An improved technique for output-sensitive hidden surface removal

Micha Sharir and Mark H. Overmars

RUU-CS-89-32
December 1989

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

An improved technique for output-sensitive hidden surface removal

Micha Sharir and Mark H. Overmars

Technical Report RUU-CS-89-32
December 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

An Improved Technique for Output-Sensitive Hidden Surface Removal*

Micha Sharir[†] Mark H. Overmars[‡]

December 20, 1989

Abstract

We derive a new output-sensitive algorithm for hidden surface removal in a collection of n triangles, viewed from a point z such that they can be ordered in an acyclic fashion according to their nearness to z. If k is the combinatorial complexity of the output *visibility map*, then we obtain a sophisticated randomized algorithm that runs in time $O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta})$, for any $\delta > 0$. The method is based on a new technique for tracing the visible contours using ray shooting.

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 non-intersecting polyhedral objects in 3-space, and a viewing point v, and our goal is to construct the visible parts of the given scene, as seen from v.

*Work by the first author was partially supported by the ESPRIT II Basic Research Actions Program of the EC, under contract No. 3075 (project ALCOM). Work by the second author 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 NCRD - the Israeli National Council for Research and Development, and the Fund for Basic Research in Electronics, Computers and Communication administered by the Israeli Academy of Sciences. A preliminary version of this paper appeared as part of the conference proceedings paper [15].

[†]School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, U.S.A.

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

Many solutions have been developed to date. The classical solutions from computer 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 [21]. In computational geometry a lot of work has been done in obtaining "object-space" solutions that 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 for e.g. 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. Most object-space hidden surface removal algorithms use these observations to compute the visible parts. They 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 $O(n^2)$ [5, 12]. More careful implementations run in time $O((n + I)\log n)$, where I denotes the number of intersections between the projected edges [8]. See also [11, 14, 19].

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 grid-like 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 [3, 9, 16]. Another output-sensitive algorithm has recently been proposed by Reif and Sen [17], for the special case of a polyhedral terrain.

Only recently some output-sensitive solutions have been proposed for the general problem. 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 [18] creates the view of the scene by adding the objects one

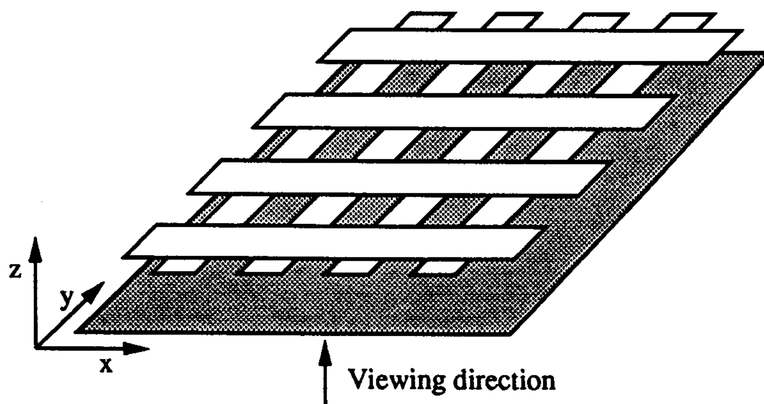


Figure 1: A scene with a small output size

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. The best worst-case output-sensitive result up to now was presented in a previous paper [20] and runs in time $O(n\sqrt{k}\log n)$, where k is the output size. This solution is quite simple. The only assumption that is made is that the objects can be ordered by "nearness" to the viewing point.

In this paper we continue our study and derive a more sophisticated and more efficient output-sensitive algorithm for the case our objects are triangles. Again we assume a depth ordering exists. In fact, we will only study the case of a set of horizontal triangles and a viewing point at $z = -\infty$. When a depth ordering exists the set of objects can always be transformed to this situation. We first present an initial coarse version of the algorithm. This version has no merits in itself — it is more complicated than the algorithm given in [20] and is actually inferior when the output size k is large. It uses a battery of sophisticated techniques, some of which use randomization,[1] recently developed in [10, 1, 2] for ray shooting and point location amidst a collection of intersecting segments in the plane. The randomized expected time complexity of the algorithm is $O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$, for any $\delta > 0$. After presenting geometric preliminaries in Section 2, we describe this algorithm in Section

[1] The randomization used by these algorithms does not depend on any probability distribution of the input triangles; the expected time bounds derived in this paper are over the internal randomization steps and hold for any given input.

3.

We then obtain in Section 4 an improved algorithm by partitioning the xy-plane into a large number of regions, and by solving the hidden surface removal problem, using the coarse algorithm, over each region separately. The partitioning depends on the structure of the arrangement of the projected triangle edges, and is chosen so as to obtain subproblems of small size. Using the deterministic partitioning algorithm of [1], we obtain a randomized algorithm whose expected running time is $O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta})$, for any $\delta > 0$.

Note that this bound is better than the bound on the running time of the algorithm in [20], as long as k is not too close to quadratic or too small. The improved algorithm is output-sensitive (unless k is smaller than roughly $n^{8/9}$). Unfortunately, the dependence on k is still very high. Hence, improvement might still be possible. We make a few comments on this issue in the concluding Section 5, where we also discuss open problems and suggest further possible attacks on this problem.

2 Geometric Preliminaries

Let $\mathbf{\Delta} = \{\Delta_1, \ldots, \Delta_n\}$ be a collection of n horizontal triangles in 3-D space, so that Δ_i lies in the plane $z = i$. The hidden surface removal problem for $\mathbf{\Delta}$ (and for a viewing point at $z = -\infty$) can be formulated as the problem of calculating the *lower envelope* of $\mathbf{\Delta}$, or, in other words, of partitioning the xy-plane into maximal regions so that for each region R there exists a unique triangle Δ (or no triangle at all) such that for all points $(x, y) \in R$, the lowest triangle lying above (x, y) is Δ (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 $\mathbf{\Delta}$.

Let us introduce a few notations. For a point $w \in R^3$ we denote its orthogonal xy-projection by $\pi(w)$. We also define, for a point $w \in R^3$, the *upward lifting* $\tau(w)$ of w to be the unique point w' such that w' lies strictly above w on some triangle of $\mathbf{\Delta}$, and the open vertical segment ww' meets no other triangle; if no triangle lies above w then $\tau(w)$ is undefined. We say that $\tau(w)$ is *visible from* w. We also say that a point w on some triangle is *visible* if it is visible from $\pi(w)$. Thus, thinking of the triangles in $\mathbf{\Delta}$ as being opaque, a point z on a triangle is visible from another point w if we can see z when standing at w and looking vertically upwards, and z is visible if we can see it when standing vertically below it on the xy-plane and looking directly upwards. We also say that triangle Δ_i *hides* a point z from another point w if w lies vertically below z and the segment wz intersects Δ_i; we say that Δ_i just hides z if it hides it from $\pi(z)$.

The visibility map M can be regarded as the xy projection of all visible points, where each projected point is labeled with its triangle. Plainly, each face of M is either the projection of a maximal connected visible portion of some triangle in $\mathbf{\Delta}$

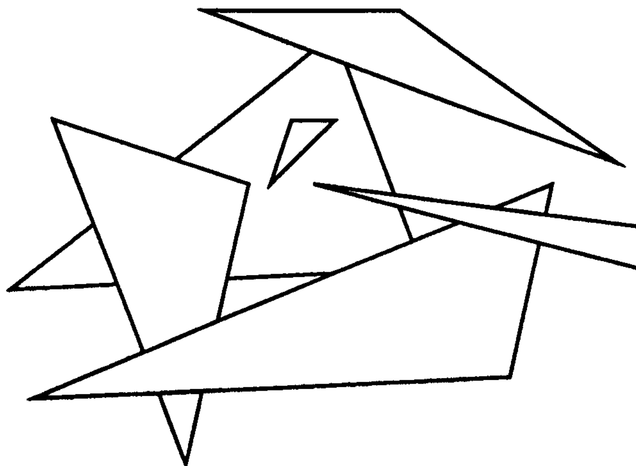


Figure 2: The visibility map

or a connected component of the complement of the union of the projections of all these triangles. The edges of M are visible connected portions of the triangle edges. Each vertex q of M is either $\pi(w)$ for some visible vertex w of one of the triangles in $\boldsymbol{\Delta}$ or the intersection of the projections of two triangle edges e, e' such that the vertical line passing through q meets these edges at the respective points w, w', with, say, w' lying above w, so that the segment qw' meets no other triangle (in other words, $\tau(q) = w$ and $\tau(w) = w'$). Let us assume that the triangles in $\boldsymbol{\Delta}$ lie in *general position*, meaning that no vertex of one triangle is co-vertical with an edge of another triangle, and that no three triangle edges have concurrent xy-projections. Note that in this case each vertex of M has degree 2 (if it is a projected triangle vertex) or 3 (in the latter case); the degree-3 vertices of M look like "T" junctions. See figure 2 for an illustration. Let k denote the number of vertices of M (the "output size"). Plainly $k = O(n^2)$, but it can be as small as 3 (when the lowest triangle hides all the others).

3 An Initial Coarse Output-Sensitive Algorithm

In this section we present a coarse output-sensitive technique for hidden surface removal. The algorithm uses randomization, and its expected time complexity is $O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$, for any $\delta > 0$; this makes the algorithm considerably inferior to the simpler algorithm reported in [20] when k is large. Nevertheless, we will later combine this algorithm with a partitioning scheme, that breaks the problem into subproblems of smaller size, and apply the algorithm of this section to

each subproblem separately. This will produce an improved solution, as detailed in the following section.

The best way to regard the algorithm presented in this section is as a high–level general technique, whose efficient implementation requires the availability of certain primitives, mainly for *ray shooting* and *point location* in a planar arrangement of overlapping triangles. The current version of the paper exploits the best known solutions for these problems. However, a lot of work is currently underway to improve these techniques, and we expect that such improvements will automatically lead to improved performance of the algorithm presented below.

3.1 An overview of the algorithm

The basis for the algorithm is the following simple yet crucial observation.

Lemma 3.1 *Let E denote the union of all edges of the visibility map M. Then each connected component of E contains a projected triangle vertex.*

Proof: Simply take the leftmost point p in the connected component (when there is more than one leftmost point, take the bottommost). Clearly, p is a vertex of M. There are only three different types of vertices in M (the points z in figure 3). Clearly, only when p is a vertex of a triangle, there is no other point to the left of p. □

Lemma 3.1 suggests the following high-level approach for obtaining an output-sensitive hidden surface removal algorithm.

1. Find all visible vertices of the triangles in $\mathbf{\Delta}$.
2. For each visible vertex v, construct the connected component E_v of E containing $\pi(v)$, by repeated "ray-shooting" along the edges of E_v.

Let us define the ray-shooting operation in more detail. At each ray-shooting step we are given an edge e of some triangle Δ, a point w on e, and a direction along e, with the property that if we follow e from w in this direction, we traverse a visible portion of e. The goal of the ray-shooting query is to follow this visible portion of e until we reach the first of one of the following types of points (see figure 3 for an illustration):

(i) a vertex z of e;

(ii) a point z on e at which e is hidden by some lower triangle;

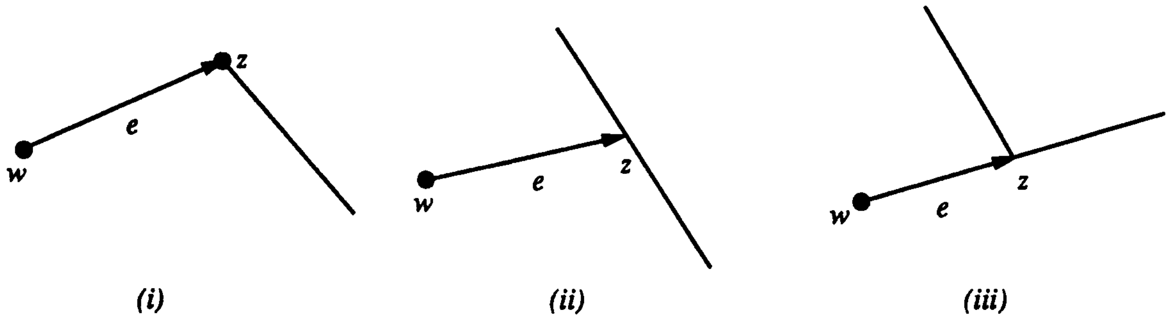


Figure 3: Ray shooting for hidden surface removal

(iii) a point z on e at which e hides another triangle edge e' lying above z (so that, in the vicinity of $\pi(z)$, an appropriate portion of e' is visible).

In each of these cases we find a new vertex $\pi(z)$ of M. Moreover, in cases (ii) and (iii) we also require from the ray shooting procedure to report the other edge e' that hides e or is hidden by e at z, and to determine which direction(s) along e' make it visible near $\pi(z)$. Thus, having found z, e' and these directions, we can then repeat the ray shooting procedure along e' in the given direction(s) from the vertical projection of z onto e'. Initiating these shootings at each visible triangle vertex along the two edges incident to it, and continuing it until no new visible edges are obtained, we discover in this way the entire visibility map M, as follows trivially from Lemma 3.1. (Some care has to be taken that parts are not reported more than once. This can easily be done by marking vertices and parts of edges visited and only traversing along vertices and edges that are not marked.)

We will show that finding all visible triangle vertices can be accomplished in time $O(n^{4/3}\log^{2.89} n)$, and that the triangles of Δ can be preprocessed in randomized expected time $O(n \log n)$ into a data structure of linear size, so that each ray shooting query can be executed in time $O(n^{2/3+\delta})$, for any $\delta > 0$. Putting all of these bounds together, we obtain the main result of this section.

Theorem 3.2 *One can perform hidden surface removal for n horizontal triangles in 3-space in randomized expected time*

$$O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$$

for any $\delta > 0$, where k is the output size of the problem.

Details of efficient implementation of the various steps of the algorithm are given in the following two subsections. In subsection 3.2 we show how to find efficiently all the visible vertices of the given triangles; as a matter of fact, we give a more general procedure for finding the triangle lying immediately above any query point in 3-space. Subsection 3.3 presents an efficient solution of the ray shooting problem.

3.2 Vertical visibility queries

In this subsection we solve the following problems

(i) Given a collection $\mathbf{\Delta}$ of horizontal triangles in 3-space as above, preprocess it so that, given any point w in 3-space, we can determine efficiently the triangle lying immediately above w (i.e., the triangle containing $\tau(w)$), or determine that no such triangle exists.

(ii) Given a collection $\mathbf{\Delta}$ as above, and a collection S of m points on the xy-plane, determine for each point the triangle lying immediately above it or else report that no such triangle exists.

Clearly, solving the second problem would immediately yield a solution to the problem of finding all visible vertices — apply it to the set of all projected triangle vertices $\pi(v)$; a vertex v of a triangle Δ is visible if and only if the triangle lying immediately above v is Δ itself. The first variant of the problem will be required in our solution of the ray-shooting problem, to be described in the following subsection.

Fortunately for us, both problems are easy to solve using as a main component the following "implicit point location" technique, which has been developed in [10], [1]. Although the technique in [1] has a faster query time, we will use for the first problem the alternative technique of [10], because it requires less preprocessing time and storage, and this will turn out to be an important factor in the analysis of the algorithm in Section 4 below.

Specifically, it is shown in [10] that, given n triangles in the plane, we can preprocess them in randomized expected time $O(n \log n)$ into a data structure of linear size, so that, given any query point a, we can determine in $O(n^{2/3+\delta})$ time whether a lies in the union of the given triangles; here δ is an arbitrarily small positive constant, and the constants of proportionality in the above bounds depend on δ.

To solve our original problem (i), we next construct a perfectly balanced tree $\mathcal{T}$ storing the triangles of $\mathbf{\Delta}$ in its leaves, sorted in increasing height. For each node ξ of $\mathcal{T}$, let $\mathcal{T}_\xi$ denote the subtree of $\mathcal{T}$ rooted at ξ, and let $\mathbf{\Delta}_\xi$ denote the corresponding subcollection of triangles stored at the leaves of $\mathcal{T}_\xi$. For each node ξ, preprocess the xy-projections of the triangles in $\mathbf{\Delta}_\xi$ for implicit point location queries.

Now, given a query point w, we find the subset $\{\Delta_j, \ldots, \Delta_n\}$ of triangles in $\mathbf{\Delta}$ whose height is greater than the height of w, and represent this subset as the union of subsets $\mathbf{\Delta}_\xi$ for $O(\log n)$ nodes ξ of $\mathcal{T}$. (These are the nodes bordering the search path in $\mathcal{T}$ towards Δ_j to the right.) We now perform a binary search over this sequence of triangles, as follows. First we go over the subcollections $\mathbf{\Delta}_\xi$, in order of increasing height of their triangles, until the first ξ for which $\pi(w)$ is in the union of the projections of the triangles in $\mathbf{\Delta}_\xi$. Then we search through the tree $\mathcal{T}_\xi$. At each node η along the search path we test whether $\pi(w)$ lies in the union of the projections of the triangles in $\mathbf{\Delta}_{\eta'}$, where η' is the left child of η. If so, we continue the search at η'; otherwise, we continue the search at the right child of η. In this manner, after $O(\log n)$ queries, we obtain the triangle lying immediately above w, or determine that no such triangle exists. The overall cost of a query is easily seen to be $O(n^{2/3+\delta})$. The overall expected preprocessing time is $O(n \log^2 n)$ and the space is $O(n \log n)$. However, as observed in [10] (see also [6]), we can decrease the preprocessing time to $O(n \log n)$ and the space to linear, without affecting the asymptotic bound on query time, by building the data structures for implicit point location only once every t levels of $\mathcal{T}$, for an appropriate parameter t.

As to the second problem (ii), we apply Agarwal's technique [1] as follows. We construct the tree $\mathcal{T}$ as above, and the run a "simultaneous binary search" through $\mathcal{T}$ with all the given points. Specifically, we project on the xy plane all triangles in $\mathbf{\Delta}_{\xi_L}$, where ξ_L is the left child of the root of $\mathcal{T}$—that is, the lower half of the triangles of $\mathbf{\Delta}$. We then test which of the points of S lie in the union of these projected triangles, using the batched implicit point location algorithm of [1]; this requires $O(n^{4/3} \log^{2.89} n)$ time. Let S_1 be the resulting subset of S and let S_2 be the remainder of S. Note that for any point in S_1 the triangle lying immediately above it must belong to $\mathbf{\Delta}_{\xi_L}$ whereas for a point in S_2 this triangle (if it exists) belongs to the higher half $\mathbf{\Delta}_{\xi_R}$, where ξ_R is the right child of the root of $\mathcal{T}$. We thus recurse twice, once with $\mathbf{\Delta}_{\xi_L}$ and S_1, and once with $\mathbf{\Delta}_{\xi_R}$ and S_2. It is shown in [1] that the overall time of this procedure is still $O(n^{4/3} \log^{2.89} n)$.

3.3 The ray shooting procedure

Let us first consider the following auxiliary problem: Given n triangles $T_1, \ldots, T_n$ in the xy plane, preprocess them so that, given any query ray ρ, we can find efficiently the first intersection of ρ with any triangle edge, or determine that no such intersection exists.

This problem has recently been studied in [10, 2]. Again, although the best query time known to date is due to Agarwal [2], we will use the alternative algorithm of [10], which requires $O(n \log n)$ randomized expected preprocessing time, linear storage, and $O(n^{2/3+\delta})$ query time, for any $\delta > 0$.

Let us now return to our original 3-D ray shooting problem. We are given an

edge e of some triangle Δ_i, a point w on e, and a direction along e from w. Our goal is twofold: to find the first edge that hides e (as we follow e from w in the specified direction), and to find the first visible edge that lies above e and is hidden by e. The answer to the query is the nearest of these two events, or the appropriate endpoint of e if it is reached before any of these events takes place. The solutions to these two subproblems are very similar, and we will consider them separately.

To facilitate both solutions, we first construct the binary tree $\mathcal{T}$ as in the previous subsection. For each node ξ of $\mathcal{T}$, we preprocess for planar ray shooting the xy projections of the triangles in $\mathbf{\Delta}_\xi$, as defined above. As argued at the end of the preceding subsection, this can be done in overall $O(n \log n)$ randomized expected preprocessing time and linear space (over all nodes ξ of $\mathcal{T}$), without affecting the asymptotic bound on query time.

Consider now the first subproblem, of finding the first edge that hides e. Take the collection of triangles that are lower than Δ_i, i.e. $\{\Delta_1, \ldots, \Delta_{i-1}\}$, and represent it as the union of subcollections $\mathbf{\Delta}_\xi$, for $O(\log n)$ nodes ξ of $\mathcal{T}$. For each such node ξ, shoot along the projection $\pi(e)$ of e from $\pi(w)$ in the specified direction, to obtain the first intersection of $\pi(e)$ with a projected edge of a triangle in $\mathbf{\Delta}_\xi$. Let the nearest of all these intersections be q. If q lies further away from $\pi(w)$ than the relevant endpoint s of $\pi(e)$, then no edge hides this portion of e. Otherwise it is easily verified that $\tau(q)$ lies on the first visible edge that hides this portion of e. The total cost of all these ray shootings is easily seen to be $O(n^{2/3+\delta})$.

The second subproblem, of finding the first visible edge that e hides, is only slightly more complicated. We first find the triangle Δ_j lying immediately above w, using the technique described in the previous subsection. We then apply the ray-shooting technique described in the previous paragraph to the collection of triangles $\{\Delta_{i+1}, \ldots, \Delta_j\}$ (or $\{\Delta_{i+1}, \ldots, \Delta_n\}$ if no triangle lies above w). Let q be the nearest point returned by the above procedure. If the endpoint s of the traversed portion of e lies nearer to w than q then e hides no other visible edge, otherwise q is the first point (after w) where the traversed portion of e hides an edge. Note that in the last case, the edge that e hides at q is not necessarily visible, because e could already be invisible at this point. The complete ray shooting procedure proceeds by performing the two types of searches – what is the first edge that e hides and what is the first edge that hides e. The nearest of these events to w (if it occurs before the endpoint of e) is the point we seek, and the procedure has enough information to report also which of the two types of hiding occurs at that point, which edge hides e or is hidden by e at this point, and which direction(s) along that edge make it visible in the vicinity of the corresponding vertex of M. The overall cost of a ray-shooting query is thus $O(n^{2/3+\delta})$, for any $\delta > 0$.

4 An Improved Partition-Based Algorithm

In this section we improve the algorithm presented in the previous section using the following fairly simple idea. We partition the xy-plane into triangular regions, called *base cells*, with the properties that

(i) the number of regions is small, and

(ii) each region is crossed by only a small number of projected triangle edges (so most of the triangle projections are either disjoint from, or fully contain, that region).

This partitioning allows us to break the original hidden surface removal problem into a small number of subproblems, one per each base cell, and each subproblem involves only a small number of triangles. We then apply the algorithm given in the preceding section to each subproblem separately, and combine the outputs to obtain the overall visibility map. There are two important characteristics of this approach. First, the total output size is (almost) the sum of the output sizes of the subproblems. Second, because the size of each subproblem is small, the overhead per vertex of the visibility map becomes smaller. (This partitioning trick has also been proposed as a heuristic for image-space techniques, but without explicit quantification of the improvement in complexity that it entails.)

Nevertheless, the partitioning raises several new technical difficulties. Each of these needs to be solved efficiently, or else the entire scheme would collapse. We describe below in detail each of the new procedures that are required in order to overcome these difficulties.

Partitioning

Let $r \le n$ be some integer parameter, whose value will be chosen later. Let G be the collection of the xy-projections of the $3n$ triangle edges. Using the partitioning algorithm of Agarwal [1], we can partition the xy plane into $m = O(r^2)$ base cells $T_1, \ldots, T_m$, such that each T_ℓ meets at most $O(n/r)$ projected edges; the time complexity of the partitioning is $O(nr \log n \cdot \log^\beta r)$, for some constant $\beta < 3.33$, i.e. $O(nr \log^{4.33} n)$.

This partitioning allows us to break the original hidden surface removal problem into $O(r^2)$ subproblems, one per each base cell. For each such T_ℓ there are only $O(n/r)$ triangles Δ_i whose projected boundaries intersect T_ℓ; the projection of every other triangle in $\mathbf{\Delta}$ either completely contains T_ℓ or is disjoint from T_ℓ. Moreover, let $\Delta_{i(\ell)}$ be the lowest triangle whose xy-projection fully contains T_ℓ; then to solve the hidden surface removal problem over T_ℓ it suffices to process only the triangles lower than $\Delta_{i(\ell)}$ whose projected boundaries intersect T_ℓ. We next describe an efficient procedure for calculating $\Delta_{i(\ell)}$ for all cells T_ℓ, in overall time $O(nr \log r)$.

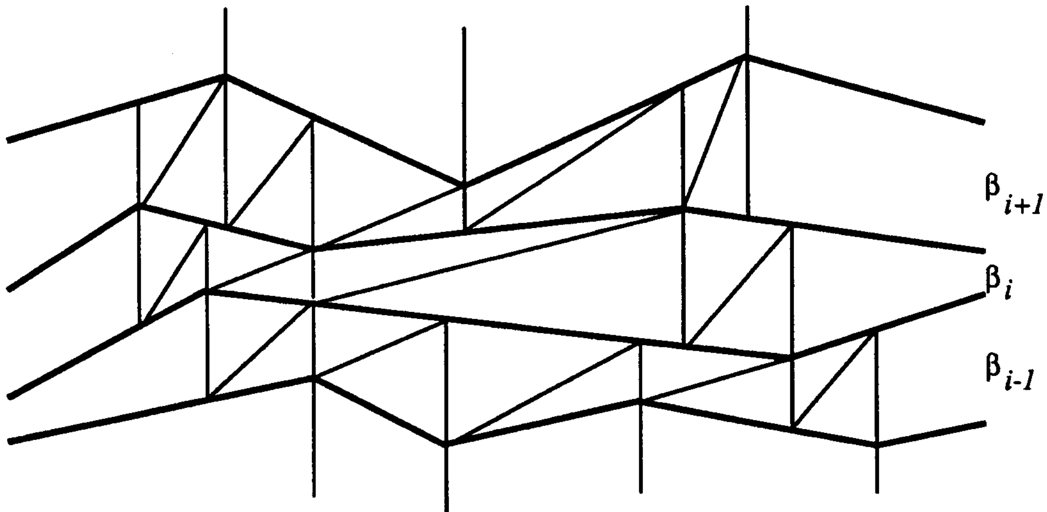


Figure 4: The base cells in a partitioning

Finding lowest "covering triangles"

Our goal is to find, for each base cell T_ℓ, the lowest triangle $\Delta_{i(\ell)}$ that fully contains T_ℓ. We note that the partition produced in [1] can be represented as a collection of r "belts", $\beta_1, \ldots, \beta_r$, separated from each other by $r-1$ x-monotone polygonal chains, so that within each belt the base cells are linearly ordered from left to right, and are separated from each other by vertical segments drawn from the vertices of the two chains bounding the belt, and by diagonals of the resulting trapezoidal cells. See figure 4 for an illustration.

We represent each belt β_i by a balanced binary tree Q_i storing in its leaves the base cells of the belt in their left-to-right order.

Consider the effect of a single triangle $\Delta \in \mathbf{\Delta}$ on Q_i. Suppose the edges of Δ intersect $j = j_{i,\Delta}$ base cells in β_i. Removing these cells from β_i partitions Q_i into at most $j+1$ contiguous portions such that the cells in each portion are either all disjoint from Δ or all contained in Δ. If β_i contains q_i base cells, then we can easily represent the portions of β_i that contain cells fully covered by Δ as a disjoint union of $O(j_{i,\Delta} \log q_i)$ subtrees, and, in time $O(\log q_i + j_{i,\Delta} \log q_i)$, we can store Δ at the roots of these subtrees. Repeating this step for each belt β_i and for each triangle in $\mathbf{\Delta}$, and observing that the sum of all the corresponding quantities $j_{i,\Delta}$ is $O(nr)$, we can "encode" all triangles of $\mathbf{\Delta}$ in this manner, in total time $O(nr \log r)$. In this encoding, we follow the rule that whenever more than one triangle Δ is stored at some node of the tree representing β_i, only the lowest of these triangles is maintained there. Now, for each T_ℓ, we find the tree Q_i and its leaf ξ where T_ℓ is stored, and

walk along the search path in Q_i to ξ. The lowest triangle Δ encountered along that path is the desired $\Delta_{i(\ell)}$. Clearly the overall cost of this procedure is $O(nr \log r)$.

Clipping the visibility map over base cell edges

We next wish to apply, for each T_ℓ, the algorithm described in the preceding section to the corresponding collection of $O(n/r)$ triangles of $\mathbf{\Delta}$ "partially overlapping" T_ℓ and lying below $\Delta_{i(\ell)}$. This however requires some extra care. To begin with, each such triangle has to be clipped to its portion lying above T_ℓ. If we denote by M_ℓ the portion of the visibility map M over T_ℓ, and let k_ℓ denote the number of vertices of M_ℓ within the interior of T_ℓ, then plainly $\sum_\ell k_\ell \leq k$. However, M_ℓ may have additional vertices and edges over the edges of T_ℓ, caused by the clipping of the relevant triangles. To find these features is easy - simply observe that the lower envelope of the triangles of $\mathbf{\Delta}$ over an edge of T_ℓ is just an envelope of $O(n/r)$ horizontal line segments, which has plainly only $O(n/r)$ features, and is easily computed in time $O((n/r)\log(n/r))$. However, the presence of these features as part of M_ℓ raises the following subtle problem.

Eliminating redundant ray-shootings

Lemma 3.1 can be easily replaced by a similar claim, asserting that it suffices to start the ray shooting process from all visible triangle vertices and from all visible vertices of M_ℓ lying above the edges of T_ℓ. However, it is imperative that when we shoot from a vertex of the latter type, we do it efficiently, because, as will be apparent from subsequent analysis, we can afford to pay $O((n/r)^{2/3+\delta})$ time for such a shooting only if it will produce a real vertex of M (i.e. not a "clipped" vertex lying over an edge of T_ℓ); see figure 5. In other words, we need a more efficient preliminary procedure to find which ray shootings from the edges of T_ℓ are worth performing. Using tricks similar to those in [2], we can obtain such a "sifting" procedure, having overall cost $O(n^{4/3} \log^{2.89} n + n r \alpha(n) \log^3 n)$.

Specifically, we fix a base cell T_ℓ and partition the set of triangle edges that meet T_ℓ into two subsets, the set $\mathcal{G}$ of "long" edges that meet the boundary of T_ℓ and the set $\mathcal{S}$ of "short" edges that are fully contained in the interior of T_ℓ.

Consider first a simpler problem: Given sets $\mathcal{G}$, $\mathcal{S}$ as above, where all segments in $\mathcal{G}$ are assumed to be clipped to within T_ℓ, and another collection $\mathcal{L}$ of lines that intersect T_ℓ, find which lines of $\mathcal{L}$ intersect at least one segment in $\mathcal{G} \cup \mathcal{S}$. Put $g = |\mathcal{G}|$, $s = |\mathcal{S}|$, and $m = |\mathcal{L}|$.

We will separately test for intersection with $\mathcal{G}$ and with $\mathcal{S}$. For intersection with $\mathcal{G}$, we first construct the unbounded face of the arrangement of the clipped segments of $\mathcal{G}$, and partition it into subfaces by adding the edges of T_ℓ (this in effect produces the *zone* of the boundary of T_ℓ in this arrangement; see figure 6). We next process each resulting subface for logarithmic-time ray shooting, as in [4]; note that each such subface is simply connected, so this is indeed possible. Finally, we shoot along each line in $\mathcal{L}$ from its intersections with ∂T_ℓ into the corresponding subface. If the

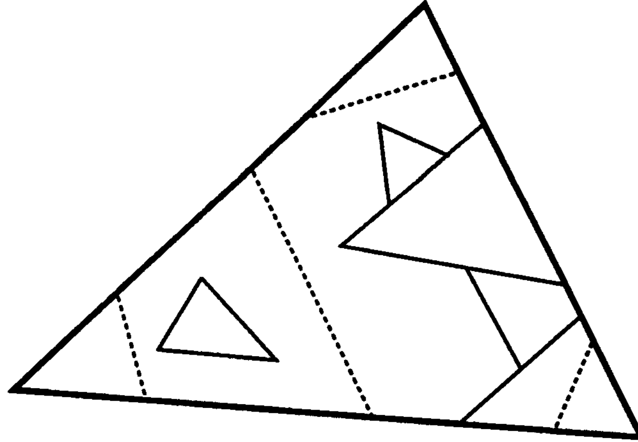


Figure 5: The dotted edges are spurious ray shootings

shooting hits a segment of $\mathcal{G}$ then an intersection has been detected. Otherwise it will hit another edge of T_ℓ, in which case it is clear that no intersection with $\mathcal{G}$ exists.

The time required for this step is

$$O(g\alpha(g)\log^2 g + m\log g)$$

where the first term bounds the time needed to construct the unbounded face of the arrangement of long edges [7], which also dominates the time needed to preprocess the resulting zone for fast ray shooting, and the second term bounds the cost of the ray shootings.

To detect intersection with $\mathcal{S}$, we pass to the dual plane, so $\mathcal{L}$ is mapped to a set $\mathcal{L}^\star$ of points, and each segment in $\mathcal{S}$ is mapped into a double wedge. A line l of $\mathcal{L}$ meets a segment e of $\mathcal{S}$ if and only if the dual point $l^\star$ lies in the double wedge $e^\star$ dual to e. The problem thus becomes a special case of the "implicit point location" problem discussed in [10], [1]. That is, given m points and s double wedges, determine which points lie in the union of the double wedges. The best algorithm for solving this problem is due to Agarwal [1], and it runs in time $O(m^{2/3}s^{2/3}\log^{2.89} s + (s+m)\log s)$.

Let us now return to the actual problem of detecting redundant ray-shootings from the boundary of T_ℓ. Let p be a point lying on an edge e of some triangle $\Delta_i \in \mathbf{\Delta}$ and visible from an edge of T_ℓ. The processing of the visibility map along the edges of T_ℓ can also determine easily the triangle Δ_j of $\mathbf{\Delta}$ lying immediately above p. Hence the ray-shooting from p along e into T_ℓ will be non-redundant if and only if the projection of e meets, within T_ℓ, another projected edge of some triangle Δ_h, for $h \leq j$ or e ends in T_ℓ. The second case is easily tested in constant time. For

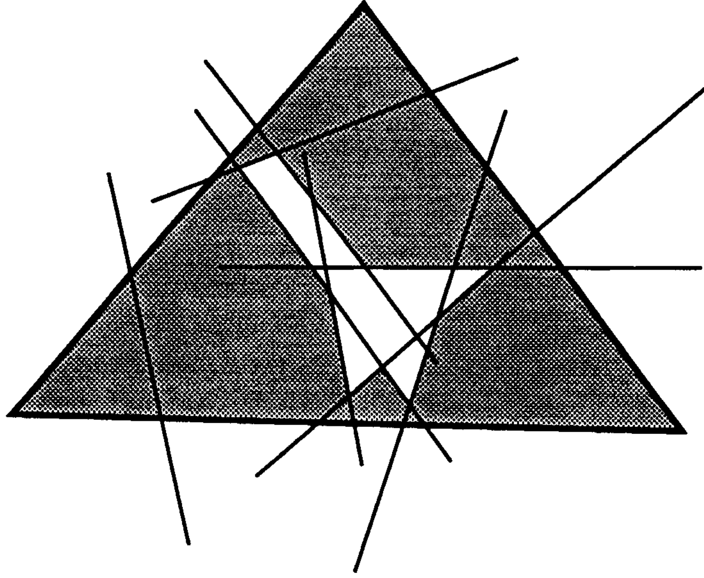


Figure 6: The zone of T_ℓ in the arrangement of the long edges

the first case we proceed as follows.

Let $\mathcal{L}$ denote the collection of lines containing the projections of edges e as above, and associate with each line $l \in \mathcal{L}$ an index $\phi(l) = j$, where j is the index of the triangle lying immediately above p, as defined above. Partition the set of triangles that partially overlap T_ℓ (and lie below $\Delta_{i(\ell)}$) into two roughly equal subsets, $\mathbf{\Delta}_1^{(\ell)}$, $\mathbf{\Delta}_2^{(\ell)}$, such that the triangles in $\mathbf{\Delta}_1^{(\ell)}$ lie lower than the triangles in $\mathbf{\Delta}_2^{(\ell)}$. Similarly partition $\mathcal{L}$ into two subsets, $\mathcal{L}_1$, $\mathcal{L}_2$, such that the index $\phi(l)$ of a line $l \in \mathcal{L}_1$ (resp. $l \in \mathcal{L}_2$) corresponds to a triangle that lies in $\mathbf{\Delta}_1^{(\ell)}$ (resp. in $\mathbf{\Delta}_2^{(\ell)}$). We then test for intersection between $\mathcal{L}_2$ and the projections of the triangles in $\mathbf{\Delta}_1^{(\ell)}$, using the procedure outlined above, and then recursively test for non-redundancy for shooting along lines of $\mathcal{L}_1$ amidst the triangles of $\mathbf{\Delta}_1^{(\ell)}$, and similarly for shooting along lines of $\mathcal{L}_2$ amidst the triangles of $\mathbf{\Delta}_2^{(\ell)}$. It is easily checked that the shooting along a line $l \in \mathcal{L}$ is non-redundant if and only if an intersection is detected along l at some recursive level of this procedure.

Since in our case $|\mathcal{G}|, |\mathcal{L}| = O(n/r)$ and the recursion has depth only $O(\log(n/r))$, it is easily verified that the cost of this procedure is

$$O\left(\frac{n}{r}\alpha(n)\log^3 n + \left(\frac{n}{r}\right)^{2/3} s_\ell{}^{2/3}\log^{2.89} n\right)$$

where $s_\ell = |S|$. Summing this over all base cells T_ℓ, using Hölder's inequality, and noting that $\sum_\ell s_\ell = O(n)$, we obtain an overall cost of

$$O(nr\alpha(n)\log^3 n + n^{4/3}\log^{2.89} n) .$$

Calculating visible vertices

Let us now return to the overall algorithm. Another step required there is to find all visible triangle vertices. An analysis similar to the one carried out above shows that the overall cost of this step is $O(n^{4/3}\log^{2.89} n)$, if performed separately within each base cell. Alternatively, we could compute all visible vertices *before* the partitioning, and then distribute them among the base cells. This requires the same amount of time asymptotically.

Putting it all together

Having all this machinery at hand, we can now apply the algorithm of Section 3 to each base cell separately. The (net) randomized expected time needed to solve the ℓ-th subproblem, exclusive of the various preparatory steps described above, is thus $O(k_\ell(n/r)^{2/3+\delta})$. We now sum this over all $O(r^2)$ subproblems, and add the cost of all the other steps, namely the partitioning, computing visible vertices, computing the "covering triangles" $\Delta_{i(\ell)}$, computing M over edges of the cells, and sifting out redundant ray shootings. We thus obtain the overall bound

$$O\left(n^{4/3}\log^{2.89} n + nr\log^{4.33} n + (\sum_\ell k_\ell)\cdot\left(\frac{n}{r}\right)^{2/3+\delta}\right) .$$

But $\sum_\ell k_\ell = k$. Now choose

$$r = \frac{k^{\frac{3}{5(1+3\delta/2)}}}{n^{1/5}} .$$

It now can easily be checked that this yields the bound

$$O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta'})$$

for a different, but still arbitrarily small, $\delta' > 0$.

Guessing the value of k

One last tricky issue remains. Since we do not know k in advance, how can we choose r to be a function of k? This is handled by guessing the value of k, as follows. Start with $k = n^{8/9}$ and run the algorithm as described above. If the time of the algorithm begins to exceed the bound given above then we know that the actual value of k is larger. In this case we double the guessed value of k and rerun the algorithm. This is continued until the time alloted for the execution of the algorithm is not exceeded, in which case the algorithm will complete the calculation of M as required. (A final note of caution: The preprocessing for ray shooting, using the technique of [10], is randomized. However, we can tune it so that it verifies that

the random sample drawn is one that guarantees the desired query time; if the sample is bad, it is discarded and another sample is chosen. In this manner, when we execute the various hidden surface removal procedures described above, we know what time complexity to expect, if our choice of r is the correct one.) The overall time complexity of the algorithm is easily seen to be asymptotically the same as the complexity of the last step, namely

$$O(n^{4/3} \log^{2.89} n + k^{3/5} n^{4/5+\delta'})$$

for any $\delta' > 0$.

Theorem 4.1 *One can perform hidden surface removal for n horizontal triangles in randomized expected time*

$$O(n^{4/3} \log^{2.89} n + k^{3/5} n^{4/5+\delta'})$$

for any $\delta' > 0$.

A comment worth making is that the algorithm is not very efficient when k is really small, that is much smaller than n, because the overhead of preparing the data structures becomes too large. We can handle this problem by first executing the simple algorithm of [20], and stop it as soon as it has detected more than $n^{2/3}$ output vertices. If k is smaller than this, the first algorithm will run to completion in time $O(n\sqrt{k} \log n) = O(n^{4/3} \log n)$. Otherwise, we run the algorithm presented above. In this way the total amount of time used becomes

$$\min\{O(n\sqrt{k} \log n), O(n^{4/3} \log^{2.89} n + k^{3/5} n^{4/5+\delta'})\}$$

Remark: When k is between roughly $n^{2/3}$ and $n^{8/9}$, the algorithm (as just modified) is a bit insensitive to k, because it runs in time $O(n^{4/3} \log^{2.89} n)$, which does not take advantage of k being smaller than $n^{8/9}$. Can this gap be "closed"?

5 Conclusion

We have presented a new output-sensitive hidden surface removal algorithm that is faster than previous solutions. The method is based on a large number of different techniques including a method of tracing the visibility map using ray shooting, partitioning techniques, implicit point location and a method for guessing the output size. The time complexity of the algorithm depends on the performance of the ray shooting and point location techniques. If these problems could be solved using close to linear (randomized expected) preprocessing time and close to $O(n^{1/2})$ query time, then the dependence of the above algorithm on k would come close to $O(k^{2/3} n^{2/3})$.

We conjecture that such complexity can be attained, and pose this as a challenging open problem.

The results obtained in this paper also raise several related open problems. A first problem deals with dynamically maintaining the visibility map under insertions or deletions of objects or when moving the point of view. Almost no results are known for this problem. A second questions concerns generalizations of the method. As noted the same technique can be used for arbitrary collections of triangles as long as no cyclic overlap occurs, using a simple transformation. The method can also easily be adapted to simple polygons of bounded degree (rather than triangles only). The method will be exactly the same. Unfortunately, the techniques for ray shooting and implicit point location no longer work when objects have curved boundaries. Finally, can we use the new ideas developed here to obtain a subquadratic solution to the following problem: Given n triangles in the plane, does their union completely cover some specified region, e.g. the unit square?

References

[1] P.K. Agarwal, A deterministic algorithm for partitioning arrangements of lines and its applications, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 11-22.

[2] P.K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 315-325.

[3] M. Bern, Hidden surface removal for rectangles, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 183-192.

[4] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 135-146.

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

[6] D. Dobkin and H. Edelsbrunner, Space searching for intersecting objects, *J. Algorithms* **8** (1987), 348-361.

[7] H. Edelsbrunner, L. Guibas and M. Sharir, The complexity of many faces in arrangements of lines and of segments, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 44-55.

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

[9] 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, Tech. Rept. RUU-CS-89-24, Dept. of Comp. Science, Utrecht University, 1989.

[10] L. GUIBAS, M. OVERMARS AND M. SHARIR, Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques, *Proc. SWAT 88*, Lecture Notes in Comp. Science 318, Springer Verlag, 1988, pp. 64-73.

[11] R.H. GÜTING AND T. OTTMAN, New algorithms for special cases of the hidden line elimination problem, *Comp. Vision, Graphics and Image Processing* **40** (1987), 188-204.

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

[13] K. MULMULEY, An efficient algorithm for hidden surface removal, *Computer Graphics* **23** (1989), 379-388.

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

[15] M.H. OVERMARS AND M. SHARIR, Output-sensitive hidden surface removal, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 598–603.

[16] F.P. Preparata, J.S. Vitter and M. Yvinec, Computation of the axial view of a set of isothetic parallelepipeds, Techn. Rep. LIENS-88-1, Lab. d'Informatique de L'Ecole Normal Supérieure, 1988.

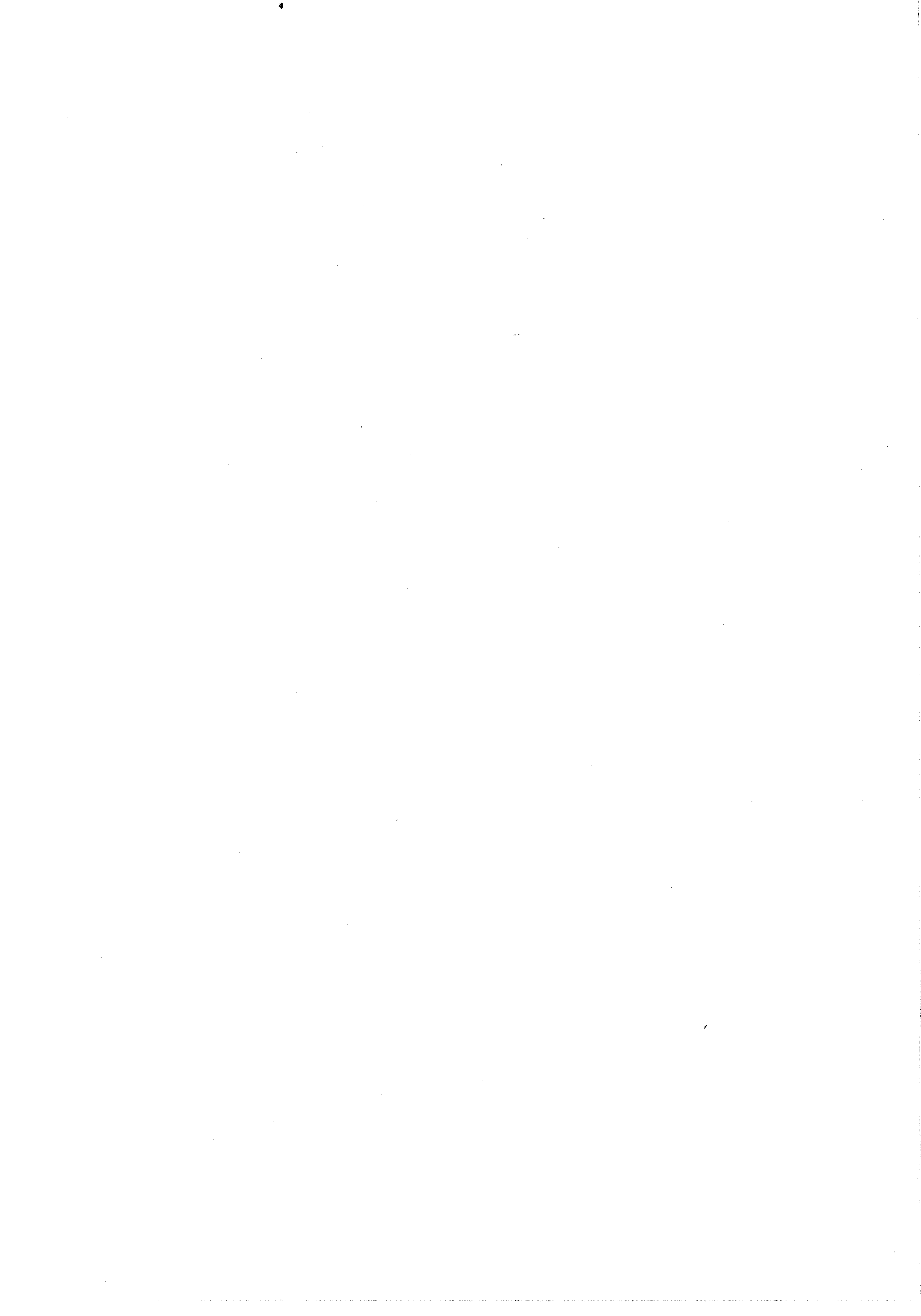
[17] J. REIF AND S. SEN, An efficient output-sensitive hidden surface removal algorithm and its parallelization, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 193-200.

[18] H. SCHIPPER AND M.H. OVERMARS, Applications of dynamic partition trees, Tech. Rept. RUU-CS-89-25, Dept. of Comp. Science, Utrecht University, 1989.

[19] A. SCHMITT, Time and space bounds for hidden line and hidden surface algorithms, *Eurographics '81*, pp. 43-56.

[20] M. SHARIR AND M.H. OVERMARS, A simple output-sensitive algorithm for hidden surface removal, Tech. Rept.RUU-CS-89-26, Dept. of Comp. Science, Utrecht University, 1989.

[21] I.E. SUTHERLAND, R.F. SPROULL AND R.A. SCHUMACKER, A characterization of ten hidden-surface algorithms, *Computing Surveys* **6** (1974), 1-25.



An improved technique for output-sensitive hidden surface removal

Micha Sharir and Mark H. Overmars

RUU-CS-89-32
December 1989

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

An improved technique for output-sensitive hidden surface removal

Micha Sharir and Mark H. Overmars

Technical Report RUU-CS-89-32
December 1989

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

An Improved Technique for Output-Sensitive Hidden Surface Removal*

Micha Sharir[†] Mark H. Overmars[‡]

December 20, 1989

Abstract

We derive a new output-sensitive algorithm for hidden surface removal in a collection of n triangles, viewed from a point z such that they can be ordered in an acyclic fashion according to their nearness to z. If k is the combinatorial complexity of the output *visibility map*, then we obtain a sophisticated randomized algorithm that runs in time $O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta})$, for any $\delta > 0$. The method is based on a new technique for tracing the visible contours using ray shooting.

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 non-intersecting polyhedral objects in 3-space, and a viewing point v, and our goal is to construct the visible parts of the given scene, as seen from v.

*Work by the first author was partially supported by the ESPRIT II Basic Research Actions Program of the EC, under contract No. 3075 (project ALCOM). Work by the second author 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 NCRD - the Israeli National Council for Research and Development, and the Fund for Basic Research in Electronics, Computers and Communication administered by the Israeli Academy of Sciences. A preliminary version of this paper appeared as part of the conference proceedings paper [15].

†School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, U.S.A.

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

Many solutions have been developed to date. The classical solutions from computer 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 [21]. In computational geometry a lot of work has been done in obtaining "object-space" solutions that 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 for e.g. 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. Most object-space hidden surface removal algorithms use these observations to compute the visible parts. They 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 $O(n^2)$ [5, 12]. More careful implementations run in time $O((n + I)\log n)$, where I denotes the number of intersections between the projected edges [8]. See also [11, 14, 19].

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 grid-like 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 [3, 9, 16]. Another output-sensitive algorithm has recently been proposed by Reif and Sen [17], for the special case of a polyhedral terrain.

Only recently some output-sensitive solutions have been proposed for the general problem. 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 [18] creates the view of the scene by adding the objects one

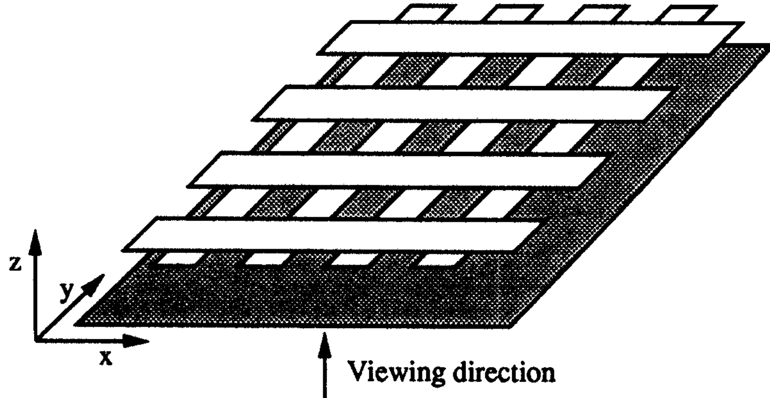


Figure 1: A scene with a small output size

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. The best worst-case output-sensitive result up to now was presented in a previous paper [20] and runs in time $O(n\sqrt{k}\log n)$, where k is the output size. This solution is quite simple. The only assumption that is made is that the objects can be ordered by "nearness" to the viewing point.

In this paper we continue our study and derive a more sophisticated and more efficient output-sensitive algorithm for the case our objects are triangles. Again we assume a depth ordering exists. In fact, we will only study the case of a set of horizontal triangles and a viewing point at $z = -\infty$. When a depth ordering exists the set of objects can always be transformed to this situation. We first present an initial coarse version of the algorithm. This version has no merits in itself — it is more complicated than the algorithm given in [20] and is actually inferior when the output size k is large. It uses a battery of sophisticated techniques, some of which use randomization,[1] recently developed in [10, 1, 2] for ray shooting and point location amidst a collection of intersecting segments in the plane. The randomized expected time complexity of the algorithm is $O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$, for any $\delta > 0$. After presenting geometric preliminaries in Section 2, we describe this algorithm in Section

[1]The randomization used by these algorithms does not depend on any probability distribution of the input triangles; the expected time bounds derived in this paper are over the internal randomization steps and hold for any given input.

3.

We then obtain in Section 4 an improved algorithm by partitioning the xy-plane into a large number of regions, and by solving the hidden surface removal problem, using the coarse algorithm, over each region separately. The partitioning depends on the structure of the arrangement of the projected triangle edges, and is chosen so as to obtain subproblems of small size. Using the deterministic partitioning algorithm of [1], we obtain a randomized algorithm whose expected running time is $O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta})$, for any $\delta > 0$.

Note that this bound is better than the bound on the running time of the algorithm in [20], as long as k is not too close to quadratic or too small. The improved algorithm is output-sensitive (unless k is smaller than roughly $n^{8/9}$). Unfortunately, the dependence on k is still very high. Hence, improvement might still be possible. We make a few comments on this issue in the concluding Section 5, where we also discuss open problems and suggest further possible attacks on this problem.

2 Geometric Preliminaries

Let $\mathbf{\Delta} = \{\Delta_1, \ldots, \Delta_n\}$ be a collection of n horizontal triangles in 3-D space, so that Δ_i lies in the plane $z = i$. The hidden surface removal problem for $\mathbf{\Delta}$ (and for a viewing point at $z = -\infty$) can be formulated as the problem of calculating the *lower envelope* of $\mathbf{\Delta}$, or, in other words, of partitioning the xy-plane into maximal regions so that for each region R there exists a unique triangle Δ (or no triangle at all) such that for all points $(x, y) \in R$, the lowest triangle lying above (x, y) is Δ (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 $\mathbf{\Delta}$.

Let us introduce a few notations. For a point $w \in R^3$ we denote its orthogonal xy-projection by $\pi(w)$. We also define, for a point $w \in R^3$, the *upward lifting* $\tau(w)$ of w to be the unique point w' such that w' lies strictly above w on some triangle of $\mathbf{\Delta}$, and the open vertical segment ww' meets no other triangle; if no triangle lies above w then $\tau(w)$ is undefined. We say that $\tau(w)$ is *visible from* w. We also say that a point w on some triangle is *visible* if it is visible from $\pi(w)$. Thus, thinking of the triangles in $\mathbf{\Delta}$ as being opaque, a point z on a triangle is visible from another point w if we can see z when standing at w and looking vertically upwards, and z is visible if we can see it when standing vertically below it on the xy-plane and looking directly upwards. We also say that triangle Δ_i *hides* a point z from another point w if w lies vertically below z and the segment wz intersects Δ_i; we say that Δ_i just hides z if it hides it from $\pi(z)$.

The visibility map M can be regarded as the xy projection of all visible points, where each projected point is labeled with its triangle. Plainly, each face of M is either the projection of a maximal connected visible portion of some triangle in $\mathbf{\Delta}$

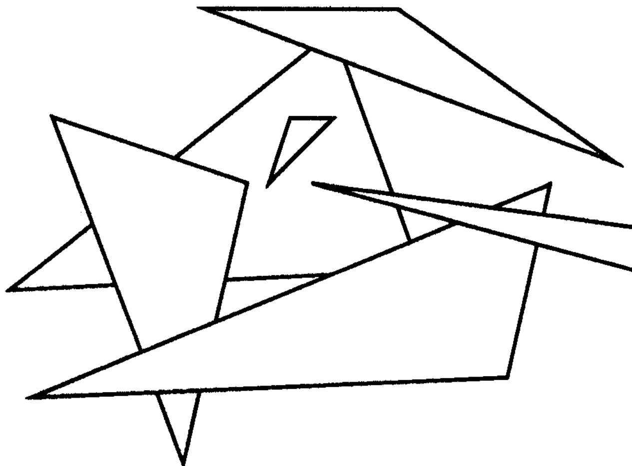


Figure 2: The visibility map

or a connected component of the complement of the union of the projections of all these triangles. The edges of M are visible connected portions of the triangle edges. Each vertex q of M is either $\pi(w)$ for some visible vertex w of one of the triangles in Δ or the intersection of the projections of two triangle edges e, e' such that the vertical line passing through q meets these edges at the respective points w, w', with, say, w' lying above w, so that the segment qw' meets no other triangle (in other words, $\tau(q) = w$ and $\tau(w) = w'$). Let us assume that the triangles in Δ lie in *general position*, meaning that no vertex of one triangle is co-vertical with an edge of another triangle, and that no three triangle edges have concurrent xy-projections. Note that in this case each vertex of M has degree 2 (if it is a projected triangle vertex) or 3 (in the latter case); the degree-3 vertices of M look like "T" junctions. See figure 2 for an illustration. Let k denote the number of vertices of M (the "output size"). Plainly $k = O(n^2)$, but it can be as small as 3 (when the lowest triangle hides all the others).

3 An Initial Coarse Output-Sensitive Algorithm

In this section we present a coarse output-sensitive technique for hidden surface removal. The algorithm uses randomization, and its expected time complexity is $O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$, for any $\delta > 0$; this makes the algorithm considerably inferior to the simpler algorithm reported in [20] when k is large. Nevertheless, we will later combine this algorithm with a partitioning scheme, that breaks the problem into subproblems of smaller size, and apply the algorithm of this section to

each subproblem separately. This will produce an improved solution, as detailed in the following section.

The best way to regard the algorithm presented in this section is as a high-level general technique, whose efficient implementation requires the availability of certain primitives, mainly for *ray shooting* and *point location* in a planar arrangement of overlapping triangles. The current version of the paper exploits the best known solutions for these problems. However, a lot of work is currently underway to improve these techniques, and we expect that such improvements will automatically lead to improved performance of the algorithm presented below.

3.1 An overview of the algorithm

The basis for the algorithm is the following simple yet crucial observation.

Lemma 3.1 *Let E denote the union of all edges of the visibility map M. Then each connected component of E contains a projected triangle vertex.*

Proof: Simply take the leftmost point p in the connected component (when there is more than one leftmost point, take the bottommost). Clearly, p is a vertex of M. There are only three different types of vertices in M (the points z in figure 3). Clearly, only when p is a vertex of a triangle, there is no other point to the left of p. □

Lemma 3.1 suggests the following high-level approach for obtaining an output-sensitive hidden surface removal algorithm.

1. Find all visible vertices of the triangles in $\mathbf{\Delta}$.
2. For each visible vertex v, construct the connected component E_v of E containing $\pi(v)$, by repeated "ray-shooting" along the edges of E_v.

Let us define the ray-shooting operation in more detail. At each ray-shooting step we are given an edge e of some triangle Δ, a point w on e, and a direction along e, with the property that if we follow e from w in this direction, we traverse a visible portion of e. The goal of the ray-shooting query is to follow this visible portion of e until we reach the first of one of the following types of points (see figure 3 for an illustration):

(i) a vertex z of e;

(ii) a point z on e at which e is hidden by some lower triangle;

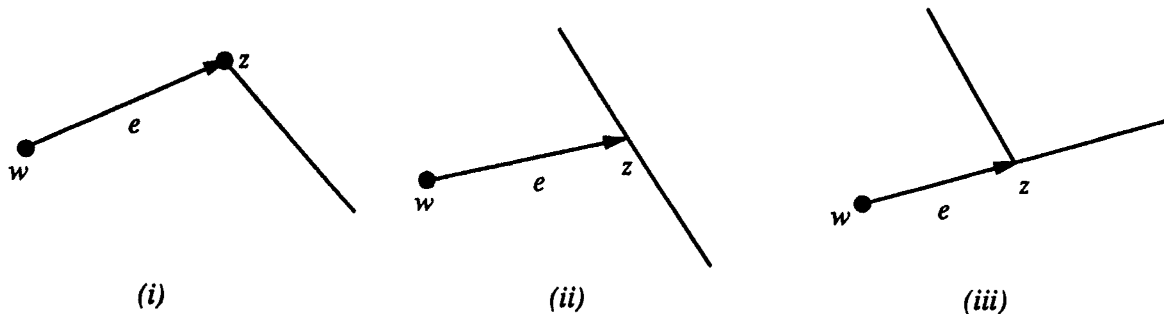


Figure 3: Ray shooting for hidden surface removal

(iii) a point z on e at which e hides another triangle edge e' lying above z (so that, in the vicinity of $\pi(z)$, an appropriate portion of e' is visible).

In each of these cases we find a new vertex $\pi(z)$ of M. Moreover, in cases (ii) and (iii) we also require from the ray shooting procedure to report the other edge e' that hides e or is hidden by e at z, and to determine which direction(s) along e' make it visible near $\pi(z)$. Thus, having found z, e' and these directions, we can then repeat the ray shooting procedure along e' in the given direction(s) from the vertical projection of z onto e'. Initiating these shootings at each visible triangle vertex along the two edges incident to it, and continuing it until no new visible edges are obtained, we discover in this way the entire visibility map M, as follows trivially from Lemma 3.1. (Some care has to be taken that parts are not reported more than once. This can easily be done by marking vertices and parts of edges visited and only traversing along vertices and edges that are not marked.)

We will show that finding all visible triangle vertices can be accomplished in time $O(n^{4/3}\log^{2.89} n)$, and that the triangles of Δ can be preprocessed in randomized expected time $O(n \log n)$ into a data structure of linear size, so that each ray shooting query can be executed in time $O(n^{2/3+\delta})$, for any $\delta > 0$. Putting all of these bounds together, we obtain the main result of this section.

Theorem 3.2 *One can perform hidden surface removal for n horizontal triangles in 3-space in randomized expected time*

$$O(n^{4/3}\log^{2.89} n + kn^{2/3+\delta})$$

for any $\delta > 0$, where k is the output size of the problem.

Details of efficient implementation of the various steps of the algorithm are given in the following two subsections. In subsection 3.2 we show how to find efficiently all the visible vertices of the given triangles; as a matter of fact, we give a more general procedure for finding the triangle lying immediately above any query point in 3-space. Subsection 3.3 presents an efficient solution of the ray shooting problem.

3.2 Vertical visibility queries

In this subsection we solve the following problems

(i) Given a collection $\mathbf{\Delta}$ of horizontal triangles in 3-space as above, preprocess it so that, given any point w in 3-space, we can determine efficiently the triangle lying immediately above w (i.e., the triangle containing $\tau(w)$), or determine that no such triangle exists.

(ii) Given a collection $\mathbf{\Delta}$ as above, and a collection S of m points on the xy-plane, determine for each point the triangle lying immediately above it or else report that no such triangle exists.

Clearly, solving the second problem would immediately yield a solution to the problem of finding all visible vertices — apply it to the set of all projected triangle vertices $\pi(v)$; a vertex v of a triangle Δ is visible if and only if the triangle lying immediately above v is Δ itself. The first variant of the problem will be required in our solution of the ray-shooting problem, to be described in the following subsection.

Fortunately for us, both problems are easy to solve using as a main component the following "implicit point location" technique, which has been developed in [10], [1]. Although the technique in [1] has a faster query time, we will use for the first problem the alternative technique of [10], because it requires less preprocessing time and storage, and this will turn out to be an important factor in the analysis of the algorithm in Section 4 below.

Specifically, it is shown in [10] that, given n triangles in the plane, we can preprocess them in randomized expected time $O(n \log n)$ into a data structure of linear size, so that, given any query point a, we can determine in $O(n^{2/3+\delta})$ time whether a lies in the union of the given triangles; here δ is an arbitrarily small positive constant, and the constants of proportionality in the above bounds depend on δ.

To solve our original problem (i), we next construct a perfectly balanced tree $\mathcal{T}$ storing the triangles of $\mathbf{\Delta}$ in its leaves, sorted in increasing height. For each node ξ of $\mathcal{T}$, let $\mathcal{T}_\xi$ denote the subtree of $\mathcal{T}$ rooted at ξ, and let $\mathbf{\Delta}_\xi$ denote the corresponding subcollection of triangles stored at the leaves of $\mathcal{T}_\xi$. For each node ξ, preprocess the xy-projections of the triangles in $\mathbf{\Delta}_\xi$ for implicit point location queries.

Now, given a query point w, we find the subset $\{\Delta_j, \ldots, \Delta_n\}$ of triangles in $\mathbf{\Delta}$ whose height is greater than the height of w, and represent this subset as the union of subsets $\mathbf{\Delta}_\xi$ for $O(\log n)$ nodes ξ of $\mathcal{T}$. (These are the nodes bordering the search path in $\mathcal{T}$ towards Δ_j to the right.) We now perform a binary search over this sequence of triangles, as follows. First we go over the subcollections $\mathbf{\Delta}_\xi$, in order of increasing height of their triangles, until the first ξ for which $\pi(w)$ is in the union of the projections of the triangles in $\mathbf{\Delta}_\xi$. Then we search through the tree $\mathcal{T}_\xi$. At each node η along the search path we test whether $\pi(w)$ lies in the union of the projections of the triangles in $\mathbf{\Delta}_{\eta'}$, where η' is the left child of η. If so, we continue the search at η'; otherwise, we continue the search at the right child of η. In this manner, after $O(\log n)$ queries, we obtain the triangle lying immediately above w, or determine that no such triangle exists. The overall cost of a query is easily seen to be $O(n^{2/3+\delta})$. The overall expected preprocessing time is $O(n \log^2 n)$ and the space is $O(n \log n)$. However, as observed in [10] (see also [6]), we can decrease the preprocessing time to $O(n \log n)$ and the space to linear, without affecting the asymptotic bound on query time, by building the data structures for implicit point location only once every t levels of $\mathcal{T}$, for an appropriate parameter t.

As to the second problem (ii), we apply Agarwal's technique [1] as follows. We construct the tree $\mathcal{T}$ as above, and the run a "simultaneous binary search" through $\mathcal{T}$ with all the given points. Specifically, we project on the xy plane all triangles in $\mathbf{\Delta}_{\xi_L}$, where ξ_L is the left child of the root of $\mathcal{T}$—that is, the lower half of the triangles of $\mathbf{\Delta}$. We then test which of the points of S lie in the union of these projected triangles, using the batched implicit point location algorithm of [1]; this requires $O(n^{4/3} \log^{2.89} n)$ time. Let S_1 be the resulting subset of S and let S_2 be the remainder of S. Note that for any point in S_1 the triangle lying immediately above it must belong to $\mathbf{\Delta}_{\xi_L}$ whereas for a point in S_2 this triangle (if it exists) belongs to the higher half $\mathbf{\Delta}_{\xi_R}$, where ξ_R is the right child of the root of $\mathcal{T}$. We thus recurse twice, once with $\mathbf{\Delta}_{\xi_L}$ and S_1, and once with $\mathbf{\Delta}_{\xi_R}$ and S_2. It is shown in [1] that the overall time of this procedure is still $O(n^{4/3} \log^{2.89} n)$.

3.3 The ray shooting procedure

Let us first consider the following auxiliary problem: Given n triangles $T_1, \ldots, T_n$ in the xy plane, preprocess them so that, given any query ray ρ, we can find efficiently the first intersection of ρ with any triangle edge, or determine that no such intersection exists.

This problem has recently been studied in [10, 2]. Again, although the best query time known to date is due to Agarwal [2], we will use the alternative algorithm of [10], which requires $O(n \log n)$ randomized expected preprocessing time, linear storage, and $O(n^{2/3+\delta})$ query time, for any $\delta > 0$.

Let us now return to our original 3-D ray shooting problem. We are given an

edge e of some triangle Δ_i, a point w on e, and a direction along e from w. Our goal is twofold: to find the first edge that hides e (as we follow e from w in the specified direction), and to find the first visible edge that lies above e and is hidden by e. The answer to the query is the nearest of these two events, or the appropriate endpoint of e if it is reached before any of these events takes place. The solutions to these two subproblems are very similar, and we will consider them separately.

To facilitate both solutions, we first construct the binary tree $\mathcal{T}$ as in the previous subsection. For each node ξ of $\mathcal{T}$, we preprocess for planar ray shooting the xy projections of the triangles in $\mathbf{\Delta}_\xi$, as defined above. As argued at the end of the preceding subsection, this can be done in overall $O(n \log n)$ randomized expected preprocessing time and linear space (over all nodes ξ of $\mathcal{T}$), without affecting the asymptotic bound on query time.

Consider now the first subproblem, of finding the first edge that hides e. Take the collection of triangles that are lower than Δ_i, i.e. $\{\Delta_1, \ldots, \Delta_{i-1}\}$, and represent it as the union of subcollections $\mathbf{\Delta}_\xi$, for $O(\log n)$ nodes ξ of $\mathcal{T}$. For each such node ξ, shoot along the projection $\pi(e)$ of e from $\pi(w)$ in the specified direction, to obtain the first intersection of $\pi(e)$ with a projected edge of a triangle in $\mathbf{\Delta}_\xi$. Let the nearest of all these intersections be q. If q lies further away from $\pi(w)$ than the relevant endpoint s of $\pi(e)$, then no edge hides this portion of e. Otherwise it is easily verified that $\tau(q)$ lies on the first visible edge that hides this portion of e. The total cost of all these ray shootings is easily seen to be $O(n^{2/3+\delta})$.

The second subproblem, of finding the first visible edge that e hides, is only slightly more complicated. We first find the triangle Δ_j lying immediately above w, using the technique described in the previous subsection. We then apply the ray-shooting technique described in the previous paragraph to the collection of triangles $\{\Delta_{i+1}, \ldots, \Delta_j\}$ (or $\{\Delta_{i+1}, \ldots, \Delta_n\}$ if no triangle lies above w). Let q be the nearest point returned by the above procedure. If the endpoint s of the traversed portion of e lies nearer to w than q then e hides no other visible edge, otherwise q is the first point (after w) where the traversed portion of e hides an edge. Note that in the last case, the edge that e hides at q is not necessarily visible, because e could already be invisible at this point. The complete ray shooting procedure proceeds by performing the two types of searches – what is the first edge that e hides and what is the first edge that hides e. The nearest of these events to w (if it occurs before the endpoint of e) is the point we seek, and the procedure has enough information to report also which of the two types of hiding occurs at that point, which edge hides e or is hidden by e at this point, and which direction(s) along that edge make it visible in the vicinity of the corresponding vertex of M. The overall cost of a ray-shooting query is thus $O(n^{2/3+\delta})$, for any $\delta > 0$.

4 An Improved Partition-Based Algorithm

In this section we improve the algorithm presented in the previous section using the following fairly simple idea. We partition the xy-plane into triangular regions, called *base cells*, with the properties that

(i) the number of regions is small, and

(ii) each region is crossed by only a small number of projected triangle edges (so most of the triangle projections are either disjoint from, or fully contain, that region).

This partitioning allows us to break the original hidden surface removal problem into a small number of subproblems, one per each base cell, and each subproblem involves only a small number of triangles. We then apply the algorithm given in the preceding section to each subproblem separately, and combine the outputs to obtain the overall visibility map. There are two important characteristics of this approach. First, the total output size is (almost) the sum of the output sizes of the subproblems. Second, because the size of each subproblem is small, the overhead per vertex of the visibility map becomes smaller. (This partitioning trick has also been proposed as a heuristic for image-space techniques, but without explicit quantification of the improvement in complexity that it entails.)

Nevertheless, the partitioning raises several new technical difficulties. Each of these needs to be solved efficiently, or else the entire scheme would collapse. We describe below in detail each of the new procedures that are required in order to overcome these difficulties.

Partitioning

Let $r \leq n$ be some integer parameter, whose value will be chosen later. Let G be the collection of the xy-projections of the $3n$ triangle edges. Using the partitioning algorithm of Agarwal [1], we can partition the xy plane into $m = O(r^2)$ base cells $T_1, \ldots, T_m$, such that each T_ℓ meets at most $O(n/r)$ projected edges; the time complexity of the partitioning is $O(nr \log n \cdot \log^\beta r)$, for some constant $\beta < 3.33$, i.e. $O(nr \log^{4.33} n)$.

This partitioning allows us to break the original hidden surface removal problem into $O(r^2)$ subproblems, one per each base cell. For each such T_ℓ there are only $O(n/r)$ triangles Δ_i whose projected boundaries intersect T_ℓ; the projection of every other triangle in $\mathbf{\Delta}$ either completely contains T_ℓ or is disjoint from T_ℓ. Moreover, let $\Delta_{i(\ell)}$ be the lowest triangle whose xy-projection fully contains T_ℓ; then to solve the hidden surface removal problem over T_ℓ it suffices to process only the triangles lower than $\Delta_{i(\ell)}$ whose projected boundaries intersect T_ℓ. We next describe an efficient procedure for calculating $\Delta_{i(\ell)}$ for all cells T_ℓ, in overall time $O(nr \log r)$.

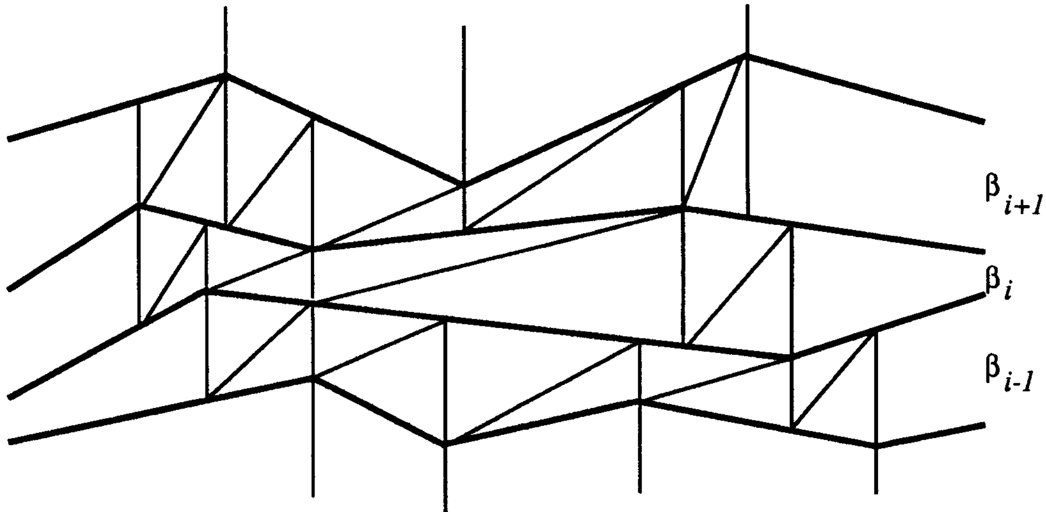


Figure 4: The base cells in a partitioning

Finding lowest "covering triangles"

Our goal is to find, for each base cell T_ℓ, the lowest triangle $\Delta_{i(\ell)}$ that fully contains T_ℓ. We note that the partition produced in [1] can be represented as a collection of r "belts", $\beta_1, \ldots, \beta_r$, separated from each other by $r - 1$ x-monotone polygonal chains, so that within each belt the base cells are linearly ordered from left to right, and are separated from each other by vertical segments drawn from the vertices of the two chains bounding the belt, and by diagonals of the resulting trapezoidal cells. See figure 4 for an illustration.

We represent each belt β_i by a balanced binary tree Q_i storing in its leaves the base cells of the belt in their left-to-right order.

Consider the effect of a single triangle $\Delta \in \mathbf{\Delta}$ on Q_i. Suppose the edges of Δ intersect $j = j_{i,\Delta}$ base cells in β_i. Removing these cells from β_i partitions Q_i into at most $j + 1$ contiguous portions such that the cells in each portion are either all disjoint from Δ or all contained in Δ. If β_i contains q_i base cells, then we can easily represent the portions of β_i that contain cells fully covered by Δ as a disjoint union of $O(j_{i,\Delta} \log q_i)$ subtrees, and, in time $O(\log q_i + j_{i,\Delta} \log q_i)$, we can store Δ at the roots of these subtrees. Repeating this step for each belt β_i and for each triangle in $\mathbf{\Delta}$, and observing that the sum of all the corresponding quantities $j_{i,\Delta}$ is $O(nr)$, we can "encode" all triangles of $\mathbf{\Delta}$ in this manner, in total time $O(nr \log r)$. In this encoding, we follow the rule that whenever more than one triangle Δ is stored at some node of the tree representing β_i, only the lowest of these triangles is maintained there. Now, for each T_ℓ, we find the tree Q_i and its leaf ξ where T_ℓ is stored, and

walk along the search path in Q_i to ξ. The lowest triangle Δ encountered along that path is the desired $\Delta_{i(\ell)}$. Clearly the overall cost of this procedure is $O(nr \log r)$.

Clipping the visibility map over base cell edges

We next wish to apply, for each T_ℓ, the algorithm described in the preceding section to the corresponding collection of $O(n/r)$ triangles of $\mathbf{\Delta}$ "partially overlapping" T_ℓ and lying below $\Delta_{i(\ell)}$. This however requires some extra care. To begin with, each such triangle has to be clipped to its portion lying above T_ℓ. If we denote by M_ℓ the portion of the visibility map M over T_ℓ, and let k_ℓ denote the number of vertices of M_ℓ within the interior of T_ℓ, then plainly $\sum_\ell k_\ell \le k$. However, M_ℓ may have additional vertices and edges over the edges of T_ℓ, caused by the clipping of the relevant triangles. To find these features is easy – simply observe that the lower envelope of the triangles of $\mathbf{\Delta}$ over an edge of T_ℓ is just an envelope of $O(n/r)$ horizontal line segments, which has plainly only $O(n/r)$ features, and is easily computed in time $O((n/r)\log(n/r))$. However, the presence of these features as part of M_ℓ raises the following subtle problem.

Eliminating redundant ray-shootings

Lemma 3.1 can be easily replaced by a similar claim, asserting that it suffices to start the ray shooting process from all visible triangle vertices and from all visible vertices of M_ℓ lying above the edges of T_ℓ. However, it is imperative that when we shoot from a vertex of the latter type, we do it efficiently, because, as will be apparent from subsequent analysis, we can afford to pay $O((n/r)^{2/3+\delta})$ time for such a shooting only if it will produce a real vertex of M (i.e. not a "clipped" vertex lying over an edge of T_ℓ); see figure 5. In other words, we need a more efficient preliminary procedure to find which ray shootings from the edges of T_ℓ are worth performing. Using tricks similar to those in [2], we can obtain such a "sifting" procedure, having overall cost $O(n^{4/3}\log^{2.89} n + nr\alpha(n)\log^3 n)$.

Specifically, we fix a base cell T_ℓ and partition the set of triangle edges that meet T_ℓ into two subsets, the set $\mathcal{G}$ of "long" edges that meet the boundary of T_ℓ and the set $\mathcal{S}$ of "short" edges that are fully contained in the interior of T_ℓ.

Consider first a simpler problem: Given sets $\mathcal{G}$, $\mathcal{S}$ as above, where all segments in $\mathcal{G}$ are assumed to be clipped to within T_ℓ, and another collection $\mathcal{L}$ of lines that intersect T_ℓ, find which lines of $\mathcal{L}$ intersect at least one segment in $\mathcal{G} \cup \mathcal{S}$. Put $g = |\mathcal{G}|$, $s = |\mathcal{S}|$, and $m = |\mathcal{L}|$.

We will separately test for intersection with $\mathcal{G}$ and with $\mathcal{S}$. For intersection with $\mathcal{G}$, we first construct the unbounded face of the arrangement of the clipped segments of $\mathcal{G}$, and partition it into subfaces by adding the edges of T_ℓ (this in effect produces the *zone* of the boundary of T_ℓ in this arrangement; see figure 6). We next process each resulting subface for logarithmic-time ray shooting, as in [4]; note that each such subface is simply connected, so this is indeed possible. Finally, we shoot along each line in $\mathcal{L}$ from its intersections with ∂T_ℓ into the corresponding subface. If the

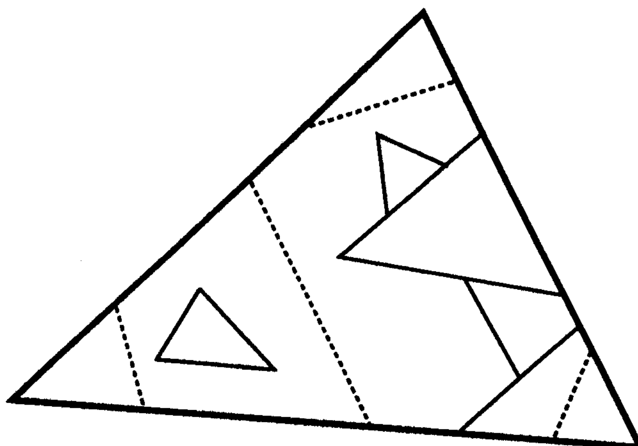


Figure 5: The dotted edges are spurious ray shootings

shooting hits a segment of $\mathcal{G}$ then an intersection has been detected. Otherwise it will hit another edge of T_ℓ, in which case it is clear that no intersection with $\mathcal{G}$ exists.

The time required for this step is

$$O(g\alpha(g)\log^2 g + m\log g)$$

where the first term bounds the time needed to construct the unbounded face of the arrangement of long edges [7], which also dominates the time needed to preprocess the resulting zone for fast ray shooting, and the second term bounds the cost of the ray shootings.

To detect intersection with $\mathcal{S}$, we pass to the dual plane, so $\mathcal{L}$ is mapped to a set $\mathcal{L}^*$ of points, and each segment in $\mathcal{S}$ is mapped into a double wedge. A line l of $\mathcal{L}$ meets a segment e of $\mathcal{S}$ if and only if the dual point l^* lies in the double wedge e^* dual to e. The problem thus becomes a special case of the "implicit point location" problem discussed in [10], [1]. That is, given m points and s double wedges, determine which points lie in the union of the double wedges. The best algorithm for solving this problem is due to Agarwal [1], and it runs in time $O(m^{2/3}s^{2/3}\log^{2.89} s + (s+m)\log s)$.

Let us now return to the actual problem of detecting redundant ray-shootings from the boundary of T_ℓ. Let p be a point lying on an edge e of some triangle $\Delta_i \in \mathbf{\Delta}$ and visible from an edge of T_ℓ. The processing of the visibility map along the edges of T_ℓ can also determine easily the triangle Δ_j of $\mathbf{\Delta}$ lying immediately above p. Hence the ray-shooting from p along e into T_ℓ will be non-redundant if and only if the projection of e meets, within T_ℓ, another projected edge of some triangle Δ_h, for $h \le j$ or e ends in T_ℓ. The second case is easily tested in constant time. For

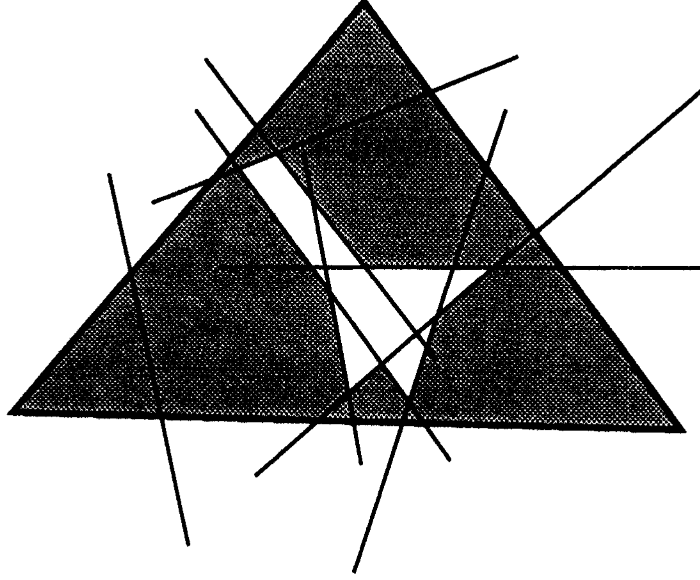


Figure 6: The zone of T_ℓ in the arrangement of the long edges

the first case we proceed as follows.

Let $\mathcal{L}$ denote the collection of lines containing the projections of edges e as above, and associate with each line $l \in \mathcal{L}$ an index $\phi(l) = j$, where j is the index of the triangle lying immediately above p, as defined above. Partition the set of triangles that partially overlap T_ℓ (and lie below $\Delta_{i(\ell)}$) into two roughly equal subsets, $\boldsymbol{\Delta}_1^{(\ell)}$, $\boldsymbol{\Delta}_2^{(\ell)}$, such that the triangles in $\boldsymbol{\Delta}_1^{(\ell)}$ lie lower than the triangles in $\boldsymbol{\Delta}_2^{(\ell)}$. Similarly partition $\mathcal{L}$ into two subsets, $\mathcal{L}_1$, $\mathcal{L}_2$, such that the index $\phi(l)$ of a line $l \in \mathcal{L}_1$ (resp. $l \in \mathcal{L}_2$) corresponds to a triangle that lies in $\boldsymbol{\Delta}_1^{(\ell)}$ (resp. in $\boldsymbol{\Delta}_2^{(\ell)}$). We then test for intersection between $\mathcal{L}_2$ and the projections of the triangles in $\boldsymbol{\Delta}_1^{(\ell)}$, using the procedure outlined above, and then recursively test for non-redundancy for shooting along lines of $\mathcal{L}_1$ amidst the triangles of $\boldsymbol{\Delta}_1^{(\ell)}$, and similarly for shooting along lines of $\mathcal{L}_2$ amidst the triangles of $\boldsymbol{\Delta}_2^{(\ell)}$. It is easily checked that the shooting along a line $l \in \mathcal{L}$ is non-redundant if and only if an intersection is detected along l at some recursive level of this procedure.

Since in our case $|\mathcal{G}|, |\mathcal{L}| = O(n/r)$ and the recursion has depth only $O(\log(n/r))$, it is easily verified that the cost of this procedure is

$$O\left(\frac{n}{r}\alpha(n)\log^3 n + \left(\frac{n}{r}\right)^{2/3} {s_\ell}^{2/3}\log^{2.89} n\right)$$

where $s_\ell = |S|$. Summing this over all base cells T_ℓ, using Hölder's inequality, and noting that $\sum_\ell s_\ell = O(n)$, we obtain an overall cost of

$$O(nr\alpha(n)\log^3 n + n^{4/3}\log^{2.89} n) \ .$$

Calculating visible vertices

Let us now return to the overall algorithm. Another step required there is to find all visible triangle vertices. An analysis similar to the one carried out above shows that the overall cost of this step is $O(n^{4/3}\log^{2.89} n)$, if performed separately within each base cell. Alternatively, we could compute all visible vertices *before* the partitioning, and then distribute them among the base cells. This requires the same amount of time asymptotically.

Putting it all together

Having all this machinery at hand, we can now apply the algorithm of Section 3 to each base cell separately. The (net) randomized expected time needed to solve the ℓ-th subproblem, exclusive of the various preparatory steps described above, is thus $O(k_\ell(n/r)^{2/3+\delta})$. We now sum this over all $O(r^2)$ subproblems, and add the cost of all the other steps, namely the partitioning, computing visible vertices, computing the "covering triangles" $\Delta_{i(\ell)}$, computing M over edges of the cells, and sifting out redundant ray shootings. We thus obtain the overall bound

$$O\left(n^{4/3}\log^{2.89} n + nr\log^{4.33} n + (\sum_\ell k_\ell)\cdot\left(\frac{n}{r}\right)^{2/3+\delta}\right) \ .$$

But $\sum_\ell k_\ell = k$. Now choose

$$r = \frac{k^{\frac{3}{5(1+3\delta/2)}}}{n^{1/5}} \ .$$

It now can easily be checked that this yields the bound

$$O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta'})$$

for a different, but still arbitrarily small, $\delta' > 0$.

Guessing the value of k

One last tricky issue remains. Since we do not know k in advance, how can we choose r to be a function of k? This is handled by guessing the value of k, as follows. Start with $k = n^{8/9}$ and run the algorithm as described above. If the time of the algorithm begins to exceed the bound given above then we know that the actual value of k is larger. In this case we double the guessed value of k and rerun the algorithm. This is continued until the time alloted for the execution of the algorithm is not exceeded, in which case the algorithm will complete the calculation of M as required. (A final note of caution: The preprocessing for ray shooting, using the technique of [10], is randomized. However, we can tune it so that it verifies that

the random sample drawn is one that guarantees the desired query time; if the sample is bad, it is discarded and another sample is chosen. In this manner, when we execute the various hidden surface removal procedures described above, we know what time complexity to expect, if our choice of r is the correct one.) The overall time complexity of the algorithm is easily seen to be asymptotically the same as the complexity of the last step, namely

$$O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta'})$$

for any $\delta' > 0$.

Theorem 4.1 *One can perform hidden surface removal for n horizontal triangles in randomized expected time*

$$O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta'})$$

for any $\delta' > 0$.

A comment worth making is that the algorithm is not very efficient when k is really small, that is much smaller than n, because the overhead of preparing the data structures becomes too large. We can handle this problem by first executing the simple algorithm of [20], and stop it as soon as it has detected more than $n^{2/3}$ output vertices. If k is smaller than this, the first algorithm will run to completion in time $O(n\sqrt{k}\log n) = O(n^{4/3}\log n)$. Otherwise, we run the algorithm presented above. In this way the total amount of time used becomes

$$\min\{O(n\sqrt{k}\log n), O(n^{4/3}\log^{2.89} n + k^{3/5}n^{4/5+\delta'})\}$$

Remark: When k is between roughly $n^{2/3}$ and $n^{8/9}$, the algorithm (as just modified) is a bit insensitive to k, because it runs in time $O(n^{4/3}\log^{2.89} n)$, which does not take advantage of k being smaller than $n^{8/9}$. Can this gap be "closed"?

5 Conclusion

We have presented a new output-sensitive hidden surface removal algorithm that is faster than previous solutions. The method is based on a large number of different techniques including a method of tracing the visibility map using ray shooting, partitioning techniques, implicit point location and a method for guessing the output size. The time complexity of the algorithm depends on the performance of the ray shooting and point location techniques. If these problems could be solved using close to linear (randomized expected) preprocessing time and close to $O(n^{1/2})$ query time, then the dependence of the above algorithm on k would come close to $O(k^{2/3}n^{2/3})$.

We conjecture that such complexity can be attained, and pose this as a challenging open problem.

The results obtained in this paper also raise several related open problems. A first problem deals with dynamically maintaining the visibility map under insertions or deletions of objects or when moving the point of view. Almost no results are known for this problem. A second questions concerns generalizations of the method. As noted the same technique can be used for arbitrary collections of triangles as long as no cyclic overlap occurs, using a simple transformation. The method can also easily be adapted to simple polygons of bounded degree (rather than triangles only). The method will be exactly the same. Unfortunately, the techniques for ray shooting and implicit point location no longer work when objects have curved boundaries. Finally, can we use the new ideas developed here to obtain a subquadratic solution to the following problem: Given n triangles in the plane, does their union completely cover some specified region, e.g. the unit square?

References

[1] P.K. AGARWAL, A deterministic algorithm for partitioning arrangements of lines and its applications, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 11-22.

[2] P.K. AGARWAL, Ray shooting and other applications of spanning trees with low stabbing number, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 315-325.

[3] M. BERN, Hidden surface removal for rectangles, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 183-192.

[4] B. CHAZELLE AND L. GUIBAS, Visibility and intersection problems in plane geometry, *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 135-146.

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

[6] D. DOBKIN AND H. EDELSBRUNNER, Space searching for intersecting objects, *J. Algorithms* **8** (1987), 348-361.

[7] H. EDELSBRUNNER, L. GUIBAS AND M. SHARIR, The complexity of many faces in arrangements of lines and of segments, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 44-55.

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

[9] 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, Tech. Rept. RUU-CS-89-24, Dept. of Comp. Science, Utrecht University, 1989.

[10] L. GUIBAS, M. OVERMARS AND M. SHARIR, Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques, *Proc. SWAT 88*, Lecture Notes in Comp. Science 318, Springer Verlag, 1988, pp. 64-73.

[11] R.H. GÜTING AND T. OTTMAN, New algorithms for special cases of the hidden line elimination problem, *Comp. Vision, Graphics and Image Processing* **40** (1987), 188-204.

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

[13] K. MULMULEY, An efficient algorithm for hidden surface removal, *Computer Graphics* **23** (1989), 379-388.

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

[15] M.H. OVERMARS AND M. SHARIR, Output-sensitive hidden surface removal, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 598–603.

[16] F.P. Preparata, J.S. Vitter and M. Yvinec, Computation of the axial view of a set of isothetic parallelepipeds, Techn. Rep. LIENS-88-1, Lab. d'Informatique de L'Ecole Normal Supérieure, 1988.

[17] 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.

[18] H. SCHIPPER AND M.H. OVERMARS, Applications of dynamic partition trees, Tech. Rept. RUU-CS-89-25, Dept. of Comp. Science, Utrecht University, 1989.

[19] A. SCHMITT, Time and space bounds for hidden line and hidden surface algorithms, *Eurographics '81*, pp. 43-56.

[20] M. SHARIR AND M.H. OVERMARS, A simple output-sensitive algorithm for hidden surface removal, Tech. Rept.RUU-CS-89-26, Dept. of Comp. Science, Utrecht University, 1989.

[21] I.E. SUTHERLAND, R.F. SPROULL AND R.A. SCHUMACKER, A characterization of ten hidden-surface algorithms, *Computing Surveys* **6** (1974), 1-25.