Efficient Ray Shooting and Hidden Surface Removal

M. de Berg, D. Halperin, M. Overmars, J. Snoeyink

M. van Kreveld

RUU-CS-91-28
July 1991

Utrecht University

Department of Computer Science

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

Efficient Ray Shooting and Hidden Surface Removal

M. de Berg, D. Halperin, M. Overmars, J. Snoeyink

M. van Kreveld

Technical Report RUU-CS-91-28
July 1991

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

ISSN: 0924-3275

Efficient Ray Shooting and Hidden Surface Removal*

M. de Berg[†] D. Halperin[‡] M. Overmars[†] J. Snoeyink[†§]
M. van Kreveld[†]

Abstract

In this paper we study the ray shooting problem for three special classes of polyhedral objects in space: axis-parallel polyhedra, curtains (unbounded polygons with three edges, two of which are parallel to the z-axis and extend downward to minus infinity) and fat horizontal triangles (triangles parallel to the xy-plane whose angles are greater than some fixed constant). For all three problems structures are presented using $O(n^{2+\epsilon})$ preprocessing, for any fixed $\epsilon > 0$, with $O(\log n)$ query time. We also study the general ray shooting problem in an arbitrary set of (possibly intersecting) triangles. Here we present a structure that uses $O(n^{4+\epsilon})$ preprocessing and has a query time of $O(\log n)$.

As an application of the ray shooting structure for curtains we show that the view of a set of (non-intersecting) polyhedra with n edges in total can be computed in $O(n^{1+\epsilon}\sqrt{k})$ time, where k is the size of the output, for any fixed $\epsilon > 0$. This is the first output-sensitive algorithm for this problem that does not need a depth-order on the faces of the polyhedra.

1 Introduction

The ray shooting problem is to preprocess a set of objects such that the first object hit by a query ray can be determined efficiently. This problem (also called the ray tracing problem) is an important problem in computer graphics. To compute the shading information that is necessary to render a realistic picture of a scene, one can trace rays from the view point until they hit an object, trace the deflected rays,

*This research was supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM). The first and third author were also supported by the Dutch Organization for Scientific Research (N.W.O.).

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

[‡]Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

[§]On leave from the Department of Computer Science of the University of British Columbia.

etcetera, to see if a light source can be reached. Since this ray tracing operation has to be performed many times, it is natural to preprocess the objects in order to speed up the tracing process.

For this reason the ray shooting problem is one of the more widely studied problems in computational geometry. In the plane this has led to many efficient solutions, both for general scenes (where the objects are arbitrary line segments [1, 10, 17, 21] or curved segments [3]) and for special cases (such as ray shooting inside a simple polygon [5, 8]).

In 3–dimensional space, however, the ray shooting problem is still far from resolved. When the origin of the query ray is fixed and the objects are the faces of a polyhedral terrain, then an efficient solution exists [12]. For arbitrary query rays, we know of only three results in the literature.

The first result is due to Schmitt, Müller and Leister [25]; they show that a set of axis-parallel polyhedra in space can be preprocessed into a data structure of size $O(n^3 \text{ polylog } n)$ such that a query takes $O(\log^3 n)$ time. They also present an $O(n \text{ polylog } n)$ size structure with $O(n^{0.695})$ query time.

The second result is by Chazelle et al. [7]. They have shown how to preprocess a polyhedral terrain into a structure of size $O(n^{2+\epsilon})$ such that ray shooting queries take $O(\log^2 n)$ time.

Finally, there is a result by Pellegrini [23], who gives a structure for ray shooting in a set of non-intersecting triangles. His structures uses $O(n^{5+\epsilon})$ preprocessing, and it has $O(\log n)$ query time[1].

In this paper we improve these results, and we also obtain new results for other classes of objects.

First, we consider ray shooting in axis-parallel polyhedra. In this case it is possible to obtain $O(\log^5 n)$ query time after $O(n^{2+\epsilon})$ preprocessing, by using the recently developed recursive partition trees of Chazelle et al. [9] instead of the conjugation trees of Dobkin and Edelsbrunner [14] in the second structure of [25]. (In fact, anything between near-linear storage and roughly $O(\sqrt{n})$ query time and near-quadratic storage and polylogarithmic query time is possible.) We take a different approach and obtain a structure using the same amount of preprocessing time and space, namely $O(n^{2+\epsilon})$, but with a query time of only $O(\log^2 n)$. The solution extends to polyhedra with faces having only g different inclinations. The query time then becomes $O(g^2 \log^2 n)$, while the amount of preprocessing remains $O(n^{2+\epsilon})$. An interesting subproblem that we solve is the stabbing-counting problem for axis-parallel faces. We show that after $O(n^{2+\epsilon})$ preprocessing it is possible to count in $O(\log n)$ time the number of faces intersected by a query line. We also show how to solve

[1] Pellegrini's solution in [23] is incomplete. An improved and corrected result can be found in [24]. We have recently learned that Agarwal and Sharir [2] also study the general ray shooting problem, and give an $O(n^{1+\epsilon})$ preprocessing, $O(n^{4/5})$ query time solution. Finally, we note that Pellegrini [24] has obtained results on ray shooting in axis-parallel polyhedra, which are similar to the results of this paper.

the ray shooting problem for axis-parallel polyhedra in $(\log n)$ time after $O(n^{2+\epsilon})$ preprocessing using yet another method; this method, however, is much more complicated.

The second class of objects we consider is the class of *curtains*. A curtain is an unbounded polygon in space with three edges, two of which are parallel to the z-axis and extend to minus infinity. Thus the polygon can be viewed as an infinite curtain hanging from the third, bounded, edge. Our solution uses $O(n^{2+\epsilon})$ preprocessing and has $O(\log n)$ query time. If we hang curtains from the edges of a polyhedral terrain, then by ray shooting in this set of non-intersecting curtains we can obtain the answer to the ray shooting query in the terrain. Furthermore, we allow curtains to intersect; thus, curtains can be considered as a generalization of a polyhedral terrain. Notice that the query time we achieve is better than that achieved for terrains in [7], while the amount of preprocessing is the same.

Thirdly, we study the ray shooting problem in a set of fat horizontal triangles, i.e., a set of triangles that are parallel to the xy-plane in which all angles of the triangles are greater than some fixed constant. Again, an $O(n^{2+\epsilon})$ preprocessing, $O(\log n)$ query time structure is given.

After studying these special cases, we return to the general ray shooting problem. It is shown how ray shooting queries in an arbitrary set of (possibly intersecting) triangles can be answered in $O(\log n)$ time after $O(n^{4+\epsilon})$ preprocessing.

Another basic problem in computer graphics is the hidden surface removal problem: Given a set of objects in space (typically non-intersecting polyhedra), compute which parts of the polyhedra can be seen by an observer standing at a given view point. More precisely, we want to compute the visibility map of the scene, i.e., the subdivision of the viewing plane into maximally connected regions such that in each region exactly one face of a polyhedron is visible or no face at all is visible. Until very recently, no output-sensitive algorithm (algorithms whose complexity depends not only on n, the total number of edges of the polyhedra, but also on k, the complexity of the visibility map) was known for this problem except when there is a known depth order on the faces. Since cyclic overlap can occur at many places a depth order does not always exist. Furthermore, even if there is no cyclic overlap it is hard to compute a valid depth order. (See [6] for an initial study of these problems.) Hence, the restriction to scenes for which there is a known depth order is a severe one. De Berg and Overmars [13] have shown that a depth order is not necessary to obtain an output-sensitive algorithm if the polyhedra are axis-parallel. Using some of their ideas we give the first output-sensitive hidden surface removal algorithm for arbitrary (non-intersecting) polyhedra. Our algorithm uses the ray shooting structure for curtains. It runs in $O(n^{1+\epsilon}\sqrt{k})$ time. Thus it is quasi-optimal for very small (near-constant) values of k as well as for very large (near-quadratic) values of k.

2 Ray shooting

In this section we study four versions of the ray shooting problem: ray shooting in axis-parallel polyhedra, in curtains, in fat horizontal triangles and, finally, the general case of arbitrary (possibly intersecting) triangles. We will first describe data structures that have an $O(\log^2 n)$ query time. Then we show how the query time can be reduced to $O(\log n)$, without changing the asymptotic preprocessing time.

Before we proceed, it is convenient to introduce some notation and to state a technical lemma that we use repeatedly. This lemma allows us to build recursive data structures in an efficient way.

Lemma 1 *Let $\varepsilon > 0$ be a constant, let r be some sufficiently large parameter, and let $S(n) = O(r^2 n^{2+\varepsilon}) + O(r^2)S(\frac{n}{r})$ where $S(O(1)) = O(1)$. Then $S(n) = O(r^2 n^{2+\varepsilon})$.*

Here r being sufficiently large means that r is larger than some constant which depends on ε and the constants involved in the recurrence itself. The (inductive) proof of this lemma is straightforward and therefore omitted. Note that r can be a function of n, for example $r = n^\delta$ for some $\delta > 0$. This lemma (and also some variations of it) will be used in connection with the following result of Matoušek [19] on *cuttings* of sets of hyperplanes. Define a $(\frac{1}{r})$-cutting of a set S of hyperplanes in d–space to be a subdivision of d-space into simplices such that any simplex is intersected by at most $\frac{n}{r}$ of the hyperplanes in S. The size of the cutting is the number of simplices in the cutting.

Lemma 2 (Matoušek [19]) *Given a set S of n hyperplanes in d–space, there exists a $(\frac{1}{r})$-cutting $\Xi(S)$ of size $O(r^d)$. Moreover, such a cutting can be computed deterministically in time $O(nr^{d-1})$ for $r < n^{1-\delta}$.*

In the remainder of this section the query ray is denoted by ρ. The point $p = (p_x, p_y, p_z)$ is the starting point of ρ and $l(\rho)$ denotes the line containing ρ. The projection of an object o onto the xy-plane is denoted $\overline{o}$. Finally, we say that a segment e in space, also called a *rod*, *passes above* a rod e' iff $\overline{e} \cap \overline{e'} \neq \emptyset$ and 'at this intersection point' e has greater z-coordinate. The notion of 'passing above' is defined similarly for lines with respect to rods, rays with respect to lines, etcetera.

2.1 Axis-parallel polyhedra

Let S be a set of axis-parallel polyhedra with n edges in total and let F be the set of faces of the polyhedra in S. We want to find the first face that is hit by some (not necessarily axis-parallel) query ray ρ. To this end we split F into three subsets, F_1, F_2 and F_3, that contain the faces parallel to the yz-plane, the xz-plane and the xy-plane. For each subset we build a separate structure. A query is performed in all three structures; of the (at most) three faces we find we then select the one that is intersected first. Next we show how to preprocess F_1 for efficient ray shooting; F_2 and F_3 can be handled in the same way.

Because a face f in F_1 is parallel to the yz-plane, f has one specific x-coordinate, denoted f_x. The first level of the data structure is just a balanced binary tree T storing the x-coordinates of the faces in increasing order from left to right in its leaves. With each node δ in this tree we associate a structure that can answer the following query on the set F_1^δ of faces that are stored in the subtree rooted at δ: 'Given a query line l, does it stab at least one of the faces in F_1^δ ?'.

Before we turn our attention to the implementation of the associated structures, let us describe how to use this structure to answer a ray shooting query. Assume w.l.o.g. that ρ is directed to the right, i.e., in the positive x-direction. If the search path of p_x (the x-coordinate of the starting point of ρ) in T turns right at the root of T, then ρ cannot stab any face in the left subtree, so we only have to search recursively in the right subtree. If the search path turns left, then we search recursively in the left subtree. If we find an answer in the left subtree then this will be the answer to the ray shooting query. If ρ misses all faces in the left subtree then we also have to search in the right subtree. But in that case we know that the starting point of ρ lies to the left of all faces in the right subtree. Therefore the search can be done as follows. Starting at the right child of the root, we walk down the tree. Using the associated structures we test if the left subtree of the current node contains at least one face that is stabbed by $l(\rho)$ (and, hence, by ρ); if this is the case then we turn to the left, otherwise we turn to the right. The search will end in the leaf that contains the answer to the ray shooting query (or we find out that ρ misses all faces). The query algorithm visits at most two nodes at every level of the tree. So after $O(\log n)$ queries in associated structures we have found the answer to the ray shooting query.

We are left with the following subproblem: Preprocess a set of axis-parallel faces that are parallel to the yz-plane—let's call this set A—to decide efficiently whether a query line stabs at least one of the faces. In fact, a more general structure will be presented: instead of telling us if at least one face is stabbed, it answers *stabbing-counting queries*, i.e., it can tell us exactly how many faces are stabbed.

Consider a face $f \in A$. A *bottom edge* of f is an edge that bounds f from below, and a *top edge* is an edge that bounds f from above. For any line l that stabs f, we know that the number of bottom edges of f above which l passes is one greater than the number of top edges of f above which l passes. Similarly, a line l' that does not stab f passes above an equal number of bottom and top edges. This leads to the following observation. Let $\sigma(l, A)$ be the number of faces in A stabbed by a line l. Let E_A^b and E_A^t be the set of bottom and top edges of the faces in A, and let $\phi(l, E_A^b)$ and $\phi(l, E_A^t)$ be the number of bottom and top edges passing below l.

Observation 1 $\sigma(l, A) = \phi(l, E_A^b) - \phi(l, E_A^t)$

Our strategy will be to store the sets E_A^b and E_A^t such that $\phi(l, E_A^b)$ and $\phi(l, E_A^t)$ can be computed efficiently. Consider the set E_A^b of bottom edges; E_A^t can be handled in the same way. Recall that to pass above an edge $e \in E_A^b$, the projection $\bar{l}$ of l onto

the xy-plane has to intersect the projection $\overline{e}$ of e. If this is the case then l passes either above or below e. To distinguish between these two cases we project l and e onto the xz-plane; the projection $\tilde{l}$ of l is a line and the projection $\tilde{e}$ of e is a point. Now l passes above e if and only if $\tilde{e} \in \tilde{l}^-$, where $\tilde{l}^-$ is the half-plane below $\tilde{l}$.

This leads to the following structure. Let $\overline{E}_A^b = \{\overline{e} \mid e \in E_A^b\}$ be the set of projections of edges in E_A^b onto the xy-plane. For an edge $\overline{e}$ let $\overline{e}^*$ denote its dual, which is a double wedge, and let $W = \{\overline{e}^* \mid e \in E_A^b\}$. (We use the standard duality transform, described for example by Edelsbrunner [16], that maps points to lines and vice versa.) Let $\Xi(W)$ be a $(\frac{1}{r})$-cutting for the lines that define the double wedges in W, where r is a parameter to be determined later. For a cell c in $\Xi(W)$, let $W_1(c)$ be the subset of double wedges that fully contain c and let $W_2(c)$ be the subset of double wedges that partially cover c. Since $\Xi(W)$ is a $(\frac{1}{r})$-cutting we know that $|W_2(c)| \leqslant \frac{n}{r}$ for each cell c, where $n = |W|$. Our structure can be seen as a tree of branching degree $O(r^2)$. The root of this tree stores the subdivision $\Xi(W)$, preprocessed for point location queries using e.g. Kirkpatricks method [18]. Furthermore, for each cell c in $\Xi(W)$ the set of points $\{\tilde{e} \mid \overline{e}^* \in W_1(c)\}$ is stored, preprocessed for half planar range counting as described in [9]. This half planar range counting structure uses $O(|W_1(c)|^2 \log |W_1(c)|)$ preprocessing time and $O(|W_1(c)|^2)$ space and it allows us to count the number of points in $\{\tilde{e} \mid \overline{e}^* \in W_1(c)\}$ below a query line in $O(\log n)$ time. Finally, the $O(r^2)$ children of the root correspond to recursively defined structures on the set $W_2(c)$.

Next it is described how to count the number of edges in E_A^b passing below a query line l with this structure. First $\overline{l}^*$, the dual of the projection of l onto the xy-plane, is located in the subdivision $\Xi(W)$ that is stored at the root of the structure. Let c be the cell containing $\overline{l}^*$. Then we perform a query with $\tilde{l}^-$, the half-plane below the projection of l onto the xz-plane, in the half planar range counting structure that stores $\{\tilde{e} \mid \overline{e}^* \in W_1(c)\}$. This gives us the number of edges in $\{e \mid \overline{e}^* \in W_1(c)\}$ that pass below l. However, $\{e \mid \overline{e}^* \in W_2(c)\}$ can contain edges that pass below l as well, so we recurse in the child of the root corresponding to cell c. This way we compute $\phi(l, E_A^b)$. A similar structure allows us to count $\phi(l, E_A^t)$. Using Observation 1 we obtain:

Theorem 1 *Stabbing-counting queries in a set of axis-parallel faces with n edges in total can be performed in time $O(\log n)$ with a structure that uses $O(n^{2+\varepsilon})$ preprocessing time and space, for any fixed $\varepsilon > 0$.*

Proof: The correctness of the approach follows from the discussion above. It remains to analyze the query time and the preprocessing. To compute $\phi(l, E_A^b)$ we first perform a point location in the subdivision $\Xi(W)$ associated with the root, taking $O(\log r)$ time. Then we perform a half planar range query, which costs $O(\log n)$ time, and then we recurse. Hence, the query time $Q(n)$ satisfies the recurrence $Q(n) = O(\log r) + O(\log n) + Q(\frac{n}{r})$. The preprocessing time $S(n)$ can be seen to satisfy $S(n) = O(r^2 n^2 \log n) + O(r^2) S(\frac{n}{r})$. The query time and the bounds on the

preprocessing (use Lemma 1) follow if we set $r = n^{\epsilon'}$ for a sufficiently small $\epsilon' > 0$. □

Let us now return to our original ray shooting problem. Recall that we have a balanced binary tree on the x-coordinates of the faces. At every node δ we have an associated structure for stabbing-counting queries on the set of faces whose x-coordinate is stored in the subtree rooted at δ. To answer a ray shooting query we have to perform stabbing-counting queries in $O(\log n)$ associated structures. Using the structure of Theorem 1 for the associated structures we obtain:

Theorem 2 *Ray shooting queries in a set of axis-parallel polyhedra with n edges in total can be performed in time $O(\log^2 n)$ with a structure that uses $O(n^{2+\epsilon})$ preprocessing time and space, for any fixed $\epsilon > 0$.*

This result can be generalized to polyhedra whose faces have g different inclinations. To this end, we partion the set F of faces of the polyhedra into g subsets $F_1, \ldots, F_g$, one for each inclination. For each subset we build a separate structure. Consider the faces in some subset F_i, and assume w.l.o.g. that these faces are parallel to the yz-plane. The main structure is a tree on the x-coordinates of the faces, and the secondary structure is a structure for stabbing counting queries. Hence, the structure is similar to the structure for axis-parallel faces; the only difference is that the structure for stabbing-counting queries now consists of $g-1$ 'substructures', one for each possible orientation of the edges. (Notice that the edges that are stored in some stabbing-counting structure are the intersection of a face in F_i and some other face, and, hence, they have only $g-1$ different orientations.) Each such substructure is identical to the stabbing-counting structure described above for the axis-parallel case. Thus, stabbing counting queries now take $O(g \log n)$ time, and ray shooting queries in F_i take $O(g \log^2 n)$ time. To answer a ray shooting query, we search the subsets F_i separately; the answer to the ray shooting query is easily computed from the subanswers in $O(g)$ time. This results in a total query time of $O(g^2 \log^2 n)$. Since each edge is present in exactly two structures (one for each face that is incident to it), the preprocessing time and space are independent of g and remain $O(n^{2+\epsilon})$.

Theorem 3 *Ray shooting queries in a set of polyhedra with n edges in total whose faces have g different inclinations can be performed in time $O(g^2 \log^2 n)$ with a structure that uses $O(n^{2+\epsilon})$ preprocessing time and space, for any (fixed) $\epsilon > 0$.*

Remark: As was already noted in the introduction, it is possible to get a trade-off between query and preprocessing time, by using the recursive partition trees of Chazelle et al. [9] in the method of Schmitt et al. [25]. More precisely, it is possible to achieve $O(n^{1+\epsilon}/\sqrt{m})$ query time using $O(m^{1+\epsilon})$ preprocessing, for any $n < m < n^2$. It should also be noted that these bounds lead to an improvement over Pellegrini's result on batched ray shooting for axis-parallel polyhedra [23].

2.2 Curtains

A *curtain* is an unbounded polygon in space with three edges, two of which are parallel to the z-axis and extend to $z = -\infty$. Thus the polygon can be seen as an infinitely long curtain hanging from the third (bounded) edge, which we call its top edge. Observe that two curtains can intersect each other. Let S be a set of n such curtains. We want to preprocess S for ray shooting queries. As in the case of axis-parallel polyhedra we first reduce the problem to a stabbing problem: 'Does a given line intersect at least one curtain?' (This time, however, we are unable to compute the exact number of curtains stabbed by the line. In fact, if we could devise a structure for stabbing-counting queries in a set of curtains, it would allow us to solve the ray shooting problem for an arbitrary set of triangles efficiently.)

To reduce the ray shooting problem to a stabbing problem we would like to impose an order on the curtains. Since they can intersect, however, it seems hard to obtain such an order in an efficient way. Fortunately, the cuttings that we have used before are also useful in this respect. Project the curtains onto the xy-plane, obtaining a set $\overline{S}$ of segments. Construct a $(\frac{1}{r})$-cutting $\Xi(\overline{S})$ for the lines that contain the segments in $\overline{S}$, where r is some sufficiently large constant. For a cell c in $\Xi(\overline{S})$, let $S(c)$ be the set of curtains whose projections intersect c. (More precisely, we restrict our attention to that part of each curtain that projects onto c.) The main structure is a tree T of degree $O(r^2)$. With the root we associate $\Xi(\overline{S})$ and for each cell c we have an associated structure that can tell us whether at least one curtain in $S(c)$ is stabbed by some query line. Furthermore, every child of the root corresponds to a recursively defined structure for some set $S(c)$.

A query in this structure is performed in much the same way as in the axis-parallel case. The projection $\overline{\rho}$ of the query ray intersects $O(r^2)$ cells $c_1, c_2, \ldots, c_t$ of $\Xi(\overline{S})$ that are stored at the root of T; the cells are numbered according to the order in which they are intersected by $\overline{\rho}$. See Figure 1. We recursively find the first curtain in $S(c_1)$ that is hit. If there is such a curtain, then this must be the answer to the query. If none of the curtains is intersected, then we do the following. Using the associated structures we test if ρ stabs at least one curtain in $S(c_2)$, if not we test $S(c_3)$, etcetera, until we find the first cell c_i such that ρ stabs at least one curtain in $S(c_i)$. This set contains the answer, so we recurse in the corresponding child of the root. This process is repeated until we reach the leaf of T that contains the answer. The reason that we treat c_1 separately is that $\overline{\rho}$ does not cut completely through c_1, so we cannot replace ρ by $l(\rho)$ inside c_1. However, when none of the curtains in $S(c_1)$ is intersected, then this problem will not occur again. From this it follows that at each level of our tree we visit only $O(r^2)$ nodes. Since the depth of the tree is $O(\log_r n)$ and we chose r to be a constant, it follows that we perform $O(\log n)$ queries in associated structures in total.

What remains is to devise a structure for the stabbing problem: 'Does a query line l stab at least one curtain in some set A?' This structure is closely related to

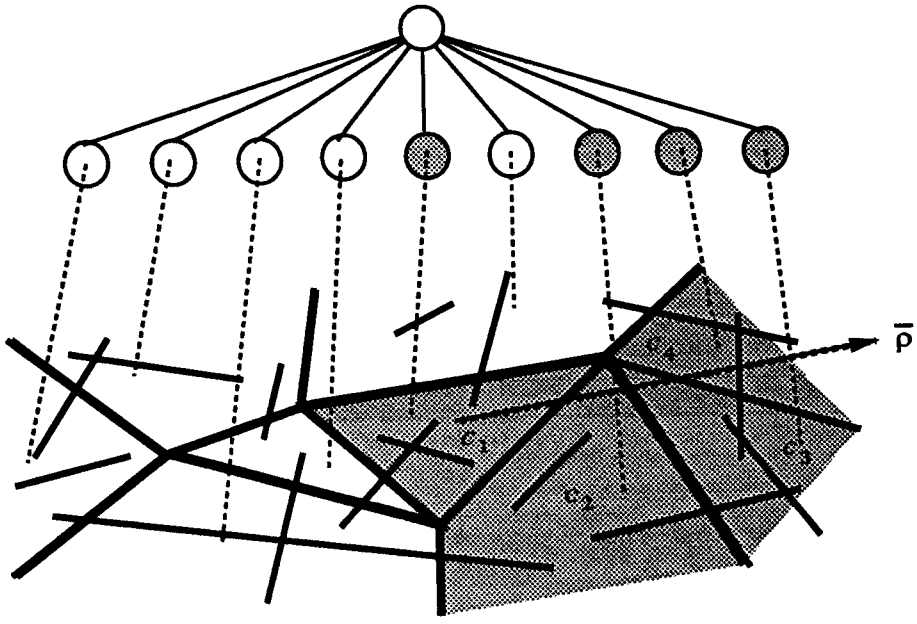

Figure 1: The tree corresponding to a cutting.

the structure for stabbing-counting queries in axis-parallel faces that was described in Section 2.1. We project the curtains in A onto the xy-plane, resulting in a set $\overline{A}$ of segments, and dualize this set to obtain a set W of double wedges. Next we construct a $(\frac{1}{r'})$-cutting $\Xi(W)$ for the lines defining the double wedges in W. The structure that stores A is a tree of degree $O((r')^2)$. With the root we store the subdivision $\Xi(W)$, preprocessed for point location queries, and each child of the root corresponds to a recursively defined structure for the set $A_2(c)$ of curtains whose corresponding double wedges cross the cell c. For each cell c we also store a structure that can test if a query line stabs at least one of the curtains in $A_1(c)$, the set of curtains corresponding to double wedges that fully contain c. This structure is defined as follows. Because we know that the projection of a query line that visits c intersects the projection of each curtain in $A_1(c)$, we can extend the top edges of these curtains to full lines. Now a query line l stabs none of the curtains if and only if it passes above all these lines. This can be tested in $O(\log n)$ time after $O(n^{2+\varepsilon})$ preprocessing, for any $\varepsilon > 0$, see [7]. We thus can test if a query line stabs at least one curtain in A in time $O(\log n)$ after $O(n^{2+\varepsilon})$ preprocessing, for any $\varepsilon > 0$, if we set $r' = n^{\varepsilon'}$ for an appropriate value of ε' and apply Lemma 1. This concludes the description of the associated structure of our main tree.

Summarizing, the ray shooting structure consists of three levels. The first level recursively imposes an ordering on the curtains and thus reduces the shooting queries to stabbing queries. The second level is used to filter out the curtains that are

intersected in the projection so that the top edges can be extended to lines. The third level then answers the stabbing queries for these extended curtains. We have to perform $O(\log n)$ stabbing queries to find the answer to the ray shooting query. Since the stabbing queries take $O(\log n)$ time and the depth of the trees on the second level is constant, the total query time is $O(\log^2 n)$. Using Lemma 1 once more, the preprocessing time of the structure is seen to be $O(n^{2+\epsilon})$ for a, by this time fairly large, $\epsilon > 0$ that can still be chosen arbitrarily small.

Theorem 4 *Ray shooting queries in a set of n curtains can be performed in time $O(\log^2 n)$ with a structure that uses $O(n^{2+\epsilon})$ preprocessing time and space, for any fixed $\epsilon > 0$.*

2.3 Fat horizontal triangles

We call a triangle *fat* if all its internal angles are greater than some fixed constant θ. Fat *horizontal* triangles, that is, triangles parallel to the xy-plane, have the following important property.

Observation 2 *There exists a set of slopes $\mathcal{D}$ of constant size, such that, for each vertex v of any fat horizontal triangle t, it is possible to split t into at most two (non-empty) triangles with a segment incident to v whose slope is in $\mathcal{D}$.*

The size of $\mathcal{D}$ is inversely proportional to the minimum angle θ of the triangles. For example, we can take the set $\mathcal{D} = \{i\theta/2 : 0 \leqslant i < 4\pi/\theta\}$. Let S be a set of n fat horizontal triangles. The property stated above enables us to decompose each triangle $t \in S$ into at most four triangles t_1, t_2, t_3 and t_4 such that each t_i has two edges whose slopes are in $\mathcal{D}$: Pick any vertex of t and split t according to Observation 2 using some segment s. Split the two resulting triangles from the vertices opposite s, see Figure 2. This clearly results in four triangles that each have

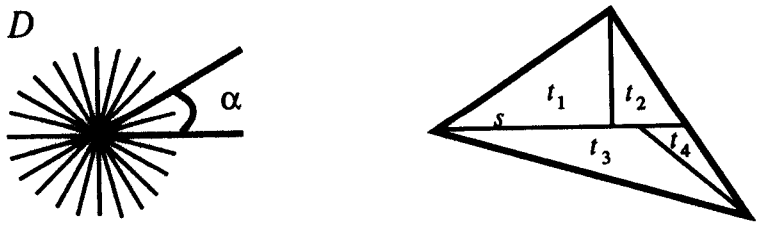

Figure 2: Splitting a fat triangle using segments with slope in $\mathcal{D}$.

two edges with slope in $\mathcal{D}$. We call these edges the *fixed edges* of the triangles. Next we partition the set of $4n$ triangles thus obtained into $|\mathcal{D}|^2$ subsets $S_1, \ldots, S_{|\mathcal{D}|^2}$; two triangles are in the same subset if and only if the two fixed edges of one triangle

are parallel to the two fixed edges of the other triangle. For each S_i a separate structure is built. A query is performed in all structures and the final answer to the ray shooting query is easily computed from the $|\mathcal{D}|^2$ 'subanswers' that are found.

Consider one subset S_i. Assume w.l.o.g. that each triangle $t \in S_i$ has one edge that is parallel to the x-axis, and one edge that is parallel to the y-axis. Assume that the triangles lie above (i.e. in the positive y-direction of) the edge that is parallel to the x-axis; the triangles that lie below this edge are treated separately. For a triangle t, we call the edge that is parallel to the x-axis its *bottom edge*, the edge that is parallel to the y-axis its *vertical edge*, and its third edge, which does not have a fixed slope, its *top edge*. The idea of the structure is as follows. First we select all triangles t such that $l(\rho)$ passes in the y-direction above the line containing the bottom edge of t. Once we know that $l(\rho)$ passes above the bottom edge of these triangles, we can as well extend them to $y = -\infty$. In other words, we can regard each triangle t as a curtain hanging from its top edge into the negative y-direction (which is the direction of its vertical edge). Thus, if we can find all triangles whose bottom edges pass below a query line efficiently, we can use the structure developed in the previous section.

How do we find these triangles quickly, and, equally important, in a small number of groups? Here we use the fact that all bottom edges are parallel to the x-axis. So the idea that was used in the axis-parallel case applies: we project the set $E^b_{S_i}$ of bottom edges of the triangles in S_i onto the yz-plane, giving a set $\tilde{E}^b_{S_i}$ of points. A line l passes above the line containing a bottom edge $e \in E^b_{S_i}$ if and only if $\tilde{e} \in \tilde{l}^-$, where $\tilde{e}$ and $\tilde{l}$ are the projections of e and l onto the yz-plane, and $\tilde{l}^-$ denotes the half-plane below $\tilde{l}$ (i.e. in the negative y-direction of $\tilde{l}$). To find all points $\tilde{e} \in \tilde{l}^-$ for a query line l we can use the same technique that we have used before: we dualize the set of points $\tilde{E}^b_{S_i}$, and construct a $(\frac{1}{r})$-cutting $\Xi((\tilde{E}^b_{S_i})^*)$ for the resulting set $(\tilde{E}^b_{S_i})^*$ of lines. The subdivision $\Xi((\tilde{E}^b_{S_i})^*)$, preprocessed for point location, is stored at the root of our main structure, which is an $O(r^2)$-ary tree. For each cell c of $\Xi((\tilde{E}^b_{S_i})^*)$ we have an associated structure on the set of triangles that correspond to the lines below c, and we recursively store the at most $\frac{n}{r}$ lines that intersect c. The associated structure is a ray shooting structure, as described in the previous section, on the set of curtains hanging from the top edges of the triangles into negative y-direction. Choosing r to be $n^{\varepsilon'}$ for an appropriately small $\varepsilon' > 0$, we get the triangles for which the bottom line passes below $l(\rho)$ in a constant (dependent of ε') number of groups. Thus the total query time will be the same as the query time for the curtains, which is $O(\log^2 n)$ by Theorem 4. Using the fact that the ray shooting structure for curtains uses $O(n^{2+\varepsilon})$ preprocessing, the preprocessing can be done in time $O(n^{2+\varepsilon})$ for any (slightly larger) $\varepsilon > 0$, see Lemma 1.

Theorem 5 *Ray shooting queries in a set of n fat horizontal triangles can be performed in time $O(\log^2 n)$ with a structure that uses $O(n^{2+\varepsilon})$ preprocessing time and space, for any fixed $\varepsilon > 0$.*

Observe that, using the same techniques, it is possible to obtain an alternative solution for the ray shooting problem in a set of axis-parallel polyhedra. This is true because each face of an axis-parallel polyhedron can be split into rectangles whose edges have a fixed orientation. These rectangles can be treated in the same way as the triangles that have two fixed edges: first select the ones whose bottom edge passes below the query line, and then treat the rectangles as curtains hanging from the top edge into the direction of the, in this case two, vertical edges.

2.4 The general case

This section tackles the general ray shooting problem, which is to preprocess a set of possibly intersecting triangles in space for efficient ray shooting. First, we impose an ordering on the triangles in the same manner as we did in Section 2.2 for curtains. Once we have imposed the ordering, the ray shooting problem reduces to a stabbing problem: 'Does a query line stab at least one triangle of a given set of triangles?'

Let S be a set of n possibly intersecting triangles in space. Project the triangles in S onto the xy-plane, obtaining a set $\overline{S}$. Next, construct a $(\frac{1}{r})$-cutting $\Xi(\overline{S})$ for the $3n$ lines containing the edges of the projected triangles, for a sufficiently large constant r. Associate with each cell c in $\Xi(\overline{S})$ the portions of the projected triangles that intersect c. For a query ray ρ we have to test the cells $c_1, c_2, \ldots$ of $\Xi(\overline{S})$ that are intersected by $\overline{\rho}$, until we find the first c_i that contains a triangle stabbed by ρ. We handle the triangles whose projection fully contains c_i at this level and we recursively search in the at most $\frac{n}{r}$ triangles whose projected boundary intersects c_i.

This approach leads to two subproblems that we have to solve. Firstly, we have to be able to decide if a query line stabs at least one triangle of a given set. Secondly, we have to treat in an efficient manner the triangles whose projection fully contains a cell c. Observe that the second subproblem did not occur when we studied curtains, since a curtain projects onto a segment.

The first subproblem can be solved using Plücker coordinates, as in [23]. To this end, the lines through the edges of the triangles are oriented and mapped to hyperplanes in Plücker 5–space, and an (oriented) query line is mapped to a point. The position of the point—whether it is above, on or below—relative to a hyperplane determines the 'twist' of the query line—whether it is clockwise, intersecting, or counterclockwise—with respect to the line corresponding to the hyperplane. See [7, 26] for more details. Thus, if we consider the arrangement of three hyperplanes corresponding to the lines through the edges of a triangle, then there are exactly two cells corresponding to query lines that stab the triangles. One cell corresponds to lines that are oriented such that they stab the triangle from front to back, and the other cell corresponds to lines stabbing the triangle from back to front. Hence, point location with the Plücker point of the query line in the subdivision of hyperplanes corresponding to all lines through triangle edges tells us which triangles are stabbed. Moreover, only the cells of this subdivision that intersect the Plücker hypersurface

(the hypersurface containing the images of all lines in 3–space, also called the Grassman manifold) are of interest. Recently, Aronov and Sharir [4] have shown that the total complexity of all these cells is $O(n^4 \log n)$. Therefore, the point location can be done in $O(\log n)$ time after $O(n^{4+\varepsilon})$ preprocessing, in the same way as the point location method for arrangements of hyperplanes, as described by Clarkson [11]: take a sample $R \subset H$ of size $O(r)$ such that any cell in the triangulated arrangement $\mathcal{A}(R)$ is intersected by no more than $\frac{n}{r} \log r$ hyperplanes of H, for a sufficiently large constant r. Because a random sample has this property with high probability, such a sample can be found in $O(nr^5)$ expected time [11]. Consider a cell in $\mathcal{A}(R)$ and some triangle. If none of the three Plücker hyperplanes that correspond to this triangle intersect the cell, then we know that either any line whose Plücker point lies in this cell intersects the triangle, or any such line misses the triangle. If one or more of the Plücker hyperplanes intersect the cell, then some lines whose Plücker point is inside the cell intersect the triangle, while other lines miss it. Thus, the question if at least one triangle is hit by a query line can be answered with the following tree. The root of the tree stores the cells of the triangulated arrangement $\mathcal{A}(R)$ that are intersected by the Plücker hypersurface. Each child of the root correspond to such a cell. Thus, the root has $O(r^4 \log r)$ children. If for such a cell there is a triangle that is intersected by all lines whose Plücker points are in the cell, then the child corresponding to the cell is a leaf. Otherwise, we recursively store the at most $\frac{n}{r} \log r$ triangles that are sometimes intersected at this child. Thus $S(n)$, the preprocessing time and space, satisfies $S(n) = O(nr^5) + O(r^4 \log r) S(\frac{n}{r} \log r)$, which solves to $S(n) = O(n^{4+\varepsilon})$, for any $\varepsilon > 0$. To answer a query we locate in time $O(r^4 \log r)$ the cell of $\mathcal{A}(R)$ that contains the Plücker point of the query line. If this cell corresponds to a leaf of the tree, then we know the answer. Otherwise, we have to search recursively in the child corresponding to this cell. Since r is a constant, the search takes $O(\log n)$ time.

In the second subproblem we are given a cell c of $\Xi(\overline{S})$ and a set $S(c)$ of triangles whose projections fully contain c, restricted to the portions that project onto c. Given a query ray ρ whose projection intersects c, we want to find the first triangle in $S(c)$ hit by ρ. Since the projections of the triangles in $S(c)$ fully contain c, we are, in effect, ray shooting in the arrangement of the planes containing the triangles. Let us describe a simple solution for this problem, that has a query time of $O(\log n)$ and which uses $O(n^{3+\varepsilon})$ preprocessing. First, we build a point location structure for the planes containing the triangles; an easy way to do this is using random sampling, see [11]. Every leaf in this structure corresponds to a cell in the arrangement of planes. We construct this full arrangement in $O(n^3)$ time [16]. Each cell in the arrangement is preprocessed in linear time (linear in the size of the cell) for $O(\log n)$ time ray shooting queries using the hierarchical representation of Dobkin and Kirkpatrick [15]. The only problem that is left is to associate these ray shooting structures with the right leaves in the point location structure. To this end we take a point in each cell and search with it in the point location structure; the

ray shooting structure for this cell is associated with the leaf in which the search ends. A ray shooting query in the set $S(c)$ now proceeds as follows. First we search with the starting point of the ray (more precisely, the starting point of the part of the ray projecting onto c) in the point location structure in $O(\log n)$ time. Then a ray shooting query is performed in the structure that is associated with the leaf in the point location structure, also taking $O(\log n)$ time. We now have found the first plane that is hit in the arrangement of planes containing the triangles. Finally, we test if the projection of this first intersection lies in c. If so, the ray will also hit the triangle that corresponds to this plane; otherwise none of the triangles in $S(c)$ is hit.

Returning to the original problem, we see that the query time is $O(\log^2 n)$ (this follows in the same way as the query time for curtains) and that the preprocessing time satisfies

$$S(n) = O(r^2(\frac{n}{r})^{4+\varepsilon}) + O(r^2 n^{3+\varepsilon}) + O(r^2)S(\frac{n}{r}),$$

which leads to $S(n) = O(n^{4+\varepsilon})$.

Theorem 6 *Ray shooting queries in a set of n possibly intersecting triangles in space can be performed in time $O(\log^2 n)$ with a structure that uses $O(n^{4+\varepsilon})$ expected preprocessing time and $O(n^{4+\varepsilon})$ space, for any fixed $\varepsilon > 0$.*

2.5 Reducing the query time

Next it is shown that the query time for the ray shooting problems studied above can be reduced to $O(\log n)$ without affecting the preprocessing time asymptotically. The new structures, however, are much more complicated.

Let us first consider the general ray shooting problem. Thus, we are given a set S of possibly intersecting triangles in space. The first step in devising the structure remains the same: we project the triangles onto the xy-plane and we compute a $(\frac{1}{r})$-cutting $\Xi(\overline{S})$ for the lines containing the projections of the edges of the triangles. The main idea behind the reduction in query time is to choose the parameter r to be n^ε for a sufficiently small $\varepsilon > 0$ instead of choosing r to be constant. This way the depth of the recursion becomes constant instead of logarithmic. On the other hand, this also means that we cannot afford to check all of the $O(r^2)$ cells of $\Xi(\overline{S})$ that are intersected by the projection $\overline{\rho}$ of the query ray, to see in which cell we have to recurse. Thus, all these cells have to be tested simultaneously.

Let E be the set of $O(r^2)$ edges of the subdivision $\Xi(\overline{S})$. Let $W = \{e^* \mid e \in E\}$ be the set of double wedges dual to the edges in E. Finally, let $\mathcal{A}(W)$ be the arrangement on the dual plane defined by the double wedges in W. Note that $\mathcal{A}(W)$ has size $O(r^4) = O(n^{4\varepsilon})$. A cell of $\mathcal{A}(W)$ corresponds to a fixed subset of the edges in E that are stabbed by lines whose dual points are in this cell. Hence, a cell in $\mathcal{A}(W)$ also corresponds to a fixed set of cells in $\Xi(\overline{S})$ intersected by such a line. Now consider a query ray ρ. Let $c_1, \ldots, c_t$ be the $O(r^2)$ cells of $\Xi(\overline{S})$ that

are intersected by its projection $\overline{p}$. These cells can be found in the following way. Locate $l(\overline{p})^*$, the dual of the line containing the projection of ρ, in $\mathcal{A}(W)$; then do a binary search with the starting point of $\overline{p}$ on the edges that are intersected by $\overline{p}$. This way we find the cell c_1 that contains the starting point of $\overline{p}$ and also the other cells that are intersected. As before, we always recursively search in c_1. If we find an answer in c_1, then we are finished. Otherwise, the answer is in one of the cells $c_2, \ldots, c_t$. This sequence of cells is called the *suffix* of ρ, and we denote it by σ_ρ. The suffix σ_ρ will be preprocessed such that we can decide in $O(\log n)$ time into which cell of σ_ρ we must recurse, if there is no answer in c_1. Thus, at each of the $O(r^4)$ cells of $\mathcal{A}(W)$ we store $O(r^2)$ data structures, one for each suffix of the cells of $\Xi(\overline{S})$ that are stabbed by the lines whose duals are in this cell.

Consider a suffix $\sigma = c_2, \ldots, c_t$. For cell c_i, let $S(c_i)$ denote the subset of triangles in S whose projection intersects c_i. Thus, both triangles whose projection completely contains c_i and triangles whose projection partially covers c_i are present in $S(c_i)$; of the latter type there are no more than $\frac{n}{r}$. Furthermore, the triangles in $S(c_i)$ are restricted to the parts that project onto c_i. For example, if the projection of a triangle intersects two cells, say c_i and c_j, then this triangle is split into two parts; $S(c_i)$ contains the part that projects onto c_i, and $S(c_j)$ contains the part that projects onto c_j. Hence, different parts of the same triangle can occur in different sets $S(c_i)$. This means that if we stab a certain part of a triangle we know in which cell the (projection of the) intersection will occur. Note that a part of a triangle need not be a triangle itself, but that it can be a (convex) k–gon, for $3 \leqslant k \leqslant 6$. Also note that the total complexity of $S_\sigma = \bigcup_{2 \leqslant i \leqslant t} S(c_i)$ is $O(nr^2)$. Now consider the arrangement in Plücker 5–space induced by the triangle parts in S_σ. This arrangement gives us all the stabbing information about the triangle parts. Hence, by performing a point location in it, we can determine which triangle parts of S_σ are stabbed by a line and thus into which cell of $\Xi(\overline{S})$ we have to recurse. This point location can be done in $O(\log(nr^2)) = O(\log n)$ time after $O((nr^2)^{4+\varepsilon'})$ preprocessing, for any $\varepsilon' > 0$, in the same way as before.

Putting everything together we see that we can find the cell into which to recurse in $O(\log n)$ time. Then we do an $O(\log n)$ time ray shooting query on the triangles whose projection fully contains this cell (clearly we cannot recurse on them), and we recurse on the $\frac{n}{r}$ triangles whose projection partially covers the cell. Since $r = n^\varepsilon$, the depth of the recursion is constant, so the total query time is $O(\log n)$. The preprocessing time $S(n)$ satisfies $S(n) = O(r^6) \times O((nr^2)^{4+\varepsilon'}) + O(r^2)S(\frac{n}{r})$, where $r = n^\varepsilon$. By choosing ε and ε' sufficiently small we obtain:

Theorem 7 *Ray shooting queries in a set of n possibly intersecting triangles in space can be performed in time $O(\log n)$ with a structure that uses $O(n^{4+\varepsilon})$ preprocessing time and space, for any fixed $\varepsilon > 0$.*

To reduce the query time for ray shooting in curtains, we use the same trick. Thus we construct a $(\frac{1}{r})$-cutting $\Xi(\overline{S})$, with $r = n^\varepsilon$. For a query ray ρ whose projection intersects cells $c_1, \ldots, c_t$, we always recurse on c_1. If we don't find an answer there,

then we decide in $O(\log n)$ time in which cell of the suffix $\sigma_\rho = c_2, \ldots, c_t$ we must recurse. So we need to determine the suffix of a query ray, and for each suffix we must be able to determine the cell of the suffix in which we have to recurse.

The structure to compute the suffix for a query ray is the same as in the general case: a point location structure for the arrangement $\mathcal{A}(W)$, where W is the set of double wedges that are the duals of the edges of the cutting $\Xi(\overline{S})$. Hence, we can find this suffix in $O(\log n)$ time. So now consider a suffix $\sigma = c_2, \ldots, c_t$, and let $S_\sigma = \bigcup_{2 \leqslant i \leqslant t} S(c_i)$, where $S(c_i)$ is the set of curtains whose projection intersects c_i. Note that $|S_\sigma| = O(\frac{n}{r}t) = O(nr)$, since each c_i is intersected by $O(\frac{n}{r})$ curtains. First we select the curtains that are intersected by the query ray in the projection. This is done in the usual way: dualize the projections of the curtains to obtain a set of double wedges and compute a $(\frac{1}{r})$-cutting for the lines defining the double wedges, for $r = n^{\varepsilon'}$ and ε' sufficiently small. For each cell in the cutting, we store the curtains whose double wedges fully contain the cell in an associated structure to be described next, and we recursively store the curtains whose double wedges partially cover the cell. Using this structure, it is possible to find the curtains that are intersected by a query line in a constant (depending on ε') number of groups. This is exactly the same as in section 2.2, so we will not go further into details and we concentrate on the associated structure.

The associated structure should solve the following problem. Given an ordered collection of sets of curtains —namely subsets of the sets $S(c_2), \ldots, S(c_t)$— find the first set that contains a curtain that is stabbed by the query line. Furthermore, we know that we only query with lines that intersect all the curtains in the projection. Hence, we can extend the top edges of the curtains to full lines. Thus we are given an ordered collection of sets of lines and we want to know the first set that contains a line that passes above a query line. As mentioned before, the twist of a query line with respect to a given line is determined by the position of the corresponding query point relative to the corresponding hyperplane in Plücker 5–space. If the two lines are consistently oriented—say both from left to right—then the query line passes above the given line if and only if its Plücker point is on one distinguished side of the hyperplane h. For notational convenience, we denote the half-space on this side of h by h^+, and the other half-space by h^-. Now consider a query line that is consistently oriented with respect to a set of given lines. This line passes above all given lines if and only if the Plücker point corresponding to the line lies in the convex polytope $\bigcap h^+$ determined by the hyperplanes h corresponding to the given lines. Chazelle et al. [7] have shown how to enforce the consistency constraint on the orientations of the lines. For our problem we can use their technique directly, without affecting the asymptotic preprocessing and query time, so we will ignore this consistency constraint from now on.

Hence, we have the following problem. Let $H_1, \ldots, H_m$ be an ordered collection of $m = O(n^{2\epsilon})$ sets of hyperplanes in 5–space, with $\sum_{i=1}^m |H_i| = n$. (The bound on the size of $\sum_{i=1}^m |H_i|$ follows from the fact that each projected curtain is intersected at

most once by $\overline{p}$.) Let $P(H_i) = \bigcap\{h^+ | h \in H_i\}$ be the convex polytope determined by the hyperplanes in H_i and define $Compl(P(H_i)) = E^5 - P(H_i)$ to be the complement of $P(H_i)$. We want to preprocess $H = H_1 \cup \cdots \cup H_m$ such that we can efficiently find the smallest i^* such that $Compl(P(H_{i^*}))$ contains a query point q.

Before we describe the solution in all its technical details, let us give an overview of the method. First, we note that the subdivision of 5–space induced by the $m = O(n^{2\epsilon})$ polytopes $P(H_i)$ has size $O(n^2 m) = O(n^{2+2\epsilon})$ (see claim **(B)** below), and that a point location in this subdivision tells us exactly which polytopes do and do not contain a query point. However, there are two problems in preprocessing this subdivision for point location queries using Clarkson's method [11]. The first problem is that if we take a sample of the hyperplanes of size r, then the subdivision defined by these hyperplanes has size $O(r^2 m)$. If we take r to be a constant, then the $O(m)$ factor dominates the size of the subdivision, resulting in a structure that uses too much space. This problem is overcome by taking large samples of size $r = n^{1/10}$ so that we can afford the extra $O(m)$ factor in the complexity. But this imposes a new problem, namely that we can no longer locate the query point in the subdivision in a brute-force way. So we need another structure for locating the query point in the subdivision. A second, and more serious, problem that we encounter is the following. The subdivisions that we consider are not full arrangements of hyperplanes. Therefore, random sampling theory does not guarantee anything about the number of hyperplanes that cut a simplex in the triangulated subdivision. Indeed, since we have only a small number of simplices (much less than r^5) there is no way in which we can bound this number in a satisfactory way. Moreover, the regions of the subdivision are not convex, so how should we triangulate them? The fact that saves us is that only the hyperplanes from polytopes $P(H_1), \ldots, P(H_{i-1})$ are important, when we consider a region (in the subdivision induced by some sample) that is already outside $P(H_i)$.

It is time to make these ideas concrete. Let $R \subset H$ be a random sample of H of size $r = n^{1/10}$, and let $R_i = R \cap H_i$. Define $A_i = Compl(P(R_i)) \cap P(R_{i-1}) \cap \cdots \cap P(R_1)$. If $R_i = \emptyset$ then we define $P(R_i) = E^5$; hence, $A_i = \emptyset$ in that case. Thus, for $R = H$, A_i is exactly the region where the answer to a query is i. Finally, define $A_{m+1} = P(R_m) \cap \cdots \cap P(R_1)$. See Figure 3 for an illustration of these definitions. For each non-empty A_i, $i > 1$, we construct a set $Sim(A_i)$ of simplices as follows. Pick an arbitrary point $p \in P(R_i) \cap \cdots \cap P(R_1)$; if $P(R_i) \cap \cdots \cap P(R_1) = \emptyset$, then $A_i = P(R_{i-1}) \cap \cdots \cap P(R_1)$ and we pick a point in A_i. We triangulate the facets of A_i that are not facets of $P(R_i) \cap \cdots \cap P(R_1)$, and extend the 4-simplices thus obtained to 5-simplices, using point p. We claim that:

(A) $A_i \subseteq \bigcup\{s | s \in Sim(A_i)\}$ for every $1 < i \leqslant m+1$

(B) $\sum_{i=1}^{m+1} |Sim(A_i)|$, the total number of simplices, is $O(r^2 m)$

(C) Each simplex in $Sim(A_i)$ is intersected by $O(\frac{n}{r} \log^2 r)$ hyperplanes in $H_1 \cup \cdots \cup H_{i-1}$, with probability greater than $\frac{1}{2}$

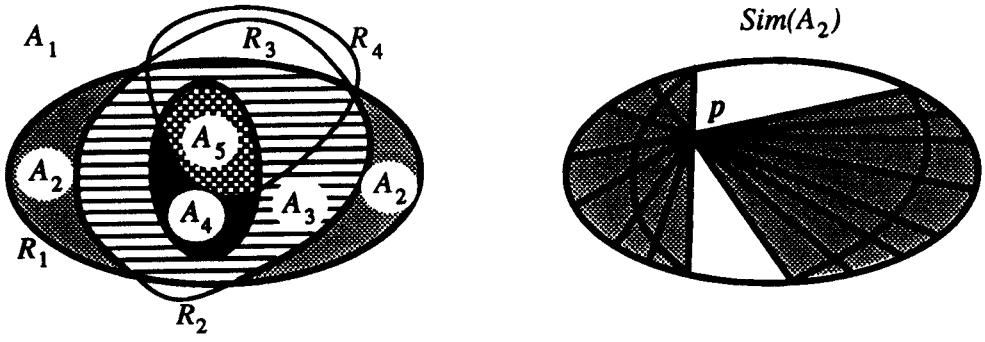

Figure 3: The regions A_i and the set $Sim(A_2)$ of triangles constructed for A_2.

We postpone the proof of **(A)**–**(C)** and continue to describe the data structure. Let F be the set of $O(r^2m)$ hyperplanes containing the facets of simplices in $Sim(R) = \bigcup_{i=1}^{m+1} Sim(A_i)$. The data structure is a tree of branching degree $O(r^2m)$, where each child of the root corresponds to a simplex in $Sim(R)$. At the root we store a point location structure for the arrangement $\mathcal{A}(F)$. With the child corresponding to simplex $s \in Sim(A_i)$ we store the value $i_s = \min(i, \{j : s \subset h^- \text{ for some } h \in H_j\})$. This is the smallest index j for which we can certify that $s \subset Compl(P(H_j))$. The child corresponding to s is also the root of a recursively defined structure on the hyperplanes in $H_1 \cup \cdots \cup H_{i_s-1}$ that intersect s. Recall that $r = n^{1/10}$, and that the point location structure for $\mathcal{A}(F)$ uses $O((r^2m)^{5+\delta})$ preprocessing, for an arbitrarily small $\delta > 0$. The term $r^{10+2\delta}$ forces us to have $r = (n')^{1/10}$, where n' is the current number of hyperplanes at some point in the recursion. This, however, would cause the depth of the tree to be $O(\log \log n)$. To avoid this, we stop the recursion when $n' = m$, and we solve the problem 'brute force' using $O(m^{5+\delta})$ preprocessing.

We have finished the description of the data structure, so now we can describe the query algorithm. Let q be the query point. We want to find the smallest index i^* such that $q \in Compl(P(H_i))$. Initialize $i^* = \infty$. First, we locate q in the arrangement $\mathcal{A}(F)$. The cell of $\mathcal{A}(F)$ that contains q uniquely determines the region A_i that contains q, and also the simplex $s \in Sim(A_i)$ that contains q. Recursively find the smallest index i^* such that $q \in Compl(P(H_{i^*}))$ in the substructure rooted at the child corresponding to s. Finally, set $i^* = \min(i^*, i_s)$.

Lemma 3 *The smallest index i^* such that $Compl(P(H_{i^*}))$ contains a query point can be found in $O(\log n)$ time with a structure that uses $O(n^{2+\varepsilon}m^{6+c\log(1/\varepsilon)}) = O(n^{2+7\varepsilon+c\varepsilon\log(1/\varepsilon)})$ space and (expected) preprocessing time, where c is a constant.*

Proof: The correctness of the method should be clear from the preceding discussion (we will prove claims **(A)**–**(C)** below), so let us concentrate on its complexity. The

point location structure for $\mathcal{A}(F)$ stored at the root of the tree uses $O((r^2m)^{5+\delta})$ preprocessing time and space, for any fixed $\delta > 0$, and it has a query time of $O(\log n)$. It is not hard to test a sample R for the condition in **(C)** in time $O((r^2m)^{5+\delta}n)$. Since the probability of success is greater than $\frac{1}{2}$ we expect to find a good sample after a constant number of trials. In the same time we can compute the values i_s, and find the hyperplanes on which to recurse for each simplex s. Using claims **(B)** and **(C)**, we see that the space and (expected) preprocessing time $S(n)$ used by our structure satisfies:

$$\begin{aligned} S(n) &= O((r^2m)^{5+\delta}n) + O(r^2m)S(\tfrac{n}{r}\log^2 r) \\ S(m) &= O(m^{5+\delta}) \end{aligned}$$

Furthermore, the query time $Q(n)$ satisfies:

$$\begin{aligned} Q(n) &= O(\log n) + Q(\tfrac{n}{r}\log^2 r) \\ Q(m) &= O(\log n) \end{aligned}$$

where $r = n^{1/10}$. It is not hard to verify that the depth of the tree is $O(\log(1/\varepsilon))$ and, hence, that the query time is $O(\log n)$. The proof of the preprocessing is slightly more involved. Let $S_d(n')$ be the space used by a subtree of depth d, with n' being the number of hyperplanes stored in the subtree. One can prove that $S_d(n') = O(m^{5+\delta+d}(n')^{2+\epsilon})$; for $d = 0$ we have $n' = m$ so the claim is true, and for $d > 1$ this follows by induction on d. Since the depth of the whole tree is $O(\log(1/\varepsilon))$, we have $S(n) = S_{c\log(1/\epsilon)}(n)$, for some constant c, and the bound follows.

It remains to prove claims **(A)**–**(C)**.

Claim (A): $A_i \subseteq \bigcup\{s | s \in Sim(A_i)\}$ for every $1 < i \leqslant m+1$.

Proof: Assume the point p that we picked to construct the simplices is inside $P(R_i) \cap \cdots \cap P(R_1)$; the case where $P(R_i) \cap \cdots \cap P(R_1) = \varnothing$ is proved in a similar way. Let x be an arbitrary point in A_i. Shoot a ray from point p in the direction of x. After the ray passes through x it will hit a facet of A_i. Since $p \in P(R_i) \cap \cdots \cap P(R_1)$, and $A_i \subseteq \mathit{Compl}(P(R_i))$, this cannot be a facet of $P(R_i)$. Thus p is contained in the 5-simplex that we constructed out of the 4-simplex that we hit in that facet. □

Claim (B): $\sum_{i=1}^{m+1} |Sim(A_i)|$, the total number of simplices, is $O(r^2m)$.

Proof: Note that every feature of A_i is either a feature of $P(R_{i-1}) \cap \cdots \cap P(R_1)$, or a feature of $P(R_i)$, or a feature of $P(R_i) \cap \cdots \cap P(R_1)$. The complexity of $P(R_{i-1}) \cap \cdots \cap P(R_1)$, $P(R_i)$, and $P(R_i) \cap \cdots \cap P(R_1)$ are $O(r^2)$ by the Upper Bound Theorem [16]. Hence, the complexity of A_i is $O(r^2)$, and since the number of A_i's is at most m, the bound follows. □

Claim (C): Each simplex in $Sim(A_i)$ is intersected by $O(\frac{n}{r}\log^2 r)$ hyperplanes in $H_1 \cup \cdots \cup H_{i-1}$, with probability greater than $\frac{1}{2}$.

Proof: We will use the following fact.

Fact 1: Let H be an ordered set of n items, and $R \subset H$ be a random subset of size r. The probability that there is a 'gap' of size greater than αn between two consecutive items in R (or before the first or after the last item) is $O(r^2(1-\alpha)^{r-2})$.

(The size of a gap between two items in R is the number of items in H that lie between the two items.) We also need the more-dimensional version of this fact, proved by Clarkson [11].

Fact 2: Let H be a set of n hyperplanes in E^d, and $R \subset H$ be a random subset of size r. The probability that there is a simplex in the triangulated arrangement $\mathcal{A}(R)$ that is intersected by more than αn hyperplanes in H is $O(r^{d(d+1)}(1-\alpha)^{r-d(d+1)})$.

In the following we shall use that for suitable $\alpha = O(\log r/r)$ these probabilities are at most $1/r^2$.

Consider the ordered set $H = H_1 \cup \cdots \cup H_m$ of hyperplanes in E^5; the hyperplanes within a set H_i are ordered arbitrarily, but a hyperplane from H_i comes before any hyperplane from H_j if $i < j$. There are n^r samples $R \subset H$ of size r. (The sampling is done with replacement, as in [11].) We will discard samples that do not satisfy condition **(C)** until we are left with a collection of samples that all satisfy **(C)**. This collection will contain at least $(1 - \frac{r+1}{r^2})n^r$ samples, which proves **(C)**.

Let $A_{j_1}, \ldots, A_{j_k}$ be the non-empty regions, with $1 < j_1 < \cdots < j_k$. We will argue that there are at least $(1 - \frac{i+1}{r^2})n^r$ samples having the following property: each simplex in $Sim(A_{j_l})$, $k - i < l \leqslant k$, is intersected by $O(\frac{n}{r}\log^2 r)$ hyperplanes from $H_1 \cup \cdots \cup H_{j_l - 1}$. Since $k \leqslant r$, **(C)** then follows by setting $i = k$. (The claim is trivially true for A_1, and for the empty regions.) The argument is by induction on i.

Since we want to prove something not only for the full set H, but also for subsets of the form $H_1 \cup \cdots \cup H_{j_l - 1}$, we want our samples to contain a sufficient number of hyperplanes from each such subset. To this end we discard all samples R such that there is a 'gap' of size αn between two consecutive hyperplanes in R (or before the first hyperplane in R). From Fact 1 it follows that for suitable $\alpha = O(\log r/r)$, we discard at most n^r/r^2 samples. Let $H = \{h_1, \ldots, h_n\}$. The bound on the gap size implies that $R' = R \cap H'$ has size $r' \geqslant n'/\alpha n$, for any initial portion $H' = \{h_1, \ldots, h_{n'}\}$ of H. In the remainder we only consider samples with this property, of which there are at least $(1 - \frac{1}{r^2})n^r$. Now we are ready to prove that each simplex in $Sim(A_{j_l})$, $k - i < l \leqslant k$, is intersected by $O(\frac{n}{r}\log^2 r)$ hyperplanes from $H_1 \cup \cdots \cup H_{j_l - 1}$.

The base case, $i = 0$, is trivially true. So assume that the statement is true for $i - 1$, and consider $Sim(A_{j_i})$. By induction we still have at least $(1 - \frac{i}{r^2})n^r$ samples available. Let $H' = H_1 \cup \cdots \cup H_{j_i - 1}$ and $n' = |H'|$. Observe that if $n' = O(\frac{n}{r}\log^2 r)$

then trivially all the samples remain good. We know that we have chosen $r' \geqslant n'/\alpha n$ hyperplanes from H'. Hence, the number X of combinations that we can make using the hyperplanes in $R \cap (H - H')$ is no more than $(n-n')^{r-r'} < n^{r-r'}$. This implies that we must have Y different subsamples $R' \subset H'$ available, where

$$Y \geqslant (1 - \frac{i}{r^2})n^r/X \;>\; (1 - \frac{i}{r^2})n^r/n^{r-r'} \;=\; (1 - \frac{i}{r^2})n^{r'}.$$

Of these subsamples, at most $\frac{1}{(r')^2}(n')^{r'}$ are bad, in the sense that there is a simplex in $Sim(A_i)$ that is intersected by more than

$$O(\frac{\log r'}{r'}n') \;=\; O(\frac{\log r'}{n'/\alpha n}n') \;=\; O(\alpha n \log r') \;=\; O(\frac{n}{r}\log^2 r)$$

hyperplanes from H'. In other words, the number of samples that satisfy the conditions is at least

$$X(Y - \frac{1}{(r')^2}(n')^{r'}) \;\geqslant\; (1 - \frac{i}{r^2})n^r - (n-n')^{r-r'}\frac{1}{(r')^2}(n')^{r'}.$$

We have to show this is at least $(1 - \frac{i+1}{r^2})n^r$ or, equivalently, that

$$\frac{n^r}{r^2} - \frac{(n-n')^{r-r'}(n')^{r'}}{(r')^2} \;\geqslant\; 0.$$

We distinguish between the case where $n' \geqslant n/2$ and the case $n' < n/2$. In the first case we have

$$\frac{n^r}{r^2} - \frac{(n-n')^{r-r'}(n')^{r'}}{(r')^2} \;\geqslant\; \frac{n^r}{r^2} - \frac{(n/2)^{r-r'}n^{r'}}{(r')^2} \;=\; n^r(\frac{1}{r^2} - \frac{(1/2)^{r-r'}}{(r')^2}).$$

So we need $(r'/r)^2 \geqslant (1/2)^{r-r'}$, which is true since $r' \leqslant r$ and we may assume that $r \geqslant 3$. For the second case, $n' < n/2$, we can show in a similar way that we need $(r'/r)^2 \geqslant (1/2)^{r'}$, which is true if $r' \geqslant 2\log r$. For $r' < 2\log r$ we have $n' = O(\frac{n}{r}\log^2 r)$ and we already noted that the condition is also fulfilled in this case.

Thus we always have enough good samples left, proving **(C)**. □

This completes the proof of the Lemma 3. □

Note that $\varepsilon \log(1/\varepsilon) \to 0$ as $\varepsilon \to 0$. Hence, by choosing ε small enough, we can find the smallest index i^* such that $Compl(P(H_{i^*}))$ contains a query point in $O(\log n)$ time after $O(n^{2+\varepsilon'})$ preprocessing, for any $\varepsilon' > 0$.

Finally, we give an overview of the total structure. The main tree has branching degree $O(r^2)$, for $r = n^\varepsilon$. This tree imposes the ordering on the curtains. Each child of the root corresponds to a cell in a $(\frac{1}{r})$-cutting for the projected curtains. With the root we associate a structure to find the first cell c_1 intersected by a query

ray, and also the suffix $c_2, \ldots, c_t$. For each of the $O(r^6)$ possible suffices we have an associated structure. This associated structure consists of two levels. The first level selects in a constant number of groups the curtains whose projections are intersected so that we can extend their top edges to full lines; the structure of the second level is a structure as described above for finding the first (Plücker) polytope that does not contain a query (Plücker) point.

The analysis of the complexity of the associated structures that we store for each suffix is similar to the analysis of the structure for line stabbing queries in Subsection 2.2. Thus an associated structure has $O(\log n)$ query time and the preprocessing can be done in time $O(n^{2+\epsilon''})$, for any $\epsilon'' > 0$.

Since $r = n^{\epsilon}$, the depth of the main tree is constant, and the total query time is $O(\log n)$. The preprocessing $S(n)$ satisfies:

$$S(n) = O(r^6) \cdot O(n^{2+\epsilon''}) + O(r^2) S(\frac{n}{r})$$

with $r = n^{\epsilon}$. By choosing ϵ and ϵ'' sufficiently small we obtain a preprocessing time of $O(n^{2+\epsilon'''})$ for any fixed $\epsilon''' > 0$.

Observe that reducing the query time for curtains immediately reduces the query time for fat horizontal triangles and for axis-parallel polyhedra.

Theorem 8 *Ray shooting queries in a set of n curtains (or in a set of n fat horizontal triangles or in a set of axis-parallel polyhedra with n edges in total) can be performed in time $O(\log n)$ with a structure that uses $O(n^{2+\epsilon})$ preprocessing time and space, for any fixed $\epsilon > 0$.*

Remark: For axis-parallel polyhedra there is an easier way to obtain $O(\log n)$ query time. A query ray intersects a rectangle with edges parallel to the y- and z-axis if and only if the ray intersects the rectangle in the projection onto the xy-plane and in the projection onto the xz-plane. Thus, one doesn't have to use Plücker coordinates to test this, leading to a simpler solution. This is the approach taken in [24].

3 Hidden surface removal

In this section we present an output-sensitive hidden surface removal algorithm for a set of non-intersecting triangles in space. (The restriction to triangles is just to simplify the description. In fact, any set of non-intersecting polyhedra can be handled.) Unlike previous output-sensitive algorithms this algorithm makes no assumptions about the triangles, except that they do not intersect. In particular we do not require a depth order on the triangles. Since a depth order often does not exist (and even when an order does exist it is, in general, hard to compute), this is an important feature of our algorithm.

Let S be a set of non-intersecting triangles with n edges in total. To simplify the description, we assume that the view point is located at $z = \infty$. We want to

compute the *visibility map* of S. The visibility map $\mathcal{M}(S)$ of S is the subdivision of the viewing plane (in our case the xy-plane) into maximally connected regions such that in each region one triangle is visible or no triangle at all. Notice that the vertices of this subdivision are of two types: they are either the projection of a visible vertex of a triangle or they are the intersection of the projection of two edges. The global method we use to compute $\mathcal{M}(S)$ is similar to the method used in [13, 22]: starting from the visible vertices we 'shoot along' the edges of $\mathcal{M}(S)$, thereby discovering the other vertices of $\mathcal{M}(S)$. There is one important difference, however; in [13, 22] the visible vertices are computed beforehand, whereas we compute them 'on the fly'. Next we describe the global algorithm in a little more detail.

The algorithm moves a horizontal sweep line from top to bottom over the viewing plane, halting at every vertex of $\mathcal{M}(S)$ and at every projection of a vertex of a triangle. Thus we have an event queue Q which initially stores the projections of vertices of triangle in order of decreasing y-coordinate; when a new vertex of $\mathcal{M}(S)$ is discovered it is inserted into Q. While sweeping, the algorithm keeps track of the edges of $\mathcal{M}(S)$ that are intersected by the sweep line. These edges are stored in a binary search tree T in the order of their intersection with the sweep line. We also store for each edge the face that is visible to its left.

The sweep line is advanced in the following way. First the vertex v with greatest y-coordinate is removed from Q. If v is the projection of a vertex of a triangle we test if it is visible. This is done by searching in T to find the edge of $\mathcal{M}(S)$ to the right of v. Since we know the face that is visible to the left of this edge, we can check whether v is visible or not. If v is not the projection of a vertex of a triangle, we already know that it is visible. If v is visible we now have to update Q and T. This means that we have to compute the other vertices of the edges of the visibility map that are incident to v and will be intersected by the sweep line when it is advanced. It is easy to ensure that we always know which edges of triangles correspond to the edges incident to v. Let e be such an edge. We use the same characterization to find the other vertex w of e that was used in [13]. Let f be the face that is immediately below v, i.e., the face whose interior is hit first when we shoot a ray from the view point into the direction of v. Let ρ be the projection onto f of the ray starting at v along e. Then we have:

Lemma 4 [13] *The vertex w is either the projection of a vertex of e or it is the intersection of the projection of e with the projection of the first edge passing above ρ.*

To compute the other vertex of e we thus have to be able to find the first edge passing above a query ray. But this edge corresponds exactly to the first curtain hit by the ray, when we hang a curtain from each edge of a triangle. Hence, a data structure for this problem was already presented in Section 2.2.

However, two problems remain. First of all, we cannot afford to spend the $O(n^{2+\epsilon})$ preprocessing time that the structure for ray shooting in a set of curtains takes. Second, we have ignored the computation of ρ itself. For this we need to

find the face immediately below v. Note that it is not always possible to find this face using T. Thus we need a structure that computes the face immediately below a visible query point p.

There is a trivial solution to the second problem: just compute the visibility map of S (which can be done in time $O(n^2)$, see [20]) and perform a point location in this map in $O(\log n)$ time. But this structure suffers from the same drawback as the ray shooting structure: we do not want to spend that much preprocessing. (And there seems to be something wrong with the idea of computing $\mathcal{M}(S)$ to be able to compute $\mathcal{M}(S)$ anyway.) So what we would like to have is a trade-off between preprocessing time and query time for our data structures. This can be accomplished by partitioning the set of triangles into m subsets each of size $\frac{n}{m}$ (for some $1 < m < n$ to be determined later). Now the preprocessing time of the ray shooting structure is $O(n^{2+\epsilon}/m)$ and the time to compute all visibility 'submaps' is $O(n^2/m)$. A query is performed by querying all subsets and taking the first of the m answers that are found. The query times thus are $O(m \log^2 n)$ and $O(m \log n)$ respectively. (For the second structure a better trade-off is in fact possible. But since this would not help us if we do not have better trade-offs for the curtains, we stick to the simpler method described above.)

The total running time of the hidden surface removal algorithm can thus be analyzed as follows: $O(n \log n)$ (to sort the vertices of the triangles and insert them into Q), plus $O(n^{2+\epsilon}/m)$ (the preprocessing time for the data structures), plus $O(n \log n)$ (to check for each triangle vertex if it is visible), plus $O(m \log^2 n)$ for each edge of $\mathcal{M}(S)$ that is discovered. Hence, the time needed to compute $\mathcal{M}(S)$ is $O(n^{2+\epsilon}/m^{1+\epsilon} + km \log^2 n)$, where k denotes the complexity of $\mathcal{M}(S)$. To minimize the time we would like to choose m depending on k. More precisely, we would like to set $m = n^{1+\epsilon}/\sqrt{k}$ to obtain a running time of $O(n^{1+\epsilon}\sqrt{k})$. Although we do not know the value of k in advance, it is still possible to achieve this running time by 'guessing' the value of k as in [22]: set $m = n^{1+\epsilon}/\sqrt{k'}$ for some constant value of k' and start running the algorithm. If $k \leqslant k'$ then the algorithm finishes within $O(n^{1+\epsilon}\sqrt{k'})$ time and we are done. Otherwise, we stop the algorithm as soon as we discover that $k > k'$, and try it again, multiplying k' by four. This way the total running time will be at most a constant factor worse than the time taken if we had plugged in the right value of k right away. We thus obtain:

Theorem 9 *The visibility map of a set of non-intersecting polyhedra with n edges in total can be computed in time $O(n^{1+\epsilon}\sqrt{k})$, where k is the complexity of the map, for any fixed $\epsilon > 0$.*

4 Concluding remarks

In this paper the ray shooting problem has been studied for three special classes of polyhedral objects: axis-parallel polyhedra, curtains and fat horizontal polyhedra.

We presented structures that use $O(n^{2+\varepsilon})$ preprocessing (for any fixed $\varepsilon > 0$) and have a query time of $O(\log n)$. For axis-parallel polyhedra it is also possible to achieve a query time of $O(n^{1+\varepsilon}/\sqrt{m})$ with $O(m^{1+\varepsilon})$ preprocessing, for any $n < m < n^2$. For the curtains and fat horizontal triangles we are only able to get the naive trade-off that uses $O(n^{2+\varepsilon}/m)$ preprocessing to achieve $O(m \log n)$ query time. The general problem of arbitrary (possibly intersecting) triangles has been studied as well. Here a structure was given whose preprocessing time is $O(n^{4+\varepsilon})$ and with a query time of $O(\log n)$. These results improve or generalize the previously best known solutions.

Furthermore the hidden surface removal problem was studied. The first output-sensitive algorithm was presented that can deal with (non-intersecting) polyhedra for which a depth order on the faces is not known. Its running time is $O(n^{1+\varepsilon}\sqrt{k})$, where n is the total number of edges of the polyhedra and k is the size of the output.

A number of open problems remain. For example, we would like to have a better preprocessing vs. query time trade-off in the structure for curtains than the one given in Section 3. This would also improve the running time of the hidden surface removal algorithm. Furthermore, if we could count the number of curtains intersected by a query line efficiently this would lead to improved bounds for the general ray shooting problem. Finally it would be interesting to have lower bounds for the ray shooting problem: what is the amount of preprocessing that is necessary to solve the general ray shooting problem with a polylogarithmic query time?

References

[1] 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.

[2] P.K. Agarwal and M. Sharir, Applications of a New Space Partitioning Technique, *Proc. Workshop on Algorithms and Data Structures*, 1991, to appear.

[3] P.K. Agarwal, M. van Kreveld and M. Overmars, Intersection Queries for Curved Objects, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 41–50.

[4] B. Aronov and M. Sharir, On the Zone of a Surface in a Hyperplane Arrangement, *Proc. Workshop on Algorithms and Data Structures*, 1991, to appear.

[5] B. Chazelle, H. Edelsbrunner, M. Gringi, L.J. Guibas, M. Sharir and J. Snoeyink, Ray Shooting in Polygons Using Geodesic Triangulations, *Proc. 18th Int. Coll. on Automata, Languages and Programming*, 1991, to appear.

[6] B. Chazelle, H. Edelsbrunner, L.J. Guibas, R. Pollack, R. Seidel, M. Sharir and J. Snoeyink, Counting and Cutting Cycles of Lines and Rods in Space, *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 242–251.

[7] B. Chazelle, H. Edelsbrunner, L.J. Guibas and M. Sharir, Lines in Space — Combinatorics, Algorithms and Applications, *Proc. 21st ACM Symp. on Theory of Computing*, 1989, pp. 382–393.

[8] B. Chazelle and L.J. Guibas, Visibility and Intersection Problems in Plane Geometry, *Discr. & Comp. Geometry* **4** (1989), pp. 551–581.

[9] B. Chazelle, M. Sharir and E. Welzl, Quasi-Optimal Upper Bounds for Simplex Range Searching and New Zone Theorems, *Proc. 6th ACM Symp. on Computational Geometry*, 1990, pp. 23–33.

[10] S.W. Cheng and R. Janardan, Space-Efficient Ray-Shooting and Intersection Searching: Algorithms, Dynamization and Applications, *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1991, pp. 7–16.

[11] K. Clarkson, New Applications of Random Sampling in Computational Geometry, *Discr. & Comp. Geometry* **2** (1987), pp. 195–222.

[12] R. Cole and M. Sharir, Visibility Problems for Polyhedral Terrains, *J. Symb. Comp.* **7** (1989), pp. 11–30.

[13] M. de Berg and M.H. Overmars, Hidden Surface Removal for Axis-Parallel Polyhedra, *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 252–261.

[14] D.P. Dobkin and H. Edelsbrunner, Space Searching for Intersecting Objects, *J. Algorithms* **8** (1987), pp. 348–361.

[15] D.P. Dobkin and D. Kirkpatrick, A Linear Algorithm for Determining the Separation of Convex Polyhedra, *J. Algorithms* **6** (1985), pp. 381–392.

[16] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[17] L. Guibas, M. Overmars and M. Sharir, *Ray Shooting, Implicit Point Location and Related Queries in Arrangements of Segments*, Tech. Rep. No. 443, Courant Institute of Math. Sciences, New York University, 1989.

[18] D.G. Kirkpatrick, Optimal Search in Planar Subdivisions, *SIAM J. Comput.* **12** (1983), pp. 28–35.

[19] J. Matoušek, Approximations and Optimal Geometric Divide-and-Conquer, *Proc. ACM Symp. on Theory of Computing*, 1991, to appear.

[20] M. McKenna, Worst-Case Optimal Hidden Surface Removal, *ACM Trans. Graphics* **6** (1987), pp. 19–28.

[21] M.H. Overmars, H. Schipper and M. Sharir, Storing Line Segments in Partition Trees, *BIT* **30** (1990), pp. 385–403.

[22] M.H. Overmars and M. Sharir, Output-Sensitive Hidden Surface Removal, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 598–603.

[23] M. Pellegrini, Stabbing and Ray Shooting in 3 Dimensional Space, *Proc. 6th ACM Symp. on Computational Geometry*, 1990, pp. 177–186.

[24] M. Pellegrini, New Results on Ray Shooting and Isotopy Classes of Lines in 3-Dimensional Space, *Proc. Workshop on Algorithms and Data Structures*, 1991, to appear.

[25] A. Schmitt, H. Müller and W. Leister, Ray Tracing Algorithms — Theory and Practice, in: R.A. Earnshaw (ed.), *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, Vol. F40, Springer-Verlag, 1988, pp. 997–1030.

[26] J. Stolfi, *Primitives for Computational Geometry*, Ph. D. Thesis, Computer Science Department, Stanford University, 1988.