hidden line elimination, *Proc. 2nd ACM Symp. on Computational Geometry*, 1986, pp. 269–275.

[10] M.T. Goodrich, A polygonal approach to hidden line elimination, *Proc. 25th Allerton Conf. on Communication, Control and Computing*, 1987, pp. 849–858.

[11] M.T. Goodrich, M.J. Atallah and M.H. Overmars, An input-size/output-size trade-off in the time-complexity of rectilinear hidden surface removal, *Proc. ICALP'90*, Springer-Verlag, Lecture Notes in Computer Science 443, 1990, pp. 689–702.

[12] R.H. Güting and T. Ottmann, New algorithms for special cases of the hidden line elimination problem, *Comp. Vision, Graphics and Image Processing* **40** (1987), 188–204.

[13] M.J. Katz, M.H. Overmars and M. Sharir, Efficient hidden surface removal for objects with small union size, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 31–40.

[14] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* **1** (1986), 59–71.

[15] H. Mairson and J. Stolfi, Reporting and counting intersections between two sets of line segments, *Theoretical Foundations of Computer Graphics and CAD*, R.A. Earnshaw, Ed., NATO ASI Series, Vol F-40, Springer Verlag, 1988, pp. 307–326.

[16] J. Matoušek, J. Pach, M. Sharir, S. Sifrony and E. Welzl, Fat triangles determine linearly many holes, Tech. Report 174/90, Eskenasy Institute of Computer Sciences, Tel Aviv University, May 1990. Also to appear in *Proc. 32nd IEEE Symp. on Foundations of Computer Science,* 1991.

[17] M. McKenna, Worst-case optimal hidden surface removal, *ACM Trans. Graphics* **6** (1987), 19–28.

[18] K. Mulmuley, An efficient algorithm for hidden surface removal, I, *Computer Graphics* **23** (1989), 379–388.

[19] O. Nurmi, A fast line-sweep algorithm for hidden line elimination, *BIT* **25** (1985), 466–472.

[20] M.H. Overmars and M. Sharir, Output-sensitive hidden surface removal, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 598–603.

[21] M.H. Overmars and M. Sharir, Merging visibility maps, *Computational Geometry, Theory and Applications* **1** (1991), 35–49.

[22] M. Paterson and F.F. Yao, Binary space partitions with applications to hidden surface removal and solid modeling, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 23–32.

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

[24] F.P. Preparata, J.S. Vitter and M. Yvinec, Computation of the axial view of a set of isothetic parallelepipeds, *ACM Trans. Graphics* **9** (1990), 278–300.

[25] J. Reif and S. Sen, An efficient output-sensitive hidden surface removal algorithm and its parallelization, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 193–200.

[26] M. Sharir and M.H. Overmars, A simple output-sensitive algorithm for hidden surface removal, *ACM Trans. Graphics*, 1991, to appear.

[27] A. Schmitt, Time and space bounds for hidden line and hidden surface algorithms, *Eurographics '81*, pp. 43–56.

[28] I.E. Sutherland, R.F. Sproull and R.A. Schumacker, A characterization of ten hidden-surface algorithms, *Computing Surveys* **6** (1974), 1–25.