Information Processing Letters 35 (1990) 137–143
North-Holland

20 July 1990

HIDDEN LINE ELIMINATION FOR ISOORIENTED RECTANGLES *

K. MEHLHORN, S. NÄHER and C. UHRIG

FB Informatik, Universität des Saarlandes, 6600 Saarbrücken, FRG

Communicated by T. Lengauer
Received 9 November 1989
Revised 26 February 1990

We consider a special case of the hidden line elimination problem. The scene consists of n isooriented rectangles in R^3 and an observer at $z = +\infty$. We give an algorithm that computes the visible parts of the edges on a random access machine in time $O(n \log n + k \log(n^2/k))$, where k is the number of line segments in the output, and uses $O(n \log n)$ space.

Keywords: Computational geometry, computer graphics, data structures, hidden line elimination, skyline problem

1. Introduction

We consider the following problem: Given n isooriented rectangles in R^3, i.e., rectangles which are parallel to the xy-plane and whose edges are parallel to the x- and y-axis, compute and report all parts of the edges, that are visible to an observer at $z = +\infty$ (see Fig. 1).

The hidden line elimination problem is of considerable interest. The running time of most hidden line elimination algorithms depends on the complexity of the projected scene, i.e., on the number of intersections between the edges projected into the xy-plane. Of course, in general, many of these intersections are invisible to the observer. The first output-sensitive algorithm for the hidden line elimination problem of isooriented rectangles was described by Güting and Ottmann [6]. They achieved running time $O((n + k)(\log n)^2)$, where k is the number of the visible parts of the edges. Preparata et al. [10] have improved the running time to $O(n(\log n)^2 + k \log n)$. They also mentioned, that one can reach running time $O((n + k)\log n \log\log n)$ using dynamic fractional cascading [8]. Bern [3] presented an algorithm with running time $O(n \log n \log \log n + k \log n)$. He recently [4] improved the running time of his algorithm to $O((n + k)\log n)$. Another $O((n + k)\log n)$ algorithm was given by Atallah, Goodrich and Overmars [1].

The algorithm described in this paper is almost identical to Bern's improved algorithm, although the analysis is slightly sharper at one point. It was found independently from his work. It runs on a random access machine in time $O(n \log n + k \log(n^2/k))$ and uses $O(n \log n)$ space. Note

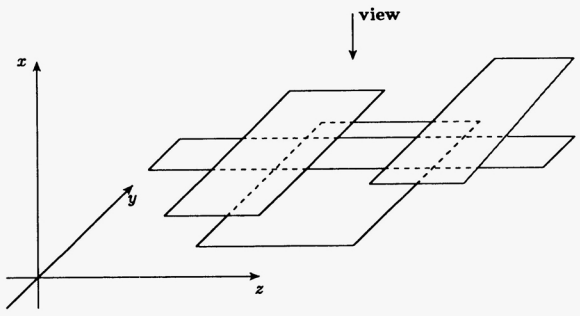

Fig. 1. A scene of rectangles.

* This research was supported by the ESPRIT II Basic Research Action Program, ESPRIT P. 3075-ALCOM, and by the DFG, grant SPP ME 620/6-1.
This paper was presented at the meeting of the DFG Schwerpunktprogramm in Paderborn, June 1989.

that for small and great k the running time is optimal. It was first presented at a DFG-colloquium in Paderborn in June 1989.

As many researchers before, we use plane sweep to solve the problem. First we sweep a plane parallel to the xz-plane along the y-axis and compute the visible parts of the edges, being parallel to the x-axis. Afterwards we sweep a plane parallel to the yz-plane along the x-axis to compute the visible parts being parallel to the y-axis. We maintain the intersection of the sweep-plane with the scene in a static segment tree. Whenever a rectangle starts (ends), an appropriate segment is inserted into (deleted from) the segment tree and its visible parts are computed.

Using standard results about segment trees, the approach outlined so far yields a running time of $O((n+k)(\log n)^2)$. We introduce two techniques to improve upon this.

(1) We augment the segment tree with additional information which allows us to identify visible parts in time $O(\log(n^2/k))$ per part. The additional information records for each node of the tree the highest and lowest visible part stored in its subtree. This type of augmented segment tree was already used in [3]; our running time analysis is slightly sharper than his.

(2) We show how to solve a two-dimensional hidden line problem for a set of q horizontal line segments in the xy-plane in linear time $O(q)$. This assumes that the endpoints of the line segments have distinct integer coordinates in $\{1,\ldots,2q\}$ and that the line segments are given in order of decreasing z-coordinate. We use this algorithm to solve the hidden line problem for the node lists of all nodes of the segment tree in total time $O(n \log n)$. This technique is also used in [4].

This paper is organized as follows. In Section 2 we define the augmented segment tree, give a sketch of the main program and present the procedure that computes the visible parts of an edge. In Section 3 we show the correctness and analyze the running time of this first version of the algorithm. Section 4 gives the modification of the data structure, explains the preprocessing phase and shows how the algorithm uses the modified data structure. Finally, Section 5 contains the analysis of the final algorithm.

2. The basic algorithm

As mentioned in Section 1 of the paper, we perform space sweep twice; first, a plane parallel to the xz-plane is swept from $y = -\infty$ to $y = +\infty$ to compute the visible parts of the edges parallel to the x-axis, which we call from now on *horizontal*. Later a similar space sweep along the x-axis computes the visible parts of the *vertical* edges, namely those being parallel to the y-axis. We only consider the first sweep.

For simplicity let all rectangles have pairwise disjoint coordinates and let all z-coordinates be positive.

At any position of the sweep plane, its intersection with the scene of rectangles is a set H of horizontal segments. Initially (at position $y = -\infty$) H is empty. Whenever the sweep plane reaches a rectangle $R = (x_0, x_1, y_0, y_1, z_0)$ (this happens at $y = y_0$), the segment $s = (x_0, x_1, z_0)$ is added to H and its visible parts are computed. It is present in H until the sweep plane reaches the position $y = y_1$. There again its visible parts are computed and it is removed from H. The y-coordinates of the rectangles are called the *transition points* of the algorithm. Thus we have to maintain a set H of horizontal line segments under the following operations:

(a) Insert a new segment s into H.

(b) Delete a segment s from H.

(c) Given a segment s, compute the parts, visible from $z = +\infty$.

Note, that the third operation is a two-dimensional visibility problem.

An appropriate and popular data structure for this kind of problem is a static segment tree. We use a special version of this data structure.

Let X be the set of all occurring x-coordinates of the given rectangles. Note, that X is known in advance, namely X is the set of the x-coordinates of all rectangles in S.

A segment tree for H is a leaf oriented balanced binary search tree T with $|X|$ internal nodes, which are labeled in symmetric order with the elements of X. With every node of T a half open interval, called *range*, is associated that is defined in the following way

$$\text{range(root)} = (-\infty, +\infty],$$

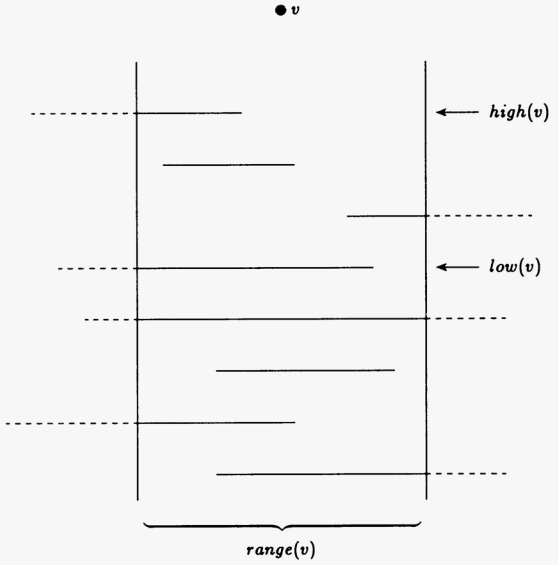

Fig. 2. The segments stored in $NL(v)$.

$$\text{range}(\text{leftchild}(v)) = \text{range}(v) \cap (-\infty, x],$$
$$\text{range}(\text{rightchild}(v)) = \text{range}(v) \cap (x, +\infty],$$

where x is the label of v.

Moreover, in every node v we maintain a set $NL(v)$, the node list of v, with

$$NL(v) = \{ s \in H \mid \text{range}(v) \subseteq s \text{ and } \text{range}(\text{parent}(v)) \not\subseteq s \}.$$

We use the notation "range$(v) \subseteq s$" for range$(v) \subseteq (x_0, x_1]$ where $s = (x_0, x_1, z_0)$. Similarly we will use "range$(v) \cap s$" for "range$(v) \cap (x_0, x_1]$".

The range of every node v covers an interval on the x-axis. The union of all nodes v with $s \in NL(v)$ corresponds to the projection of the segment s onto the x-axis. The ranges of these nodes are pairwise disjoint (see Fig. 2).

In each node we store two additional values, named low(v) and high(v), with

$$\text{low}(v) = \begin{cases} \max_z(v), & v \text{ is leaf,} \\ \max(\max_z(v), \min(\text{low}(u), \text{low}(w))), & \\ \qquad v \text{ has children } u \text{ and } w, & \end{cases}$$

and

$$\text{high}(v) = \begin{cases} \max_z(v), & v \text{ is leaf,} \\ \max(\max_z(v), \text{high}(u), \text{high}(w)), & \\ \qquad v \text{ has children } u \text{ and } w, & \end{cases}$$

where $\max_z(v) = \max\{ z \mid s \in NL(v),\ s = (x_0, x_1, z)\}$.

The low value (high value) in a node v gives us the z-coordinate of the lowest (highest) visible segment in the scene consisting of all segments stored in the subtree rooted at v. W.l.o.g. we assume, that all z-coordinates are positive; then at the beginning of the sweep, all node lists are empty and the high and low values are initialized with 0.

Let us assume for the moment (in Section 4 we will show how to do better), that the node lists $NL(v)$ are organized as balanced binary trees, ordered according to the z-coordinate of the segments. Then the low and high values can be computed in constant time per node.

The following characteristics of the segment tree are well known (cf. [2,7,etc.]).

(S1) A segment s is stored in at most O(log n) node lists.

(S2) All parents of nodes v with $s \in NL(v)$ lie on only two paths of the tree.

Now we sketch the sweep algorithm. Let Y be the set of the y-coordinates of all rectangles in the scene.

```
 0 Main Program
 1 build and initialize the segment tree
 2 for all y ∈ Y in increasing order do
 3    if y = y0 for some rectangle R = (x0, x1, y0,
        y1, z0)
 4    then
 5       COMPUTE_VISIBLE_PARTS(s, root)
            /* s = (x0, x1, z0)
 6       INSERT(s, root)
 7       UPDATE_HIGH_AND_LOW(s, root)
 8    else
 9       DELETE(s, root)
10       UPDATE_HIGH_AND_LOW(s, root)
11       COMPUTE_VISIBLE_PARTS(s, root)
12    fi
13 od
```

The operations "INSERT(s, root)" and "DELETE(s, root)" are standard operations on segment trees. We need not consider them in detail.

The same holds for "UPDATE_HIGH_AND_LOW(s, v)" for a node v. This procedure

updates all high and low values that have to change after an insertion or deletion. This is done exactly according to the definitions of low and high.

So we now concentrate on the procedure "COMPUTE_VISIBLE_PARTS". It takes two arguments s and v, where $s = (x_0, x_1, z_0)$ is a part of a segment and v is a node with $(x_0, x_1] \subseteq \text{range}(v)$ and computes all visible parts of s. Let $z(s)$ denote the z-coordinate of a segment s.

```
0  Procedure
     COMPUTE_VISIBLE_PARTS(s, v)
1  if z(s) > high(v)
2  then
3    "report s as visible"
4  else
5    if z(s) > low(v)
6    then
7      for all children u of v with range(u)
         ∩ s ≠ ∅
8      do
9        COMPUTE_VISIBLE_
           PARTS(range(u) ∩ s, u)
10     od
11   fi
12 fi
```

3. Correctness and time bounds

Lemma 1 (correctness). *A call of* "COMPUTE_VISIBLE_PARTS(s, root)" *for a segment* $s \notin L$ *computes and reports exactly the visible parts of* s.

Proof. Let V be the set of nodes v, where no further call of the procedure is performed, i.e., where the recursion stops. It follows, that

$$\bigcup_{v \in V} (\text{range}(v) \cap s) = (x_0, x_1]$$

where $s = (x_0, x_1, z)$.

In such a node $v \in V$ either $\text{range}(v) \cap s$ is reported or nothing is done.

Claim 1. *If* $\text{range}(v) \cap s$ *is reported, then* $\text{range}(v) \cap s$ *is totally visible.*

Proof. Let $\text{range}(v) \cap s$ be reported. Then $z(s) > \text{low}(w)$ for all ancestors w of v, since the condition in line 5 must be true for all ancestors of v, and hence $z(s) > \max_z(w)$ for all ancestors w of v, since $\text{low}(w) \geq \max_z(w)$ for all nodes w according to the definition of low.

That means, $\text{range}(v) \cap s$ is not covered by a segment stored in an ancestor node of v and, since $z(s) > \text{high}(v)$, there is no segment in the subtree rooted at v with higher z-coordinate than s.

Claim 2. *If* $\text{range}(v) \cap s$ *is not reported, it is not visible.*

Proof. If $\text{range}(v) \cap s$ is not reported, then

$$z(s) < \text{low}(v).$$

This implies immediately, that $\text{range}(v) \cap s$ is totally covered by segments stored in the subtree rooted at v.

These two claims prove the lemma. □

Lemma 2. *Computing the visible parts of the segment* s *takes time* $O(\log n + k_s \log(n/k_s))$, *where* k_s *is the number of visible parts of* s.

Proof (see Fig. 3). Whenever the procedure "COMPUTE_VISIBLE_PARTS(s, v)" visits a node and performs a recursive call, we have $\text{low}(w) < z(s) < \text{high}(w)$ for all ancestors w of v.

Also $\text{range}(v) \cap s \neq \emptyset$ for all such nodes v. If $\text{range}(v) \subseteq s$, then $\text{low}(w) < z(s) < \text{high}(w)$ for all ancestors w of v implies that $\text{range}(v) \cap s$ is partially but not totally visible. If $\text{range}(v) \subsetneq s$, then v lies on the search path to one of the endpoints of segment s. Thus all parents of visited nodes lie on

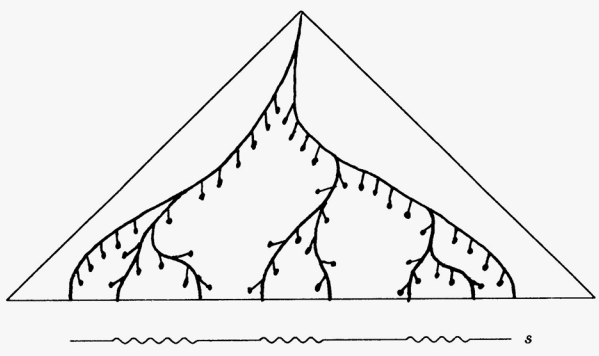

Fig. 3. Nodes visited by COMPUTE_VISIBLE_PARTS(s, root).

the paths to at most $2 + k_s$ leaves of the segment tree. Let us call the subtree induced by the set of nodes, visited by a call "COMPUTE_VISIBLE_PARTS(s, v)" the *visited tree* T_v. Let $|T_v|$ be the number of nodes in T_v; then

$$\begin{aligned}
&\max\{|T_v| \mid T_v \text{ is a visited tree with } 2 + k_s \text{ leaves}\} \\
&\leqslant \sum_{i=1}^{\log n} \min\{2^i, 2 + k_s\} \qquad \text{(i)} \\
&= (2 + k_s) \log n - (2 + k_s) \log(2 + k_s) \\
&\quad + (2 + k_s) \qquad \text{(ii)} \\
&= O\left((2 + k_s) \log \frac{n}{(2 + k_s)}\right) \\
&= O\left(\log n + k_s \log \frac{n}{k_s}\right)
\end{aligned}$$

(i) This follows from the fact that $|T_v|$ is maximal, if the number of nodes in every of the $\log n$ levels is maximal. But this number cannot exceed $2 + k_s$ or 2^i, if i is the depth of the level.

(ii) For the first $\log(2 + k_s)$ levels, 2^i gives the bound. These levels build a balanced binary tree with $O(k_s)$ nodes. The rest of the tree has obviously $O(k_s \log(n/k_s))$ nodes. □

Lemma 3. (a) *If the node lists are maintained as balanced binary trees, the total time spent for calls of procedures* "INSERT(s, v)" *and* DELETE(s, v)" *is* $O(n(\log n)^2)$.

(b) *The total cost of calls of* "UPDATE_HIGH_AND_LOW(s, v)" *is* $O(n \log n)$.

(c) *The total cost of calls of* "COMPUTE_VISIBLE_PARTS(v, s)" *is* $O(n \log n + k \log(n^2/k))$, *where k is the sum of visibility-to-invisibility changes of all segments.*

Proof. (a) The characteristics (S1) and (S2) of a segment tree allow a single operation to be done in time $O(\log n \cdot g(v))$, where $g(v)$ is the time needed to insert (delete) a segment into (from) the node list of the given node v.

(b) Follows immediately from (S1), (S2) and the definition of high(v) and low(v).

(c) The total time needed is

$$O\left(n \log n + \sum_{i=1}^{n} k_{s_i} \log \frac{n}{k_{s_i}}\right).$$

This sum is maximal, if $k_{s_i} = k/n$ for all i. This proves part (c). □

Since each segment is stored in $O(\log n)$ node lists and in each node we only spend a constant amount of space, the algorithm needs $O(n \log n)$ space.

4. The final algorithm

In the previous section we showed how to solve the hidden line elimination problem for isooriented rectangles in time $O(n(\log n)^2 + k \log(n^2/k))$.

The first term results from the computation of the current maximal z-coordinates when inserting (deleting) the rectangle edges into (from) the segment tree (for each insertion or deletion there have to be performed $O(\log n)$ operations on balanced trees).

In this section we show, how the fact that all rectangles (and thus all segments) are known in advance, can be exploited to precompute for every node v in the segment tree the sequence $M(v)$ of maximal z-coordinates corresponding to the sequence of operations executed on the node list $NL(v)$ in time $O(n \log n)$.

More precisely the ith element of $M(v)$ gives the maximal z-coordinate valid for the ith operation on $NL(v)$.

For each rectangle edge s ever stored in a node list $NL(v)$, there is exactly one position y_1 of the sweep line when s enters $NL(v)$ and one position y_2 when s leaves $NL(v)$. Let n_v be the number of segments ever stored in $NL(v)$. Then the node list $NL(v)$ is changed $2n_v$ times. If we number the operations on node list $NL(v)$ from 1 to $2n_v$, then we can define the life-span of a segment in $NL(v)$ as the interval $[t_1, t_2]$, where s is inserted into $NL(v)$ at t_1 and deleted from $NL(v)$ at t_2. Suppose now that we have to determine the maximal z-coordinate of a segment stored in $NL(v)$ at t. This is tantamount to computing the maximal z-coordinate of segment s with $t \in$ life-span(s).

We next show how to precompute the answers to all queries on $NL(v)$ in linear time $O(n_v)$.

More precisely, to compute the sequence $M(v)$ of maximal z-coordinates for a node list $NL(v)$

we have to solve a two-dimensional visibility problem of the following kind.

Input. A sequence S of n_v line segments in the two-dimensional tz-plane with t-coordinates from $T = \{1, \ldots, 2n_v\}$. S is ordered according to decreasing z-coordinate.

Output. An array of segments $M[1..2n_v]$, where for $t = 1, \ldots, 2n_v$, $M[t]$ is the segment $s \in S$ with maximal z-coordinate among all segments $s' \in S$, $s' = (t_1, t_2, z)$ and $t_1 \leqslant t \leqslant t_2$.

Suppose now that the array $M(v)$ is available for all nodes v of the segment tree. Then we can do without the node lists $NL(v)$ in the plane sweep algorithm of the previous section. We only have to maintain for each node v a pointer into the array $M(v)$. The pointer is advanced whenever a segment has to be inserted into or deleted from $NL(v)$; furthermore, it always points to the segment of maximal z-coordinate in $NL(v)$.

This implies that the hidden line algorithm runs in time $O(n \log n + k \log(n^2/k))$.

To compute M, we process the segments in S one by one in order of decreasing z-coordinate. Let $S' \subseteq S$ be the set of segments processed so far. We maintain a partition C of T into intervals with the following semantics: $\{t\} \in C$ iff there is no segment $s \in S'$ with $t \in$ life-span(s). The union find structure of Gabow and Tarjan [5] is used to maintain the partition C. This data structure allows to process a sequence of n_v of the following operations in time $O(n_v)$ on a random access machine (see [5] for details).

FIND(t), $t \in T$, returns the right endpoint of the interval containing t.

UNION(t), $t \in T$, unites the interval containing t with its right neighbour interval.

From the definition of C it is clear that a new segment $s = (t_1, t_2, z) \notin S'$ is visible for all $t \in [t_1, t_2]$ with $\{t\} \in C$. Therefore the following algorithm computes M for a node v. Figure 4 shows the processing of the third segment.

```
0  Procedure COMPUTE_M(v)
1  for all t ∈ T do
2    M[t] := nil
3    add the singleton {t} to C
4  od
5  for all s ∈ S in decreasing z-order do
6    t'1 := FIND(t1)
7    t'2 := FIND(t2)
8    while t'1 ≠ t'2 do
9      if M[t'1] = nil then M[t'1] := s fi
10     UNION(t'1)
11     t'1 := FIND(t'1)
12   od
13 od
```

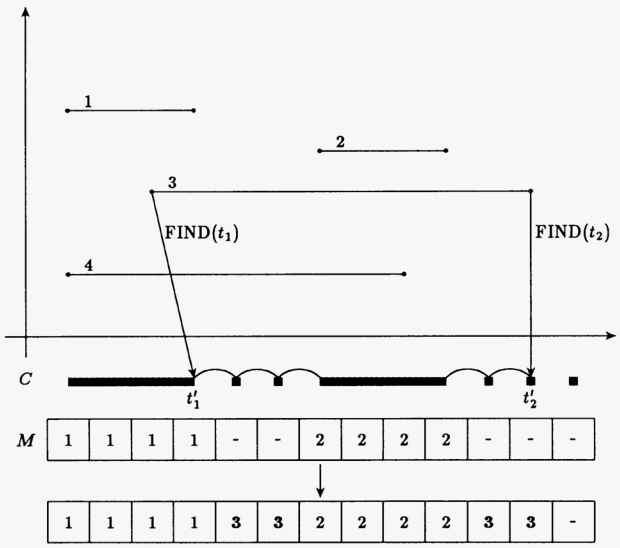

Fig. 4. Computing M: Processing interval 3.

The running time of this algorithm is clearly $O(|T| + |S| + \#$ of union operations). Since the number of union operations is at most T, $|S| \leqslant |T|$ and $|T| = 2n_v$, the running time is linear.

It remains to discuss how we determine the node lists for each node v and set up the M-problem defined above for each node v. This is done as follows.

(0) Sort the x-, y- and z-coordinates each in increasing order.

(1) Rename all occurring coordinates in each direction from 1 to $2n$.

(2) Sweep the sweep plane from $y = -\infty$ to $y = \infty$ in the following way:

(2.1) $S \leftarrow \emptyset$

(2.2) **for all** $y \in Y$ in increasing order **do**

(2.3) **for all** v, range(v) $\subseteq s$ and range (parent(v)) $\not\subseteq s$ where $s = [y_0, y_1, z]$ **do**

(2.4) $S \leftarrow S \cup \{(v, z, s, [y_0, y_1])\}$

(3) Sort the elements of S using bucket sort lexicographically according to the pair (v, z) as sort key. This computes for each node v the list $L(v)$ of all segments in order of decreasing z-coordinate. Let $n_v = |L(v)|$.

(4) Build for each tupel $(v, z, s, [y_0, y_1])$ two pairs (v, y_0) and (v, y_1) and sort all pairs lexicographically. This sorts for each node v the segments in $L(v)$ according to insertion and deletion time.

(5) Substitute for each node v the occuring y-coordinates by numbers from 1 to $2n_v$ in increasing order and change the life span of each edge according to this new universe.

5. The time bounds

Lemma 4. *The preprocessing takes time* O(n log n).

Proof.

Step (0): Obviously O(n log n).

Step (1): Obviously O(n).

Step (2): Because of (S1) and (S2), finding the nodes v with $s \in NL(v)$ needs O(log n) time per segment s. Since s is a set and not specially organized, it takes time O(n log n) altogether.

Step (3) Since there are only O(n log n) entries, step (3) takes time O(n log n).

Step (4): Obviously O(n log n).

Step (5): Obviously O(n log n).

In the solution of the two-dimensional visibility problem we execute for each node O(n_v) union and find operations. That takes time O(n_v). The sum over all nodes of the segment tree gives us the time bound of O(n log n) for the computation of the M-lists. □

Theorem. *The final algorithm needs time* O(n log n + k log(n^2/k)) *where k is the complexity of the visible scene and* O(n log n) *space.*

Proof. The operations "INSERT(s, v)" and "DELETE(s, v)" need time O(1) per node and O(log n) for the whole call "INSERT(s, root)" or "DELETE(s, root)". Then Lemmas 1–4 give the time bound. The space bound is obvious. □

References

[1] M.J. Atallah, M.G. Goodrich and M.H. Overmars, New output-sensitive methods for rectilinear hidden surface removal, Presented on a Workshop on Computational Geometry in Princeton, October 1989.

[2] J.L. Bentley, Solutions to Klee's rectangle problem, Manuscript, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1977).

[3] B. Bern, Hidden surface removal for rectangles, in: *Proc. 4th ACM Symposium on Computational Geometry* (1988) 183–192.

[4] B. Bern, Hidden surface removal for rectangles, Pers. Comm., October 1989.

[5] H.N. Gabow and R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, in: *Proc. 15th Annual SIGACT Symposium* (1983) 246–251.

[6] R.H. Güting and T.H. Ottmann, New algorithms for special cases of the hidden line elimination problem, *Comput. Vision Graph. Image Process.* **40** (1987).

[7] K. Mehlhorn, *Data Structures and Algorithms Vol. 3* (Springer, Berlin, 1984).

[8] K. Mehlhorn and S. Näher, Dynamic fractional cascading, *Algorithmica*, to appear.

[9] O. Nurmi, A fast line sweep algorithm for hidden line elimination, *BIT* **25** (1985).

[10] F.P. Preparata, J.S. Vitter and M. Yvinec, Computation of the axial view of a set of isothetic parallelepipeds, Rapport du Recherche du LIENS (1988).