Computing the Visibility Map of Fat Objects*

Mark de Berg and Chris Gray

Department of Computing Science, TU Eindhoven
{mdberg,cgray}@win.tue.nl

Abstract. We give an output-sensitive algorithm for computing the visibility map of a set of n constant-complexity convex fat polyhedra or curved objects in 3-space. Our algorithm runs in $O((n + k)\,\mathrm{polylog}\,n)$ time, where k is the combinatorial complexity of the visibility map. This is the first algorithm for computing the visibility map of fat objects that does not require a depth order on the objects and is faster than the best known algorithm for general objects. It is also the first output-sensitive algorithm for curved objects that does not require a depth order.

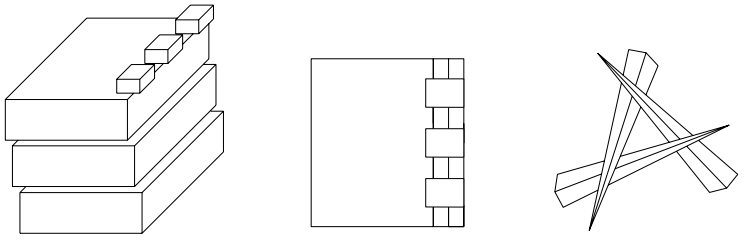
1 Introduction

Hidden-surface removal is an important and well-studied computational-geometry problem with obvious applications in computer graphics. The problem is to find those portions of objects in a scene that are visible from a given viewpoint. There are two main approaches to the hidden-surface removal problem: the *image-space approach* and the *object-space approach*. In the former, one calculates the visible object for each pixel of the image; the well known Z-buffer algorithm is the standard example of this. In the latter, one computes the so-called *visibility map* of the scene, which gives an exact description of the visible part of each object; this is the approach taken in computational geometry.

Formally, the visibility map of a set $\mathcal{P}$ of objects in $\mathbb{R}^3$ with respect to a viewpoint p is defined as the subdivision of the viewing plane into maximal regions such that in each region a single object in $\mathcal{P}$ is visible from p, or no object is visible. We will assume in this paper, as is usual, that the objects are disjoint. The visibility map of a set of n constant-complexity objects can be computed in $O(n^2)$ time [17]. Since the (combinatorial) complexity of the visibility map can be $\Omega(n^2)$—a set of n long and thin triangles that form a grid-like pattern when projected on the viewing plane is an example—this is optimal in the worst case. In most cases, however, the complexity of the visibility map is much smaller than quadratic. Therefore the main challenge in the design of algorithms for computing visibility maps has been to obtain *output-sensitive* algorithms: algorithms whose running time depends not only on the complexity of the input, n, but also on the complexity of the output (that is, the visibility map), k. Ideally the running time should be near-linear in n and k.

* This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

F. Dehne, J.-R. Sack, and N. Zeh (Eds.): WADS 2007, LNCS 4619, pp. 251–262, 2007.



(i) (ii)

Fig. 1. (i) The visibility map of fat boxes can have quadratic complexity. Left: the scene. Right: the visibility map for $p = (0, 0, \infty)$. (ii) The visibility map of a scene with cyclic overlap.

The first output-sensitive algorithms for computing visibility maps only worked for polygons parallel to the viewing plane or for the slightly more general case that a depth order on the objects exists and is given [10,13,14,19,20,21]. Unfortunately a depth order need not exist since there can be cyclic overlap among the objects—see Figure 1 (ii). De Berg and Overmars [7] (see also [3]) developed a method to obtain an output-sensitive algorithm that does not need a depth order. When applied to axis-parallel boxes (or, more generally, c-oriented polyhedra) it runs in $O((n+k)\log n)$ time [7] and when applied to arbitrary triangles it runs in $O(n^{1+\varepsilon} + n^{2/3+\varepsilon}k^{2/3})$ time [1]. Unfortunately, the running time for the algorithm when applied to arbitrary triangles is not near-linear in n and k; for example, when $k = n$ the running time is $O(n^{4/3+\varepsilon})$. For general curved objects no output-sensitive algorithm is known,[1] not even when a depth order exists and is given.

In this paper we study the hidden-surface removal problem for so-called *fat objects*—see the next section for a definition of fatness. As illustrated in Figure 1, the complexity of the visibility map of fat objects can still be $\Theta(n^2)$, so also here the main challenge is to obtain an output-sensitive algorithm. Fat objects have received ample attention over the past decade or so, both from a combinatorial and from an algorithmic point of view, and many problems can be solved much more efficiently for fat objects than for general objects. Since hidden-surface removal has been widely studied in computational geometry, it is not surprising that it has also been studied for fat objects: Katz *et al.* [15] gave an algorithm with running time $O((U(n) + k)\log^2 n)$, where $U(m)$ denotes the maximum complexity of the union of the projection onto the viewing plane of any subset of m objects. Since $U(m) = O(m \log\log m)$ for fat polyhedra [18] and $U(m) = O(\lambda_{s+2}(m)\log^2 m)$ for fat curved objects [5], their algorithm is near-linear in n and k. (Here $\lambda_{s+2}(n)$ is the maximum length of an $(n, s+2)$ Davenport-Schinzel sequence; $\lambda_{s+2}(n)$ is almost linear in n.) However, the algorithm only works if a depth order exists and is given. This leads to the main question we wish

[1] Some of the algorithms can be generalized to curved objects using standard techniques. The resulting algorithms are not very efficient, however, and typically have running time close to quadratic even when the visibility map has linear complexity.

to answer: is it possible to obtain an output-sensitive hidden-surface removal algorithm for fat objects that is near-linear in n and k and does not need a depth order on the objects? We answer this question affirmatively by giving an algorithm with running time $O((n+k)\,\mathrm{polylog}\,n)$ for fat convex objects of constant-complexity. More precisely, the running time is $O((n\log n(\log\log n)^2 + k)\log^3 n)$ when the objects are polyhedra, and it is $O((n\log^{5+\varepsilon} n + k)\log^3 n)$ when the objects are curved.

The main difficulty we have to overcome is that the only known method for output-sensitive hidden-surface removal that can handle objects without depth order [3,7] needs an auxiliary data structure for ray shooting in so-called *curtains*—these are semi-infinite surfaces, extending downward from the edges of the input objects—and it appears to be difficult to profit from the fact that the objects are fat when implementing this data structure. This also explains why there is no (efficient) output-sensitive algorithm for hidden-surface removal in curved objects: there are no efficient data structures known for ray shooting (with curved rays, in this case) in curved curtains. Our method therefore works differently: instead of ray shooting in curtains, we trace the rays on several two-dimensional planes by performing many simultaneous and coordinated sweeps on these planes. To obtain a suitable set of planes, we use a suitably augmented variant of the BSP for ray shooting that was recently introduced by De Berg [4].

2 Preliminaries

Let $\mathcal{P}$ be a set of disjoint convex objects in $\mathbb{R}^3$. We assume the objects are β-fat according to the following definition of fatness [9]: an object o in $\mathbb{R}^d$ is *β-fat* if for any ball b whose center lies in o and that does not fully contain o, we have $vol(b\cap o) \geq \beta \cdot vol(b)$, where $vol(o)$ denotes the volume of o. (For convex objects this definition is equivalent, up to constant factors, to other definitions of fatness that have been proposed.)

We define size(o), the *size* of an object o, to be the radius of the smallest enclosing ball of o. The *density* of a set S of objects is defined as the smallest number λ such that any ball b is intersected by at most λ objects $o \in S$ such that $size(o) \geq size(b)$. The following well-known lemma [9] relates the density of a group of objects to the fatness constant.

Lemma 1. **[9]** *A set of disjoint β-fat objects has density λ where $\lambda = O(1/\beta)$.*

For a curve e in $\mathbb{R}^3$ define the *curtain* of e, denoted curt(e), as the ruled surface constructed by taking a vertical ray pointing downward and moving its starting point from one end of e to the other. Thus, if e is a segment then curt(e) is an infinite polygon defined by e and two unbounded edges, each parallel to the z-axis. For a set E of curves we let $\mathrm{curt}(E) := \{\mathrm{curt}(e) | e \in E\}$.

Next we define some notation and terminology relating to visibility maps. We assume from now on that we are looking at the scene from above with the viewpoint at $z = \infty$—hence, we are dealing with a parallel view. As already mentioned, the visibility map $\mathcal{M}(\mathcal{P})$ of $\mathcal{P}$ is the subdivision of the viewing plane

into maximal regions such that in each region a single object in $\mathcal{P}$ is visible from the viewpoint p, or no object is visible. We assume without loss of generality that the viewing plane is the xy-plane.

Consider an object $o \in \mathcal{P}$. We denote the projection of o onto the viewing plane by $\text{proj}(o)$. Since o is convex, the boundary of $\text{proj}(o)$ consists of the projection of all points of vertical tangency of o. Let $\sigma(o)$ denote the curve[2] on the boundary of o that projects onto the boundary of $\text{proj}(o)$. Note that if o is polyhedral, $\sigma(o)$ consists of certain edges of o. We cut $\sigma(o)$ into two pieces at the points of minimum and maximum x-coordinate; we can assume without loss of generality that these points are unique. We call these pieces *silhouette curves*—note that for polyhedral objects a silhouette curve consists of multiple edges of the object—and their endpoints *vertices*.

$\mathcal{M}(\mathcal{P})$ is a plane graph whose *nodes* are intersection points of projected silhouette curves and whose *arcs* are portions of projected silhouette curves. Arcs of the visibility map will be denoted by a, and silhouette curves by e. The curve whose projection contains the arc a is denoted $e(a)$. It will be convenient to also consider the projections of visible endpoints of silhouette curves (that is, visible vertices) as nodes. Since we cut $\sigma(o)$ into two pieces when it changes direction with respect to the x-axis, the arcs of $\mathcal{M}(\mathcal{P})$ are x-monotone.

The existing output-sensitive hidden-surface removal algorithm from [3] works as follows. It sweeps over the viewing plane from left to right, detecting the arcs of the visibility map along the way. Note that the left endpoint of an arc is one of two types. It is either the projection of a visible vertex of a silhouette curve or it is the right endpoint of some other arc.

To detect left endpoints of the first type we need a data structure to determine for a vertex v whether it is visible or not. If it is, two new arcs start at $\text{proj}(v)$, which are contained in the projections of the two silhouette curves incident to v. Detecting if v is visible can be done by vertical ray shooting: shoot a ray from v vertically upwards; if no object is hit then v is visible.

We also need to be able to detect the right endpoint of an arc (and thereby the left endpoints of the second type). An arc a can end for two reasons. One is that the silhouette curve $e(a)$ projecting onto a ends. The other is that $\text{proj}(e(a))$ intersects some other projected silhouette curve $\text{proj}(e')$ such that either $e(a)$ becomes invisible or e' becomes visible—see Figure 2 (i). These two latter events are detected using a ray shooting operation in a set of curtains, as explained next. When $e(a)$ becomes invisible because it disappears below some object o, then the ray along $e(a)$ must hit the curtain hanging from one of o's silhouette curves. When some other silhouette curve e' becomes visible, something similar holds. To this end, we define a ray[3] $\rho(a)$ for an arc a of the visibility map as follows. Let q be the point on $e(a)$ projecting onto the left endpoint of a. Project the portion of $e(a)$ to the right of q onto the object $o(q)$ immediately below q. (If there is no such object, we project onto a plane below all objects.) This gives us

[2] For simplicity of presentation we assume o does not have any vertical facets, so that $\sigma(o)$ is uniquely defined. It is easy to adapt the definitions to the general case.

[3] Note that in case of curved objects, the ray will be curved.

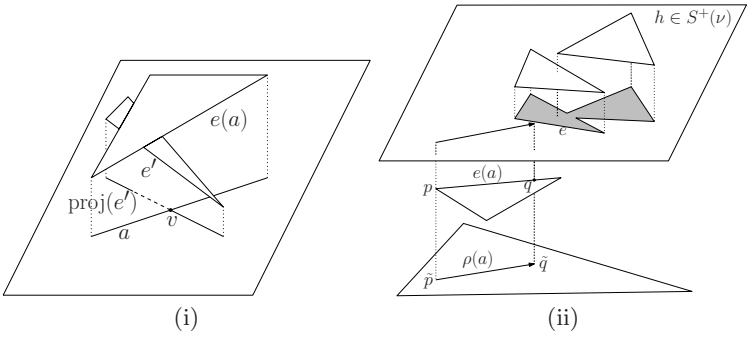


Fig. 2. (i) The node v in the visibility map is made by the intersection of $\text{proj}(e)$ and $\text{proj}(e')$. (ii) $\rho(a)$ hits a curtain in $\text{curt}(E)$ at point q when its projection intersects a silhouette curve of a union stored at $S^+(\nu)$. Note that the objects pictured here are not fat under our definition, but could be the top surfaces of fat polyhedra. We draw the objects in this way to ease visualization.

a ray on the surface of $o(q)$ whose projection contains a. It can be argued [3] that the point where $\rho(a)$ hits $\text{curt}(e')$ corresponds to the point where the silhouette curve e' becomes visible. (Note that $\rho(a)$ is about to leave $o(q)$ when it reaches a silhouette curve of $o(q)$; in this case $\rho(a)$ hits the curtain hanging from that silhouette curve, which is then the curve that becomes visible.) Since any curtain hit by the ray along $e(a)$ is also hit by $\rho(a)$—after all, $\rho(a)$ is below $e(a)$—we can detect events where $e(a)$ becomes invisible by shooting along $\rho(a)$ as well.

The next lemma summarizes the discussion above.

Lemma 2 ([3]). *Let E be the set of silhouette curves of the objects in $\mathcal{P}$. The right endpoint of an arc a of $\mathcal{M}(\mathcal{P})$ is the leftmost of the following event points:*

- *The projection of the right endpoint of $e(a)$.*
- *The projection of the first intersection of $\rho(a)$ with a curtain in* $\text{curt}(E)$.

3 The Algorithm

As mentioned in the introduction, it seems hard to implement a structure for ray shooting in curtains that profits from the fact that the objects are fat. Therefore we use the following idea.

Suppose that all objects are above a plane h and the query ray $\rho(a)$ is below h. Then we can project all objects and the ray onto h, and shoot with the projected ray in the union of the projected objects; the point where the ray hits a curtain then corresponds to the point where the projected ray hits the union. This is true because in our application the ray will always be visible, so the projected ray cannot start inside the union. Unfortunately two-dimensional ray shooting is still too costly. If, however, we have to answer many queries, then we can project all of them onto h, and perform a sweep to detect when they intersect the union.

Of course there will not be a plane h that nicely separates all objects from all rays. Therefore we construct a binary space partition (a *BSP*) on the objects. This will basically give us a collection of $O(\log n)$ planes that separate any ray from the objects. The ray will then be traced on each of these planes. In the next section we make this idea more precise.

We start by describing the BSP in Section 3.1, then discuss in Section 3.2 the correspondence between ray shooting in curtains and tracing rays on a suitable set of planes, and finally we give the details of the algorithm in Section 3.3.

3.1 The Data Structure

A *balanced aspect ratio tree* (or *BAR-tree* for short) is a special type of BSP for storing points. It was introduced by Duncan [11,12]. The variant known as the *object BAR-tree* [8] stores objects rather than points and has proved especially useful in designing data structures for fat objects. It has been used as a basis for vertical ray shooting [4,6] as well as approximate range searching and nearest neighbor searching [8].

We denote the region associated with a node ν in the object BAR-tree for $\mathcal{P}$ by region(ν), and we let $\mathcal{P}_\nu$ denote the set of all objects $o \in \mathcal{P}$ intersecting region(ν), clipped to region(ν). The following lemma states the properties of the object BAR-tree we will need.

Lemma 3. **[8]** *Let $\mathcal{P}$ be a set of n β-fat disjoint convex objects in $\mathbb{R}^d$. An object BAR-tree on $\mathcal{P}$ is a BSP tree $\mathcal{T}$ for $\mathcal{P}$ with the following properties:*

(i) the tree has $O(n)$ leaves; each leaf region intersects $O(1/\beta)$ objects from $\mathcal{P}$;
(ii) the depth of the tree is $O(\log n)$;
(iii) for each node ν, region(ν) *has constant complexity and fatness.*

De Berg [4] has shown how to augment an object BAR-tree $\mathcal{T}$ with secondary structures, so that vertical ray shooting can be performed efficiently. The augmentation is as follows.

- For each leaf node μ of $\mathcal{T}$, we store the set $\mathcal{P}_\mu$ in a list $\mathcal{L}_\mu$.
- For an internal node ν, let h_ν denote the splitting plane stored at ν.
  - If h_ν is vertical, then we store the set $\{h_\nu \cap o : o \in \mathcal{P}_\nu\}$—that is, the cross-sections of the polyhedra in $\mathcal{P}_\nu$ with h_v—in a structure $\mathcal{T}_\nu$, which is an optimal point-location structure [16] on the trapezoidal map defined by $h_\nu \cap \mathcal{P}_\nu$.
  - If h_ν is not vertical, then ν has two associated data structures, $\mathcal{T}_\nu^+$ and $\mathcal{T}_\nu^-$, defined as follows.
    Let $\mathcal{P}_\nu^+$ denote the set of object parts from $\mathcal{P}_\nu$ lying above h_ν. Thus $\mathcal{P}_\nu^+ = \mathcal{P}_\mu$, where μ is the child of ν corresponding to the region above h_ν. Let proj($\mathcal{P}_\nu^+$) denote the set of vertical projections of the objects in $\mathcal{P}_\nu^+$ onto h_ν. Then $\mathcal{T}_\nu^+$ is an optimal point-location structure for $U(\text{proj}(\mathcal{P}_\nu^+))$, the union of proj($\mathcal{P}_\nu^+$). In our application, we not only store the point-location structure for $U(\text{proj}(\mathcal{P}_\nu^+))$, but also an explicit list of all union edges.
    The associated structure $\mathcal{T}_\nu^-$ is defined similarly, but this time for the object parts below h_ν.

Recall that we want to use the structure to answer ray shooting queries in curtains, where the query rays are projections of (parts of) silhouette curves onto the object immediately below. A problem with this approach is that an object may be cut into many pieces,[4] and we would then have to spend time whenever the ray goes from one piece to the next.

To avoid this problem we need some extra information. In particular, for each object $o \in \mathcal{P}$ we need to store the union of the projection of a certain subset $\mathcal{P}(o) \subset \mathcal{P}$ onto $\partial^+(o)$, the top surface of o. (The top surface of o is the part of the boundary of o visible from above.) The subset $\mathcal{P}(o)$ is defined as follows.

Call an object o *large* at a node ν of $\mathcal{T}$ if o intersects region(ν) and the following two conditions are met: (i) size(o) $<$ size(region(parent(ν))) and (ii) either size(o) $\geq$ size(region(ν)) or ν is a leaf. Now we define

$$\mathcal{P}(o) := \{\, o' \in \mathcal{P} : \text{there is a node } \nu \text{ such that } o \text{ is large at } \nu, \\ o' \text{ intersects region}(\nu) \text{ and } o' \text{ is above } o \,\}$$

Besides these extra unions on the top surface of each object o, we also need the union of the projections of all the objects in $\mathcal{P}$ onto the xy-plane. (The xy-plane can be seen as a dummy object added below the whole scene, which is large at the root of $\mathcal{T}$.)

Next we analyze the cost of the additional information.

Lemma 4. *Any object $o \in \mathcal{P}$ is large at $O(\log n)$ nodes, and at any node ν there are $O(1/\beta)$ large objects.*

Proof. By Lemma 3 we know that every cell of $\mathcal{T}$ is $O(1)$-fat. This means that any collection of disjoint cells has density $O(1)$. Therefore, since the cells at any level of the BAR-tree are disjoint, the number of nodes ν in any level of the BAR-tree intersecting some $o \in \mathcal{P}$ with size(region(ν)) $\geq$ size(o) is $O(1)$. An object o can only be large at the node ν if size(region(parent(ν))) $\geq$ size(o). Thus, the number of cells per level at which o can be large is $O(1)$. Finally we know that $\mathcal{T}$ has $O(\log n)$ levels by Lemma 3, proving the first part of the lemma.

A set of disjoint β-fat objects has density $O(1/\beta)$—see Lemma 1—which, together with Lemma 3(i), implies the second part. □

From Lemma 4 we derive:

$$\begin{aligned} \textstyle\sum_o |\mathcal{P}(o)| &\leq \textstyle\sum_\nu \{(\#\text{ large objects at } \nu) \cdot (\#\text{ objects intersecting region}(\nu))\} \\ &\leq O(1/\beta) \cdot \textstyle\sum_\nu |\mathcal{P}_\nu| \;\leq\; O((1/\beta) \cdot n \log n), \end{aligned}$$

where the last inequality follows from [4]. Together with the known bounds on the union of fat objects [5,18] this is easily seen to imply that the total amount of

[4] The fact that an object is cut into many pieces also prevents us from applying the following simple strategy: compute the object BAR-tree, use it to find a depth order on the resulting set of pieces, and apply the algorithm of Katz *et al.* [15]. The problem is that the visibility map of the pieces may be much more complex than the visibility map of the original objects.

storage and preprocessing time for the unions of the projections of $\mathcal{P}(o)$ onto the top surfaces $\partial^+(o)$ does not increase the total amount of storage or preprocessing asymptotically, and the bounds we get are the same as in [4]. (The constants in the O-notation depend on the fatness factor β.)

Lemma 5. *Let β be a fixed constant. The data structure above for convex β-fat polyhedral objects requires $O(n \log^3 n (\log\log n)^2)$ storage and $O(n \log^4 n (\log\log n)^2)$ preprocessing time, and $O(n \log^{7+\varepsilon} n)$ storage and $O(n \log^{8+\varepsilon} n)$ preprocessing time for convex β-fat curved objects. With this structure, we can answer vertical ray shooting queries in $O(\log^2 n)$ time.*

3.2 Tracing an Arc

Recall that the right endpoint of an arc a can be found by shooting with $\rho(a)$ in $curt(E)$. Next we explain how to find the right endpoint of a using the unions stored in $\mathcal{T}$ and additional unions described above. The key is to find a collection of $O(\log n)$ unions such that the first point where $\rho(a)$ hits a curtain corresponds to the first point where one of the unions is hit.

To this end we first define for a node ν a collection $S^+(\nu)$ of $O(\log n)$ splitting planes, which consists of those splitting planes $h_{\nu'}$ such that ν' is an ancestor of ν and region(ν) is below $h_{\nu'}$. Now let $e(a)$ be the silhouette curve defining an arc a, and let $p \in e(a)$ be the point projecting onto the left endpoint of a. Recall that $\rho(a)$ is a ray on the top surface of the object o directly below p. We denote the projection of p onto o by $\tilde{p}$.

The first curtain hit by $\rho(a)$ can now be found using the following lemma.

Lemma 6. *Let $\rho(a)$ be a ray on the top surface of an object $o \in \mathcal{P}$, let $\tilde{p}$ be the starting point of $\rho(a)$, and let ν be the node in $\mathcal{T}$ such that $\tilde{p} \in$ region(ν) and o is large at ν. Then $\rho(a)$ hitting a curtain from $curt(E)$ inside* region(ν) *corresponds to (a suitable projection of) $\rho(a)$ hitting either the union of (the projection of) $\mathcal{P}(o)$ on o or a union on one of the splitting planes in $S^+(\nu)$.*

Proof. Note that the node ν referred to in the lemma is unique and must exist, since we consider the xy-plane to be a dummy object below the whole scene.

Let $\tilde{q}$ be the first point where $\rho(a)$ intersects a curtain in $curt(E)$, let e be the silhouette curve defining the curtain, and let $q \in e$ be the point directly above $\tilde{q}$. If $q \in$ region(ν) then the object containing the silhouette curve e is a member of $\mathcal{P}(o)$ and we are done. Otherwise there is a splitting plane $h_{\nu'}$ stored at some ancestor ν' of ν with q above $h_{\nu'}$ and $\tilde{q}$ below $h_{\nu'}$. Then the relevant portion of e must be part of the union stored at the first such node ν' (as seen from the root of $\mathcal{T}$). See Figure 2 (ii).

Conversely, since all the unions considered are generated by (parts of) objects above o, we know that $\rho(a)$ cannot hit such a union before it hits a curtain. □

3.3 Details of the Algorithm

We now describe a space-sweep algorithm for computing the visibility map of a set $\mathcal{P} = \{o_1, \ldots, o_n\}$ of convex, constant-complexity, β-fat objects. We move a

sweep plane h parallel to the yz-plane from left to right through space. The space sweep induces a plane sweep for each of the unions stored in $\mathcal{T}$. Thus, instead of thinking about the algorithm as a 3D sweep, one may also think about it as a number of coordinated 2D sweeps. That is, while we sweep $\mathbb{R}^3$ with h, we also sweep each (non-vertical) splitting plane h_ν with the line $h \cap h_\nu$. This 2D sweep is performed to detect intersections of the union on h_ν with certain rays (projected onto h_ν). The same holds for the unions stored for each object: while we sweep $\mathbb{R}^3$ with h, we sweep the top surface $\partial_{\text{top}}(o)$ of each object o with the curve $h \cap \partial_{\text{top}}(o)$. Finally, the sweep of h induces a sweep on the viewing plane. As in the algorithm from [3], the visibility map will be computed as we go, so that at the end of the sweep the entire visibility map has been computed.

The space-sweep algorithm is supported by the following data structures:

- There is a global event queue Q, where the priority of an event is its x-coordinate. Initially, all vertices of the objects (that is, all endpoints of silhouette curves) are placed into Q. In addition, all vertices of any of the unions stored in $\mathcal{T}$ are placed into Q. During the sweep, new event points will be inserted into Q, for example endpoints of arcs of the visibility map. It is also possible that events will be removed before they are handled.
- For every splitting plane h_ν (and the top surface of every object o) we maintain a balanced binary tree, which we will call the *intersection-detection data structure*. This tree will store the edges of the union on the splitting plane (resp. $\partial_{\text{top}}(o)$) that intersect the sweep line $h \cap h_\nu$ (resp. $h \cap \partial_{\text{top}}(o)$) as well as the rays traced on it that intersect the sweep line; the edges and rays are stored in order of their intersection with the sweep line. Thus we are essentially running the standard line-segment intersection algorithm of Bentley and Ottmann [2] on the union edges and rays.

Next we discuss the events that can take place, and how they are handled.

(i) *The sweep reaches the left endpoint of an arc a.*
Let $e(a)$ be the silhouette curve defining a, and let $p \in e(a)$ be the point whose projection is the left endpoint of a. Let o be the first object that a vertical ray downward from p hits, and ν be the node where o is large in region(ν) and $\tilde{p} \in$ region(ν). Determine $S^+(\nu)$, and insert the portion of $e(a)$ starting at p into each of the intersection-detection data structures associated with the splitting planes in $S^+(\nu)$. (More precisely, the projection of the silhouette curve on the plane is added.) Also add the projection of the silhouette curve onto $\partial_{\text{top}}(o)$ to the intersection-detection structure for o. Determine any new events using these data structures in the standard way (that is, by checking new pairs of adjacent elements); add any new events to Q. Finally, add the following three events to Q: the right endpoint of $e(a)$, the (first) intersection of $\rho(a)$ with the boundary of region(ν), and the (first) intersection of $\rho(a)$ with the silhouette of o.

(ii) *The sweep reaches the right endpoint of an arc a.*
Determine ν and o as above. Remove a from all intersection-detection data structures in $S^+(\nu)$ and the intersection-detection data structure

associated with o. Remove all events associated with a from Q. Check for new events in each of the intersection-detection data structures; add any new events to Q. Output a as an arc of $\mathcal{M}$. (Note that the right endpoint of an arc may be the left endpoint of one or two other arcs; in this case the left endpoints will be separate events, which are handled according to case (i).)

(iii) *The sweep reaches the left vertex v of a silhouette curve.*
(In other words, we reach the leftmost point of an object $o \in \mathcal{P}$.) Determine if v is visible by shooting a ray vertically up from it. If v is visible, two arcs start at the projection of v onto the viewing plane. Run the actions from case (i) for each of these arcs.

(iv) *The sweep reaches the right vertex v of a silhouette curve it is currently tracing.*
Run the actions from case (ii) for the arc ending at the projection of v.

(v) *The sweep reaches the intersection point of the union boundary on some splitting plane (or top surface of an object) and an arc a traced on the plane (or top surface).*
This case corresponds to a hitting a curtain in $\mathrm{curt}(E)$. Now the arc a ends. Run the actions from case (ii) for a. One or two new arcs may start at this point, at most one along the silhouette curve $e(a)$, and one along the silhouette curve corresponding to the curtain that is hit. Run the action from case (i) for the new arc(s).

(vi) *The sweep reaches a point p where the projection of a currently visible silhouette curve onto the object o below hits the boundary of a cell ν where o is large.*
Remove a from all the intersection-detection data structures in $S^+(\nu)$ and all events associated with a from Q. Run the action for case (i) for the continuation of the arc a defined by the silhouette curve. (The only thing that happens here is that the set $S^+(\cdot)$ changes, because the ray that we are tracing moves out of a cell where the object o on which the ray is traced is large.)

(vii) *The sweep reaches the point where the object o immediately below a currently visible silhouette curve changes.*
Now p is the right endpoint of an arc a. Run the actions from case (ii) for a. Two new arcs start at p, one that is the continuation of a, and one that is along a silhouette curve of o (which became visible). Run the actions for case (i) on both curves.

(viii) *The sweep reaches a point on a splitting plane (or top surface of an object), where a union edge starts or ends.*
In this case we only have to update the relevant intersection-detection data structure, check for new events in the intersection-detection data structures, and add any new events to Q.

Lemma 7. *The number of events of type (i)–(vii) is $O(n + k \log n)$, where k is the complexity of $\mathcal{M}$, and the total number of events of type (viii) is $O(n \log^3 n (\log \log n)^2)$ for fat polyhedra and $O(n \log^{7+\varepsilon} n)$ for fat curved objects.*

Proof. Clearly, the number of events of types (i), (ii), (iv), (v), and (vii) is $O(k)$, since they can be charged to a vertex of $\mathcal{M}$. The number of events of type (iii) is $O(n)$. It remains to bound the number of events of type (vi). Consider the portion of a silhouette curve $e(a)$ defining some arc a. This portion has a unique object o immediately below it. Since o is large at $O(\log n)$ cells by Lemma 4 and the projection of $e(a)$ onto o can leave any cell only a constant number of times, we can conclude that there are only $O(\log n)$ type (vi) events for any arc a, this giving $O(k \log n)$ such events in total.

The bound on the number of events of type (viii) follows immediately from Lemma 5. □

Lemma 8. *The time taken for each event of type (i)–(vii) is $O(\log^2 n)$, and the time taken for each event of type (viii) is $O(\log n)$.*

Proof. In all event types, we may need to perform several actions: vertical ray shooting, updating intersection-detection data structures, determining a set $S^+(\nu)$, and updating Q.

By Lemma 5, the time taken for the vertical ray shooting is $O(\log^2 n)$. Each event needs to do only a constant number of ray shooting queries, so this is $O(\log^2 n)$ in total. The intersection-detection data structures are balanced binary trees, so updates take $O(\log n)$ time. At each event we have to update $O(\log n)$ intersection-detection data structures, so the total time taken for updating is $O(\log^2 n)$. Determining new events in the intersection-detection data structures takes $O(1)$ per data structure, so the total amount of time taken for events of type (iii) is $O(\log^2 n)$. Determining a set $S^+(\nu)$ can be done in $O(\log n)$ time by searching in $\mathcal{T}$. At each event we may have to remove $O(\log n)$ event points from Q, each removal taking $O(\log n)$ time. Hence, all events of type (i)–(vii) can be handled in $O(\log^2 n)$ time, as claimed.

The events of type (viii) require $O(\log n)$ time, since they involve a constant number of operations on a single intersection-detection data structure. □

The correctness of the algorithm follows from Lemmas 2 and 6 as well as the correctness of the algorithm in [3]. We conclude with the following theorem.

Theorem 1. *The visibility map of a set of n disjoint constant-complexity convex β-fat polyhedra in $\mathbb{R}^3$ can be computed in time $O((n \log n (\log\log n)^2 + k) \log^3 n)$, where k is the complexity of the visibility map. When the objects are curved (and disjoint, constant-complexity, convex, and β-fat) the visibility map can be computed in time $O((n \log^{5+\varepsilon} n + k) \log^3 n)$.*

References

1. Agarwal, P.K., Matoušek, J.: Ray shooting and parametric search. SIAM Journal on Computing 22(4), 794–806 (1993)
2. Bentley, J., Ottmann, T.: Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers 28, 643–647 (1979)
3. de Berg, M.: Ray Shooting, Depth Orders and Hidden Surface Removal. LNCS, vol. 703. Springer, Heidelberg (1993)

4. de Berg, M.: Vertical ray shooting for fat objects. In: Proc. 21st Annual Symposium on Computational Geometry, pp. 288–295 (2005)
5. de Berg, M.: Improved bounds for the union complexity of fat objects. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 116–127. Springer, Heidelberg (2005)
6. de Berg, M., Gray, C.: Vertical ray shooting and computing depth orders for fat objects. In: Proc. 17th Annual Symposium on Discrete Algorithms, pp. 494–503 (2006)
7. de Berg, M., Overmars, M.H.: Hidden-surface removal for c-oriented polyhedra. Comput. Geom. Theory Appl. 1, 247–268 (1992)
8. de Berg, M., Streppel, M.: Approximate range searching using binary space partitions. In: Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 110–121 (2004)
9. de Berg, M., van der Stappen, A.F., Vleugels, J., Katz, M.J.: Realistic input models for geometric algorithms. Algorithmica 34(1), 81–97 (2002)
10. Bern, M.: Hidden surface removal for rectangles. J. Comp. Syst. Sciences 40, 49–69 (1990)
11. Duncan, C.: Balanced Aspect Ratio Trees. PhD thesis, Johns Hopkins University (1999)
12. Duncan, C., Goodrich, M., Kobourov, S.: Balanced aspect ratio trees: Combining the advantages of k-d trees and octtrees. In: Proc. 10th Annual ACM-SIAM Sympos. on Discrete Algorithms, pp. 300–309 (1999)
13. Goodrich, M.T., Atallah, M.J., Overmars, M.H.: An input-size/output-size trade-off in the time-complexity of rectilinear hidden surface removal. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 689–702. Springer, Heidelberg (1990)
14. Güting, R.H., Ottmann, T.: New algorithms for special cases of the hidden line elimination problem. Comp. Vision, Graphics and Image Processing 40, 188–204 (1987)
15. Katz, M.J., Overmars, M., Sharir, M.: Efficient hidden surface removal for objects with small union size. Computational Geometry: Theory and Applications 2, 223–234 (1992)
16. Kirkpatrick, D.: Optimal search in planar subdivisions. SIAM J. Comput. 12, 28–35 (1983)
17. McKenna, M.: Worst-Case Optimal Hidden Surface Removal. ACM Trans. Graphics 6, 19–28 (1987)
18. Pach, J., Tardos, G.: On the boundary complexity of the union of fat triangles. SIAM J. Comput. 31, 1745–1760 (2002)
19. Preparata, F.P., Vitter, J.S., Yvinec, M.: Computation of the axial view of a set of isothetic parallellipipes. ACM Trans. Graphics 9, 278–300 (1990)
20. Reif, J., Sen, S.: An efficient out-sensitive hidden surface removal algorithm and its parallelization. In: Proc. 4th Annual Symposium on Computational Geometry, pp. 193–200 (1988)
21. Sharir, M., Overmars, M.H.: A simple method for output-sensitive hidden surface removal. ACM Trans. Graphics 11, 1–11 (1992)