SIAM J. COMPUT.
Vol. 38, No. 1, pp. 257–275

VERTICAL RAY SHOOTING AND COMPUTING DEPTH ORDERS FOR FAT OBJECTS*

MARK DE BERG† AND CHRIS GRAY†

Abstract. We present new results for three problems dealing with a set $\mathcal{P}$ of n convex constant-complexity fat polyhedra in 3-space. (i) We describe a data structure for vertical ray shooting in $\mathcal{P}$ that has $O(\log^2 n)$ query time and uses $O(n \log^2 n)$ storage. (ii) We give an algorithm to compute in $O(n \log^3 n)$ time a depth order on $\mathcal{P}$ if it exists. (iii) We give an algorithm to verify in $O(n \log^3 n)$ time whether a given order on $\mathcal{P}$ is a valid depth order. All three results improve on previous results.

Key words. computational geometry, depth orders, fat objects, ray shooting

AMS subject classification. 68U05

DOI. 10.1137/060672261

1. Introduction.

Motivation. Many algorithms and data structures developed in computational geometry have a rather poor worst-case performance. This behavior is often caused by sets of objects in very intricate configurations, such as many long and thin objects packed closely together. For example, the worst-case running time of the best known general motion-planning algorithm is $\Theta(n^f)$, where f is the number of degrees of freedom of the robot, because for certain configurations of obstacles the free space has complexity $\Theta(n^f)$. The configurations that determine the worst-case behavior, however, are usually rather artificial constructions; one would expect that in practice the input is much more well behaved, and that better performance is possible than the worst-case analysis suggests.

These considerations have led to the study of geometric problems in so-called *realistic input models* [9]. Here one places certain restrictions on the shape and/or distribution of the input objects, so that hypothetical worst-case examples are excluded. The hope is that this will enable proving much stronger theoretical results than are possible for arbitrary inputs, while the results are still general enough for practical applications. One of the most widely studied realistic input models assumes that the input objects are *fat*, that is, they are not arbitrarily long and skinny—see the next section for a formal definition. For motion planning this has proved to be quite successful: the free space for a not-too-large robot moving amidst a set of fat obstacles has only linear complexity [26], which has enabled the development of motion-planning algorithms with near-linear running times in this setting [25].

In this paper we study two problems arising in computer graphics in the context of realistic input models: the vertical ray-shooting problem and the depth-order problem for fat polytopes in $\mathbb{R}^3$.

Problem statement and previous results. Let $\mathcal{P}$ be a set of n disjoint objects in $\mathbb{R}^3$.

*Received by the editors October 13, 2006; accepted for publication (in revised form) January 10, 2008; published electronically April 11, 2008. A preliminary version of this paper appeared in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 494–503. This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project 639.023.301.

http://www.siam.org/journals/sicomp/38-1/67226.html

†Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (mdberg@win.tue.nl, cgray@win.tue.nl).

The first problem we study is the *ray-shooting problem*: preprocess the set $\mathcal{P}$ such that ray-shooting queries (i.e., what is the first object in $\mathcal{P}$ hit by a query ray?) can be answered efficiently. Ray-shooting queries are important for realistic image synthesis: they form the basis of ray-tracing algorithms and can be used in radiosity methods to approximate form factors. Hence, data structures for ray shooting have received ample attention, in both computer graphics and computational geometry; the book by De Berg [5] and the survey by Pellegrini [24] discuss many of the (computational geometry) solutions. As for the motion-planning problem, the worst-case bounds that have been obtained for the general ray-shooting problem are rather disappointing. For ray-shooting queries with arbitrary rays in a collection of n disjoint triangles, for example, the best known structures that achieve $O(\log n)$ query time use $O(n^{4+\epsilon})$ storage [5, 23], and the best structures with near-linear storage have roughly $O(n^{3/4})$ query time [3]. For vertical ray shooting in a collection of disjoint triangles the situation is still not very rosy: to obtain $O(\log n)$ query time one needs $O(n^{2+\epsilon})$ storage, and with near-linear storage the query time becomes roughly $O(\sqrt{n})$ [5].

Given the prominence of the ray-shooting problem in computational geometry and the fact that general inputs do not seem to admit very efficient (near-linear) solutions, it is not surprising that ray shooting has already been studied from the perspective of realistic input models. In particular, the vertical ray-shooting problem—here the query ray is required to be vertical—has been studied for fat convex polyhedra. For this case Katz [19] presented a data structure that uses $O(n \log^3 n)$ storage and has $O(\log^4 n)$ query time. (In fact, Katz's solution works for polygons whose projections onto the xy-plane are fat, but it is not difficult to see that it works for fat three-dimensional polytopes as well.) Using the techniques of Efrat et al. [17], it is possible to improve the storage bound to $O(n \log^2 n)$ and the query time to $O(\log^3 n)$ [20]. Recently De Berg [6] presented a structure with $O(\log^2 n)$ query time; his structure uses $O(n \log^3 n (\log \log n)^2)$ storage.

The second problem we study is the *depth-order problem*: compute a depth order for the set $\mathcal{P}$, that is, an ordering $P_1, \dots, P_n$ of the objects in $\mathcal{P}$ such that if P_i is below P_j, then $i < j$. Here we say that P_i is below P_j, denoted by $P_i \prec P_j$, if there are points $(x, y, z_i) \in P_i$ and $(x, y, z_j) \in P_j$ with $z_i < z_j$. In other words, a depth order is a linear extension of the $\prec$-relation. Since there can be cycles in the $\prec$-relation—we then say there is *cyclic overlap* among the objects—a depth order does not always exist. In that case the algorithm should report that there is cyclic overlap. Depth orders are useful in several applications. For example, they can be used to render scenes with the Painter's algorithm [18] or to do hidden-surface removal with the algorithm of Katz, Overmars, and Sharir [21]. Depth orders also play a role in assembly planning [1].

The depth-order problem for arbitrary sets of triangles in 3-space does not seem to admit a near-linear solution; the best known algorithm runs in $O(n^{4/3+\varepsilon})$ time [11]. This has led researchers to also study this problem for fat objects. Agarwal, Katz, and Sharir [2] gave an algorithm for computing the depth order of a set of triangles whose projections onto the xy-plane are fat; their algorithm runs in $O(n \log^5 n)$ time. However, their algorithm cannot detect cycles—when there are cycles it reports an incorrect order. A subsequent result by Katz [19] produced an algorithm that runs in $O(n \log^5 n)$ time and that can detect cycles. In this case though, the constant of proportionality depends on the minimum overlap of the projections of the objects that do overlap. If there is a pair of objects whose projections barely overlap, then the running time of the algorithm increases greatly. One advantage that this algorithm has is that it can deal with convex curved objects.

Our results. First, we present a new data structure for vertical ray shooting in a collection of n convex constant-complexity fat polyhedra[1] in $\mathbb{R}^3$. Our structure uses $O((1/\beta)n \log^2 n)$ storage and has $O((1/\beta^2) \log^2 n)$ query time. Compared to Katz's structure [20] it has a better query time (while the storage is the same) and compared to De Berg's structure [6] it has a better storage bound (while keeping the same query time).

Second, we present a new algorithm for computing a depth order on a collection of n convex constant-complexity fat polyhedra in $\mathbb{R}^3$. Our algorithm runs in $O((1/\beta^3)n \log^3 n)$ time, improving the result of Agarwal, Katz, and Sharir [2] by two logarithmic factors. Like their algorithm, our algorithm unfortunately does not detect cyclic overlap. Hence, we also study the problem of verifying a given depth order. We give an algorithm that checks in $O((1/\beta^2)n \log^3 n)$ time[2] whether a given ordering for a set of fat convex polyhedra is a valid depth order. This is the first result for this problem. Until now, the only algorithm for verifying a given depth order was an algorithm for arbitrary triangles [11], which runs in $O(n^{4/3+\varepsilon})$ time.

2. Preliminaries. In this section we introduce some basic definitions and terminology.

For a three-dimensional object o, we use $vol(o)$ to denote the volume of o and we use $proj(o)$ to denote the vertical projection of o onto the xy-plane. For a two-dimensional object, we use $area(o)$ to denote its area. We use the following definition of fatness [9].

DEFINITION 1. *An object o in $\mathbb{R}^d$ is defined to be β-fat if for any ball b whose center lies in o and that does not fully contain o, we have $vol(b \cap o) \geq \beta \cdot vol(b)$.*

(For convex objects—the case considered in this paper—this definition is equivalent, up to constant factors, to other definitions of fatness that have been proposed.) It is not hard to show that the projection of a fat object is also fat, as proved by De Berg [6] and made precise in the following lemma.

LEMMA 1 (see [6]). *If an object P is a β-fat object in three dimensions, then $proj(P)$ has fatness $\Omega(\beta)$ in two dimensions.*

Define the *size* of an object o, denoted by $size(o)$ to be the radius of its smallest enclosing ball. Note that the size of a ball is simply its radius.

Finally, we need a result that will allow us to stab a set of relatively large fat objects that all intersect some region R using only a few points. Similar results have been proved earlier [7].

LEMMA 2. *Let R be a bounded region in the plane, and let c be a constant with $0 < c \leq 1$. Then there is a collection Q of $O(1/(c\beta)^2)$ points with the following property: any β-fat object o with $size(o) \geq c \cdot size(R)$ that intersects R contains at least one point from Q.*

Proof. Let U be a bounding square of R, and let $\widehat{U}$ be the square twice the size of U and with the same center. Consider a β-fat object o with $size(o) \geq c \cdot size(R)$ that intersects R. Then $area(o \cap \widehat{U}) \geq c'c\beta \cdot area(\widehat{U})$ for a suitable constant c' (cf. Van der Stappen's thesis [25, Theorem 2.9]). Hence, a regular grid on $\widehat{U}$ with $\lceil M \rceil^2$ cells, where $M = 2/(c'c\beta)$, must have at least one grid point inside P, because the area of any convex object missing all grid points is less than $2 \cdot area(\widehat{U})/M$. ☐

The following lemma was proved in a more general setting by Van Kreveld [22].

[1]Though results are presented in terms of fat polyhedra, all our results also work in the more general setting of objects that project to fat polygons.

[2]This is an improvement over the $O(n \log^4 n)$ bound that we had in the preliminary version of the paper [8], which was published in SODA 2006.

However, his definition of fatness is different from ours. Therefore we have proved it independently using our definition. The proof can be found in the appendix. In the lemma below, an α-fat triangle refers to a triangle all of whose angles are at least α. (Such a triangle is α'-fat, according to Definition 1, for some $\alpha' = \Omega(\alpha)$.)

LEMMA 3. *Let P be a β-fat convex polygon with n vertices. There is a set T of α-fat triangles that cover P, where $|T| = O(n)$ and $\alpha = \Theta(\beta)$.*

We will also need the following lemma.

LEMMA 4. *Let P_1 and P_2 be simple polygons. Let e_1 be an edge of P_1 and e_2 be an edge of P_2. If P_1 intersects P_2 so that there is no vertex of P_1 inside P_2 and no vertex of P_2 inside P_1, then there is an intersection of edges e_3 of P_1 and e_4 of P_2 such that $e_3 \neq e_1$ and $e_4 \neq e_2$.*

Proof. Let e of P_1 and e' of P_2 be edges that intersect. If $e \neq e_1$ and $e' \neq e_2$, then we are done. If $e \neq e_1$ and $e' = e_2$, then there must be an intersection between e and a different edge of P_2 (since there are no vertices of P_1 inside P_2), meaning that we have found an intersection between e and some edge $e'' \neq e_2$, and we are done. Similarly, we are done if $e = e_1$ and $e' \neq e_2$. Finally, suppose $e = e_1$ and $e' = e_2$. Since e_1 enters P_2, it must exit it, and that implies that there must be an intersection between e_1 and some edge $e'' \neq e_2$. This puts us in the previous case, so we are done. □

3. Vertical ray shooting.

Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a collection of n constant-complexity convex β-fat polyhedra that we wish to preprocess for vertical ray shooting. We start by studying the simpler case where all the objects are intersected by a common vertical line. After that we will show how to use this structure to obtain an efficient solution to the general problem.

Agarwal, Katz, and Sharir [2] already described a data structure for the case where all objects are intersected by a common vertical line and project to triangles. We observe that it is possible to apply fractional cascading to their structure to obtain the following result.

LEMMA 5. *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n disjoint convex constant-complexity β-fat polyhedra that are all stabbed by a vertical line ℓ and that all project to fat triangles. Then there is a data structure such that vertical ray shooting queries on $\mathcal{P}$ can be answered in $O(\log n)$ time. The structure uses $O((1/\beta)n \log n)$ storage and can be built in $O((1/\beta)n \log n)$ time.*

Proof. As stated above, all we need to do is apply fractional cascading to the structure of Agarwal, Katz, and Sharir [2]. For completeness, we briefly describe their solution and explain how to apply fractional cascading.

The structure is a balanced binary tree $\mathcal{T}$ with the objects in the leaves, sorted by their position along ℓ; the lowest object is in the leftmost leaf, the second lowest object in the next leaf, and so on. Since the objects are nonintersecting and convex, this ordering is well defined.

For a node ν, let $\mathcal{P}(\nu)$ denote the set of objects stored in the leaves of the subtree rooted at ν. At each nonleaf node ν of $\mathcal{T}$, we store the union $U(\nu)$ of the projections of the objects in $\mathcal{P}(\nu)$. We preprocess $U(\nu)$ for point-enclosure queries, that is, queries that ask whether a point q in the xy-plane lies inside $U(\nu)$, as follows. Let p_ℓ be the point where ℓ intersects the xy-plane. Then all projections contain p_ℓ, and since they are convex, $U(\nu)$ is star-shaped with p_ℓ in the kernel. Hence, if we partition the plane into cones by drawing half-lines from p_ℓ through all breakpoints on the boundary of $U(\nu)$, then a point-enclosure query can be answered in $O(1)$ time after we have determined in which cone the query point lies.

To perform a query with a vertical ray starting above all objects, we walk down the tree as follows. Suppose we reach a node ν. When the point p where the ray hits the xy-plane lies inside the union of the right child of ν, we proceed to the right child; otherwise we proceed to the left child. The leaf we reach must store the first object hit (if any object is hit at all). When the starting point of the ray does not lie above all objects, things are more complicated. However, Agarwal, Katz, and Sharir have shown that a query can still be answered by walking down the tree, although now up to four nodes per level may be visited. In any case, we visit $O(\log n)$ nodes in total, and at each node we have to do a point-enclosure query. As explained above, a point-enclosure query can be answered in $O(1)$ time after we have determined in which cone the query point lies. Finding the right cone can be done in $O(\log n)$ time by binary search, but this can be made faster: using fractional cascading [13, 14], finding the cones can be done in $O(1)$ time, except for the search at the root. Since the application of fractional cascading is completely standard in this setting, we omit further details.

To build the structure, we sort the objects along ℓ in $O(n \log n)$ time, and then we construct the unions to be stored at each node in a bottom-up fashion. Hence, when we arrive at a node ν, we have to merge the two unions of the children of ν. Because the unions are star-shaped with respect to the same point, computing the union of these unions boils down to merging the two circularly sorted lists of breakpoints. Hence, this can be done in linear time. The total time to construct all unions is therefore equal to the total size of the data structure, which is $\sum_\nu O(|\mathcal{P}(\nu)|) = O(n \log n)$. Adding the additional pointers for the fractional cascading does not increase the preprocessing time or the amount of storage asymptotically. $\square$

Now consider the general case, where the objects in $\mathcal{P}$ are not necessarily stabbed by a vertical line. We can cover each object by $O(1)$ subobjects whose projections are fat triangles using the technique of Lemma 3, so we can assume without loss of generality that all objects project to fat triangles. We shall make use of balanced aspect ratio trees (BAR-trees). *BAR-trees* are a special type of binary space partition (BSP) trees for point sets. They were introduced by Duncan, Goodrich, and Kobourov [15, 16], who showed that BAR-trees have excellent performance for approximate range searching and approximate nearest-neighbor searching. A BSP tree $\mathcal{T}$ for a set S of points contained in some bounding square σ is a recursive partitioning of σ by splitting lines, such that the final cells of the subdivision do not contain any points in their interior. Each node ν of $\mathcal{T}$ corresponds to a region $\mathit{region}(\nu) \subset \sigma$, which is defined recursively as follows. The region $\mathit{region}(\mathit{root}(\mathcal{T}))$ is the whole square σ. Furthermore, if the splitting line stored at a node ν is $\ell(\nu)$, then $\mathit{region}(\mathit{leftchild}(\nu)) = \mathit{region}(\nu) \cap \ell(\nu)^-$, where $\ell(\nu)^-$ is the half-plane below $\ell(\nu)$. Similarly, $\mathit{region}(\mathit{rightchild}(\nu)) = \mathit{region}(\nu) \cap \ell(\nu)^+$, where $\ell(\nu)^+$ is the half-plane above $\ell(\nu)$.

The special properties of BAR-trees that are relevant for us are the following. First, a BAR-tree on a set S of points has depth $O(\log |S|)$ and size $O(|S|)$. Furthermore, the regions corresponding to a node in a BAR-tree have bounded aspect ratio, which implies they are c-fat for some constant c. It has been shown by De Berg and Streppel [12] that this implies the following.

LEMMA 6 (see [12]). *Let o be a β-fat object. Then there is a set $G(o)$ of* 12 *points—we call these points* guards*—such that for any BAR-tree region R that intersects o but does not contain a guard from $G(o)$ in its interior we have $\mathit{size}(o) = \Omega(\mathit{size}(R))$.*

De Berg and Streppel [12] used this to design a so-called object BAR-tree: this is a BAR-tree that can be used for approximate range searching in a set of objects rather than in a point set. Our ray-shooting structure combines BAR-trees and the lemma above in a different way, as described next.

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n constant-complexity β-fat polyhedra. Let $G_i = G(\mathit{proj}(P_i))$ be a set of guards for the projection of P_i, as in Lemma 6. Our data structure for vertical ray shooting on $\mathcal{P}$ is defined as follows.

- The main tree $\mathcal{T}$ is a BAR-tree for the set $G = G_1 \cup \cdots \cup G_n$.
- Let ν be a node in $\mathcal{T}$. We say that an object P_i is *large* at ν if (i) $\mathit{proj}(P_i)$ intersects $\mathit{region}(\nu)$, and (ii) $\mathit{region}(\mathit{parent}(\nu))$ contains a guard from G_i in its interior, but $\mathit{region}(\nu)$ does not. Note that Lemma 6 implies that $\mathit{size}(P_i) = \Omega(\mathit{size}(\mathit{region}(\nu)))$ if P_i is large at ν. Let $\mathcal{P}(\nu) \subset \mathcal{P}$ be the subset of objects that are large at ν.

  Let $Q(\nu)$ be a set of points such that for any $P_i \in \mathcal{P}(\nu)$, there is a point $q \in Q(\nu)$ with $q \in \mathit{proj}(P_i)$. By Lemma 2 there exists such a set $Q(\nu)$ of size $O(1/\beta^2)$. Assign each object $P_i \in \mathcal{P}(\nu)$ arbitrarily to one of the points $q \in Q(\nu)$ contained in its projection. Let $\mathcal{P}(q)$ denote the set of objects assigned to q. We store the set $\mathcal{P}(q)$ in a data structure $\mathcal{T}(q)$ for vertical ray shooting according to Lemma 5. Thus each node ν has $|Q(\nu)|$ associated structures.

Let us first see how to answer a vertical ray shooting query with this structure.

LEMMA 7. *A vertical ray-shooting query can be answered in $O((1/\beta^2)\log^2 n)$ time.*

Proof. Let p be the point where the line through the query ray intersects the xy-plane. Search with p down the tree $\mathcal{T}$. At every node ν on the search path, perform a query in the associated structure $\mathcal{T}(q)$ of each $q \in Q(\nu)$. A query thus takes $O(\log n \cdot \log n \cdot (1/\beta^2))$ time, that is, the length of every search path, times the query time in the associated data structures along the search path, times the size of $Q(\nu)$.

To prove the correctness, it suffices to argue that any object P_i whose projection contains p must be large at one of the nodes on the search path of p. To see this, we observe that $\mathit{region}(\mathit{root}(\mathcal{T}))$ contains all guards from G_i, while the leaf regions do not contain any guards in their interior. It follows that when we follow the path of p, the object P_i must become large at some node. $\square$

We can now prove our final result on vertical ray shooting.

THEOREM 1. *Let $\mathcal{P}$ be a collection of n convex disjoint constant-complexity β-fat polyhedra in $\mathbb{R}^3$. Then there is a data structure such that vertical ray-shooting queries on $\mathcal{P}$ can be answered in $O((1/\beta^2)\log^2 n)$ time. The structure uses $O((1/\beta)n\log^2 n)$ storage and can be built in $O((1/\beta)n\log^2 n)$ time.*

Proof. The correctness of the query procedure and the query time have been shown in Lemma 7.

It remains to prove the bound on the construction time; the storage bound then follows trivially. Computing the guards for each object takes constant time per object, and constructing the BAR-tree takes $O(n \log n)$ time [16]. We claim that an object P_i is large at $O(\log n)$ nodes. Indeed, any guard is contained in the regions of the nodes on a single path down the tree, and an object can be large at a node only if the parent region contains one of its guards. Hence, $\sum_\nu |\mathcal{P}(\nu)| = O(n \log n)$. We can generate the sets $\mathcal{P}(\nu)$ in $O(n \log n)$ time by filtering the objects down the tree $\mathcal{T}$. The set $Q(\nu)$ can be constructed in $O(|Q(\nu)|)$ time, and associating the objects with the points in $Q(\nu)$ can be done in a brute-force way in $O(|Q(\nu)| \cdot |\mathcal{P}(\nu)|)$. Finally, constructing the associated structures of ν takes time

$$\sum_{q \in Q(\nu)} O((1/\beta)|\mathcal{P}(q)| \log |\mathcal{P}(q)|) \quad = \quad O((1/\beta)|\mathcal{P}(\nu)| \log |\mathcal{P}(\nu)|)$$

by Lemma 5. Hence, the overall construction time is

$$\begin{aligned}&\textstyle\sum_\nu O(|\mathcal{P}(\nu)| \cdot (|Q(\nu)| + (1/\beta)\log|\mathcal{P}(\nu)|))\\ &= O((1/\beta)n\log^2 n + (1/\beta^2)n\log n)\\ &= O((1/\beta)n\log^2 n). \qquad \square\end{aligned}$$

4. The size of the transitive reduction of depth-order graphs. Let $\mathcal{P} = \{P_1,\dots,P_n\}$ be a set of disjoint objects in $\mathbb{R}^3$. Recall that we say that P_i is below P_j, denoted by $P_i \prec P_j$, if there are points $(x,y,z_i) \in P_i$ and $(x,y,z_j) \in P_j$ with $z_i < z_j$. We define the *depth-order graph of* $\mathcal{P}$ to be the graph $\mathcal{G}(\mathcal{P}) = (\mathcal{P}, E)$, where $(P_i,P_j) \in E$ if and only if $P_i \prec P_j$. Hence, a depth order for $\mathcal{P}$ corresponds to a topological order on $\mathcal{G}(\mathcal{P})$.

In general it is too costly to compute $\mathcal{G}(\mathcal{P})$ explicitly, since it can have $\Omega(n^2)$ arcs. When computing depth orders for segments in the plane, this can be circumvented by looking only at pairs of segments that "see" each other, that is, that can be connected vertically without crossing another segment. For objects in 3-space, however, the number of pairs that see each other can be quadratic, even when the objects are fat. In this section we therefore study the size of the transitive reduction of depth-order graphs, since the transitive reduction is the smallest subgraph that is sufficient to topologically sort a graph. The main result is that the number of arcs in the transitive reduction of the depth-order graph of a set of fat objects is linear. Then in the next section we will compute a superset of the arcs in the transitive reduction.

We define the *separation* of two nodes in the depth-order graph, denoted $sep(P_i, P_j)$, to be the length of the longest path from P_i to P_j. Notice that if the graph contains cycles, $sep(P_i,P_j)$ can be infinite. We define $\mathcal{G}^{(1)}(\mathcal{P}) = (\mathcal{P}, E^{(1)})$ to be the subgraph of the depth-order graph $\mathcal{G}(\mathcal{P})$, where $(P_i,P_j) \in E^{(1)}$, if and only if $sep(P_i,P_j) = 1$ in $\mathcal{G}(\mathcal{P})$. In other words, $(P_i,P_j) \in E^{(1)}$ if there exists a vertical line that intersects both objects, and every such line does not intersect any other object between P_i and P_j.

LEMMA 8. *If $\mathcal{G}(\mathcal{P})$ is acyclic, the transitive closure of $\mathcal{G}^{(1)}(\mathcal{P})$ is the transitive closure of $\mathcal{G}(\mathcal{P})$.*

Proof. We have to prove that there is a path $P_i \rightsquigarrow P_j$ in $\mathcal{G}(\mathcal{P})$ if and only if there is a path $P_i \rightsquigarrow P_j$ in $\mathcal{G}^{(1)}(\mathcal{P})$. The "if" part is obvious since $\mathcal{G}^{(1)}(\mathcal{P})$ is a subgraph of $\mathcal{G}(\mathcal{P})$. We prove the "only if" part by induction on $sep(P_i,P_j)$.

If $sep(P_i,P_j) = 1$, the arc (P_i,P_j) exists in $\mathcal{G}^{(1)}(\mathcal{P})$ by construction. Now assume there is a path in $\mathcal{G}^{(1)}(\mathcal{P})$ between all nodes with separation m. Take P_i and P_j in $\mathcal{G}(\mathcal{P})$ which have separation $m+1$. Then there is a node x such that $sep(P_i,x) = 1$ and $sep(x,P_j) = m$. By the induction hypothesis, we then have a path $P_i \to x \rightsquigarrow P_j$ in $\mathcal{G}^{(1)}(\mathcal{P})$. $\square$

For arbitrary triangles in 3-space, the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ can still be $\Theta(n^2)$. For some special classes of objects, however, the number of arcs is linear. For example, one can show that this number is linear for a set of disjoint polyhedra whose projections form a set of polygonal pseudodisks [10]. Here we concentrate on the case where the objects in the given set $\mathcal{P}$ project onto fat convex objects. We show that in this case the number of arcs is also linear. Since fat convex objects project to fat objects, showing this also shows that the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ is small if the input is a set of fat objects. We start with an auxiliary lemma.

LEMMA 9. *Let $P_i \in \mathcal{P}$ be an object and let $\mathcal{P}^+(i)$ be the subset of objects $P_j \in \mathcal{P}$ that are above P_i and where $sep(P_i,P_j) = 1$. Then the projections $proj(P_j)$ of the objects $P_j \in \mathcal{P}^+(i)$ are pairwise disjoint.*

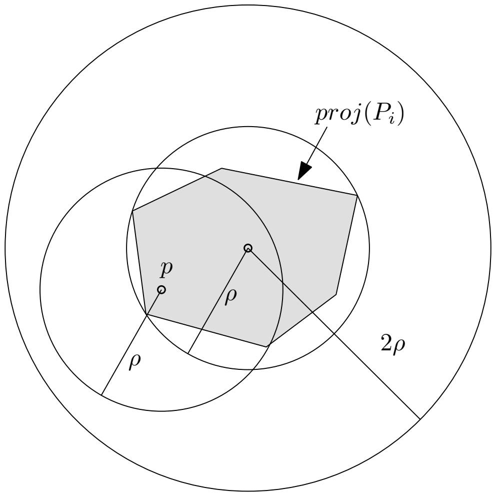

FIG. 1. *Illustration of the packing argument.*

Proof. Suppose the projections are not pairwise disjoint. Then there are objects $P_j, P_k \in \mathcal{P}^+(i)$ such that $proj(P_j) \cap proj(P_k) \neq \emptyset$ and $sep(P_i, P_j) = 1$ and $sep(P_i, P_k) = 1$. Since $proj(P_j)$ and $proj(P_k)$ intersect, they must share at least one point, so there must be an arc between P_j and P_k in $\mathcal{G}(\mathcal{P})$. Therefore, either $sep(P_i, P_j) > 1$ or $sep(P_i, P_k) > 1$, either case being a contradiction. ∎

THEOREM 2. *Let $\mathcal{P}$ be a collection of n disjoint objects in $\mathbb{R}^3$ that project to convex β-fat objects. Then the number of edges in $\mathcal{G}^{(1)}(\mathcal{P})$ is $O(n/\beta)$.*

Proof. We will charge each arc in $\mathcal{G}^{(1)}(\mathcal{P})$ to an object and then use a packing argument to show that the number of arcs in $\mathcal{G}^{(1)}(\mathcal{P})$ charged to each object is $O(1/\beta)$.

We project all objects onto the xy-plane, making them convex fat objects. In this setting, we say that one object is above another if the original objects satisfy this relationship.

Recall that for a planar object o, its size is defined as the radius of its smallest enclosing disk. Consider an arc (P_i, P_j) in $\mathcal{G}^{(1)}(\mathcal{P})$. We charge the arc to the smaller of the two objects. That is, we charge the arc to P_i if $size(proj(P_i)) < size(proj(P_j))$ and to P_j otherwise. We claim that any object is charged $O(1/\beta)$ arcs. To prove this, take an arbitrary object P_j such that (P_i, P_j) is charged to P_i. Let $\rho = size(proj(P_i))$. If there is an arc in $\mathcal{G}^{(1)}(\mathcal{P})$ between P_i and P_j, then $proj(P_j)$ intersects $proj(P_i)$. Let p be a point in this intersection. Then a circle centered at p with radius ρ is centered in $proj(P_j)$ and does not fully enclose $proj(P_j)$, or else $proj(P_j)$ would have a smallest enclosing circle that is smaller than or equal to that of $proj(P_i)$. Thus, this circle contains at least $\beta\pi\rho^2$ units of area of $proj(P_j)$ by the definition of fatness. Also, this circle is completely enclosed in a circle of radius 2ρ centered at the center of the smallest enclosing disk of $proj(P_i)$. This is illustrated in Figure 1.

Since all objects $proj(P_j)$ where P_j is above P_i and $sep(P_i, P_j) = 1$ must be disjoint by Lemma 9, and because each must have at least $\beta\pi\rho^2$ units of area inside a disk that has $4\pi\rho^2$ units of area, there can be only $4/\beta$ edges of $\mathcal{G}^{(1)}(\mathcal{P})$ charged to P_i. We must double this number to account for objects P_j below P_i such that (P_j, P_i) is charged to P_i. Therefore, we get an upper bound on the number of arcs charged to P_i of $8/\beta$. Finally, since there are n objects, $\mathcal{G}^{(1)}(\mathcal{P})$ can have at most $8n/\beta$ edges, which is $O(n/\beta)$. ∎

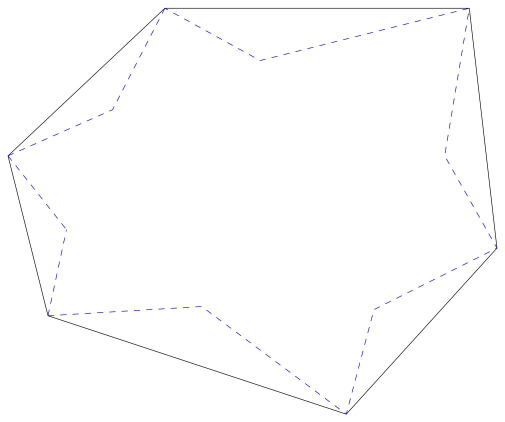

FIG. 2. *A projection of a polyhedron with witness edges added.*

5. Computing depth orders. We now present the algorithm for finding the depth order of a set $\mathcal{P} = \{P_1, \dots, P_n\}$ of n disjoint β-fat convex polyhedra. In contrast to the proof of Theorem 2, we require the complexity of the projection of each object to be constant.

Witness edges. One of the basic steps that we need to perform repeatedly in our algorithm will be to find polyhedra that are above a query polyhedron. To facilitate this, we will add so-called *witness edges* inside the projection of each P_i. They are defined as follows.

Let β' be defined so that each member of $\{proj(P_i)|P_i \in \mathcal{P}\}$ is β'-fat. By Lemma 1 we know that $\beta' = \Omega(\beta)$. Also let $\mathcal{C} = \{0, \alpha, 2\alpha, \dots, c\alpha\}$, where $\alpha = (\beta'\pi)/8$ and $c = \lfloor 2\pi/\alpha \rfloor$. We call the directions in $\mathcal{C}$ *canonical directions.* We require the witness edges to have the following properties. Let W_i and W_j be the sets of witness edges constructed for P_i and P_j, respectively.

(i) Each witness edge has one of the canonical directions.

(ii) For any pair of polyhedra P_i and P_j, we have that $proj(P_i)$ intersects $proj(P_j)$ if and only if at least one of the following is true:
  - A vertex of $proj(P_i)$ is inside $proj(P_j)$, or a vertex of $proj(P_j)$ is inside $proj(P_i)$.
  - A witness edge in W_i crosses a witness edge in W_j.

The construction of the set W_i of witness edges for P_i is done as follows. For each edge $e = vw$ of $proj(P_i)$ we add to W_i two witness edges e' and e'' that are incident to v and w, respectively, extend into the interior of P_i, and form a triangle with e. The directions of the witness are chosen from the canonical directions, such that the angles that e' and e'' make with e are minimal; see Figure 2. We claim that if we add the witness edges in this manner, they have the required properties. The first property holds by construction, so it remains to prove the second property. We first argue that the witness edges lie completely inside $proj(P_i)$, which implies that the "if" part of the second property holds.

LEMMA 10. *The witness edges in W_i lie completely inside $proj(P_i)$.*

Proof. Let e be the edge for which we are adding witness edges. Let p be the midpoint of e and consider the circle C with center p and diameter equal to the length of e. Suppose an edge of $proj(P_i)$ intersects the region bounded by e' and e''. Note that this region must be inside the lighter region in Figure 3 by the minimal-angle

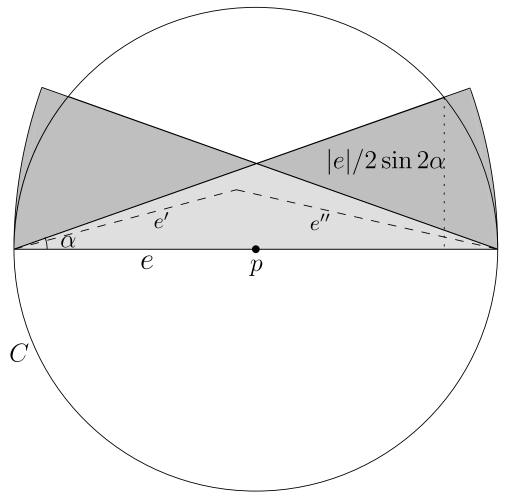

FIG. 3. *No edge of the polygon may enter the light gray region.*

condition which implies that the angles that e makes with e' and e'' are at most α. Then, by convexity of *proj*(P_i), we know that *proj*$(P_i) \cap C$ must be completely inside the union of the triangular wedges in Figure 3. These wedges have area at most $\beta'\pi|e|^2/8$ inside C. Hence, *area*(*proj*$(P_i) \cap C) < \beta'\pi|e|^2/4$, contradicting our assumption that *proj*(P_i) is β'-fat. □

The following lemma, which follows directly from Lemma 4, finishes the proof that the witness edges have the required properties.

LEMMA 11. *If proj(P_i) intersects proj(P_j) and proj(P_i) does not contain a vertex of proj(P_j) or vice versa, then a witness edge from P_i intersects a witness edge from P_j.*

The algorithm. The general idea of our algorithm is as follows. By Lemma 8 it is sufficient to find all pairs of objects P_i, P_j of separation 1 in the depth-order graph. Such a pair of objects must, of course, intersect in the projection. Thus, ideally we would like to find among all pairs P_i, P_j whose projections intersect the pairs of separation 1. Our algorithm does not quite achieve this—it will find more pairs, but the number of extra pairs we find will be small. Lemma 11 suggests that the task of finding the intersecting pairs of projections can be broken into two parts: finding pairs for which there is a vertex of the projection of one polyhedron inside the projection of another and finding crossing pairs of witness edges.

Below we give a more detailed description of the algorithm. The algorithm will find a set A of arcs—a superset of the arcs (P_i, P_j) for objects of separation 1—and then topologically sort the graph $\mathcal{G}^* = (\mathcal{P}, A)$. Initially A is empty.

1. For every vertex v of each object $P_i \in \mathcal{P}$, find the objects $P^b(v)$ and $P^a(v)$ that are directly below and above v, respectively, and add the arcs $(P^b(v), P_i)$ and $(P_i, P^a(v))$ to A.
2. Sort the objects by decreasing size so that *size*(*proj*$(P_1)) \geq \cdots \geq$ *size*(*proj* $(P_n))$, and define $\mathcal{S}_i = \{P_1, \ldots, P_i\}$.
3. For every witness edge e associated with each P_i, find a set $\mathcal{P}(e)$ consisting of objects $P_j \in \mathcal{S}_{i-1}$ with the following properties:
   (P1) Each $P_j \in \mathcal{P}(e)$ has a witness edge that intersects e.
   (P2) Each $P_j \in \mathcal{P}(e)$ is above P_i.

(P3) Each $P_j \in \mathcal{S}_{i-1}$ with $sep(P_i, P_j) = 1$ that satisfies (P1) and (P2) is a member of $\mathcal{P}(e)$.

For each P_i, add the set of arcs $\{(P_i, P_j) : P_j \in \mathcal{P}(e)$ and e is a witness edge of $P_i\}$ to A.

4. Repeat step 3 with "below" substituted for "above" and the directions of the arcs added reversed.
5. Topologically sort the graph $\mathcal{G}^* = (\mathcal{P}, A)$ and report the order.

LEMMA 12. *The order reported by the algorithm is a valid depth order for $\mathcal{P}$ if a depth order exists.*

Proof. Assume a depth order exists for $\mathcal{P}$. It follows directly from the construction that every arc added to the set A is also an arc in the depth-order graph $\mathcal{G}(\mathcal{P})$. It remains to argue that A is a superset of the set of arcs in the graph $\mathcal{G}^{(1)}(\mathcal{P})$.

Consider an arc (P_i, P_j) in $\mathcal{G}^{(1)}(\mathcal{P})$. If there is a vertex of $proj(P_i)$ in $proj(P_j)$ (or vice versa) then, because $sep(P_i, P_j) = 1$, that vertex is directly below P_j (resp., above P_i). Hence, the arc is found in step 1. By Lemma 11, the remaining case is that a witness edge of $proj(P_i)$ intersects a witness edge from $proj(P_j)$. Without loss of generality, assume P_i is smaller than P_j. Hence, $P_j \in \mathcal{S}_{i-1}$. Since (P_i, P_j) is an arc in $\mathcal{G}^{(1)}(\mathcal{P})$, $sep(P_i, P_j) = 1$. By condition (P3), the arc will be found in step 3 or 4, depending on whether P_j is above or below P_i. $\square$

Step 1 can be carried out efficiently using the ray-shooting data structure presented in the previous section. Hence, it remains to describe step 3 in more detail. This step will be performed as follows. We will treat each $P_2, \ldots, P_n$ in order. When we have to handle P_i, we will make sure we have a data structure available that we can query with each witness edge e of P_i and that will then report the set $\mathcal{P}(e)$. After having queried with all witness edges of P_i, we insert P_i into the data structure and proceed with P_{i+1}. Next we describe this data structure.

The witness-edge-intersection data structure. Consider the set of all witness edges of the objects in $\mathcal{P}_{i-1}$. These witness edges have canonical directions, so we can partition them into $|\mathcal{C}|$ subsets depending on their directions. The query segment e has one of the canonical directions as well. Hence, we construct for each subset $|\mathcal{C}|$ different data structures, one for each query direction. We now describe the structure for one such subset, call it W, and a fixed query direction.

Assume without loss of generality that the witness edges in W are all horizontal, and that the query edge e is vertical. The structure is a multilevel data structure defined as follows.

- The top level of the data structure is a segment tree $\mathcal{T}$ on the projections of the edges in W onto the x-axis. Note that each node ν in $\mathcal{T}$ corresponds to a vertical slab in the plane.
- Let $W(\nu)$ denote the edges in W whose projection is in the canonical subset of ν. Such an edge crosses the slab of ν but not the slab of the parent of ν. We store the edges in $W(\nu)$ in a balanced binary tree $\mathcal{T}(\nu)$, ordered according to their y-coordinates. We call this the "slab tree." So far our structure is just a standard two-level tree to perform intersection queries with vertical segments in a set of horizontal segments in the plane [10].
- Let μ be a node in $\mathcal{T}(\nu)$. Let $\mathcal{P}(\mu)$ denote the subset of objects that have a witness edge in the subtree rooted at μ. The node μ represents a rectangular[3] region $R(\mu)$ that is bounded by two slab boundaries and the topmost and

[3]This is only true because we assumed the edges in W are horizontal and the query edge is vertical. In general, μ will represent a parallelogram, but this does not influence the arguments.

bottommost edge stored in the subtree rooted at μ. We associate with μ a reduced subset $\overline{\mathcal{P}(\mu)} \subset \mathcal{P}(\mu)$ of the objects, in the following way: $P_j \in \overline{\mathcal{P}(\mu)}$ if and only if $P_j \in \mathcal{P}(\mu)$ and $\mathit{size}(\mathit{proj}(P_j)) \geq \mathit{size}(R(\mu))/2\sqrt{2}$.

By Lemma 2 we can find a set $Q(\mu)$ consisting of $O(1/\beta^2)$ points such that the projection of any object $P_j \in \overline{\mathcal{P}(\mu)}$ is stabbed. We arbitrarily assign each $P_j \in \overline{\mathcal{P}(\mu)}$ to one of the points q it contains, and we associate a balanced binary search tree $\mathcal{T}(q)$ with each point q on the associated objects, where the sorting order is defined by the height of the objects along the vertical line through q.

This finishes the description of the data structure. Next we describe the algorithms to query the structure and to insert an object.

LEMMA 13. *With the structure described above, we can find the set $\mathcal{P}(e)$ referred to in Step 3 of the depth-order algorithm in $O((1/\beta^3)\log^3 n)$ time. Furthermore, the set $\mathcal{P}(e)$ contains $O((1/\beta^3)\log^2 n)$ objects.*

Proof. Recall that we actually have to query $|\mathcal{C}| = O(1/\beta)$ different versions of the structure. We focus on the time spent in one of these structures.

To perform a query with a witness edge e belonging to an object P_i, we search with e in the first two levels of the tree in the standard way. This gives us $O(\log^2 n)$ nodes μ whose subtrees contain exactly those edges that intersect e. At each node μ, we use the trees $\mathcal{T}(q)$ for $q \in Q(\mu)$ to find the lowest object that is above P_i. We can search in $\mathcal{T}(q)$ since P_i is known to intersect all objects in $\mathcal{P}(q)$ in the projection. Hence, at μ, we find $|Q(\mu)|$ objects in $O(|Q(\mu)|\log n)$ time in total. The query time and the bound on the size of $\mathcal{P}(e)$ follow.

It remains to argue that the reported set has the required properties. Properties (P1) and (P2) follow immediately from the definition of the data structure and query algorithm. Furthermore, when we query a tree $\mathcal{T}(q)$ we can indeed restrict our attention to the lowest object that is above P_i, because the other objects P_j will either be below P_i or have $\mathit{sep}(P_i, P_j) > 1$. Hence, to prove (P3) it is sufficient to argue that any P_j satisfying (P1) and (P2) and with $\mathit{sep}(P_i, P_j) = 1$ will be a member of one of the sets $\overline{\mathcal{P}(\mu)}$. We know that the object will be a member of $\mathcal{P}(\mu)$ for a visited node μ.

Suppose for a contradiction that $P_j \notin \overline{\mathcal{P}(\mu)}$. This means that we must have $\mathit{size}(\mathit{proj}(P_j)) < \mathit{size}(R(\mu))/2\sqrt{2}$. This can happen only when $\mathit{size}(\mathit{proj}(P_j))$ is less than $d/2$, where d is the distance between the top and bottom edges of $R(\mu)$, because P_j crosses the slab of which $R(\mu)$ is a part. On the other hand, when we reach a node μ in the slab tree by querying with a witness edge e of $\mathcal{P}_i$, we have $\mathit{size}(\mathit{proj}(P_i)) \geq \text{length}(e)/2 \geq d/2$. This contradicts that when we query with a witness edge e of P_i, all objects P_j in the data structure have $\mathit{size}(\mathit{proj}(P_j)) \geq \mathit{size}(\mathit{proj}(P_i))$. ∎

LEMMA 14. *An object P_i can be inserted in $O((1/\beta)\log^2 n(\log n + 1/\beta^2))$ time into the structure.*

Proof. Each of the $O(1)$ witness edges of P_i has to be inserted into $|\mathcal{C}| = O(1/\beta)$ structures. To insert a witness edge, we first find each node μ in a slab tree whose canonical subset contains the witness edge. We test if $\mathit{size}(P_j) \geq \mathit{size}(R(\mu))/2$ and, if so, find a point $q \in Q(\mu)$ that is contained in $\mathit{proj}(P_i)$ and insert P_i into the tree $\mathcal{T}(q)$. This takes $O(\log^2 n(\log n + 1/\beta^2))$ time per structure, thus $O((1/\beta)\log^2 n(\log n + 1/\beta^2))$ time in total. ∎

From Lemmas 13 and 14, we see that steps 3 and 4 of the depth-order algorithm can be performed in $O((1/\beta^3)n\log^3 n)$ time in total. We get the following theorem.

THEOREM 3. *Let $\mathcal{P}$ be a collection of n disjoint constant-complexity β-fat convex polyhedra in $\mathbb{R}^3$. Then we can compute a depth order for $\mathcal{P}$ in time $O((1/\beta^3)n\log^3 n)$ if it exists.*

6. Verifying depth orders. In order for our algorithm to be complete, it should output the correct depth order if one exists, but it should also not output an incorrect depth order if no depth order exists. Unfortunately the algorithm of the previous section does not necessarily detect cycles in the $\prec$-relation. Hence, we present an algorithm for checking whether a given order is correct.

We use the general approach by De Berg, Overmars, and Schwarzkopf [11] for verifying depth orders. Let $L = P_1, \ldots, P_n$ be the given order. Define $L_1 = \{P_1, \ldots, P_{\lfloor n/2 \rfloor}\}$ and $L_2 = \{P_{\lfloor n/2 \rfloor+1}, \ldots, P_n\}$. We first check if any object in L_1 is above any object in L_2. Clearly, if the answer is "yes," then the given ordering is not valid. Otherwise, we verify the lists L_1 and L_2 recursively. If $T(\beta, n)$ is the amount of time to check the objects in L_1 against those in L_2, then the overall algorithm takes $O(T(\beta, n)\log n)$ time. We shall see that $T(\beta, n) = O((1/\beta^2)n\log^2 n)$, so the algorithm for verifying the depth order takes $O((1/\beta^2)n\log^3 n)$ time. Next we describe how to check the objects in L_1 against those in L_2.

First we introduce a new type of witness edge. The difference from the witness edges in section 5 is that the new witness edges will have canonical directions in three dimensions, rather than in the projection. In order to achieve this we need the following lemma from Aronov, De Berg, and Gray [4].

LEMMA 15 (see [4]). *Let $\sigma := \lceil 54\sqrt{3}/\beta \rceil$. For any convex β-fat object o in $\mathbb{R}^3$, there exist concentric axis-aligned cubes $C^-(o)$ and $C^+(o)$ with $C^-(o) \subseteq o \subseteq C^+(o)$ such that*

$$\frac{size(C^+(o))}{size(C^-(o))} = \sigma\,.$$

Assume we are given $C^-(o)$ and $C^+(o)$ for object o. We partition each face of $C^+(o)$ into squares with side length equal to the side length of $C^-(o)$. For each facet f of $C^-(o)$ and each square on the corresponding facet of $C^+(o)$, we sweep f so that it coincides with the square; see Figure 4(i). The sweeping directions form the set of canonical directions. There are at most σ^2 different directions that a facet of $C^-(o)$ can be swept in, so we have $O(1/\beta^2)$ canonical directions. We denote an arbitrary member of this set of directions by $\vec{d}$. Note that the set of canonical directions thus obtained does not depend on o, only on the value σ, which is specified by the fatness factor β.

For each P_i we construct a set W_i of witness edges as follows. First, we add the edges of $C^-(P_i)$ to W_i. Second, for each silhouette vertex v of P_i (a silhouette vertex is a vertex whose projection is a boundary vertex of the projection of P_i), we add an edge e_v that connects v to one of the facets of $C^-(P_i)$. This edge is allowed to be in any of the canonical directions as long as it reaches a facet of $C^-(P_i)$. We can be certain that at least one direction works for v since there must be at least one pair consisting of a facet f of $C^-(P_i)$ and a square on a facet of $C^+(P_i)$ such that v is hit when sweeping f to that square.

We also add some vertices to P_i that we call *witness vertices* as follows (see Figure 4(ii)). For each witness edge e, we add the intersection point of $proj(e)$ and $\partial\, proj(C^-(P_i))$, lifted back to e, to the set of witness vertices for P_i. Moreover, if the projected witness edges of two consecutive silhouette vertices intersect, then we lift

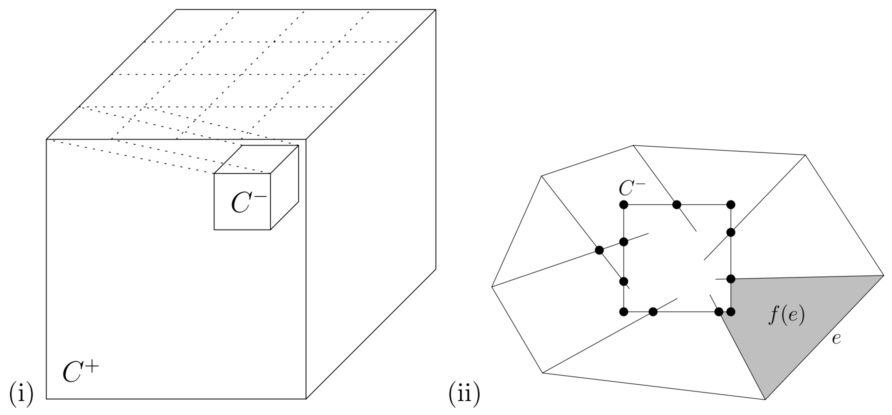

FIG. 4. (i) *One of the canonical directions.* (ii) *Projection of the new witness edges and witness vertices.*

those intersection points to either of the witness edges and make the resulting point a witness vertex. Finally, we add the vertices of $C^-(P_i)$ to the set of witness vertices.

LEMMA 16. *The witness edges satisfy the following properties.*

(i) *Each witness edge has one of the canonical directions.*

(ii) *For any pair of polyhedra P_i and P_j, we have that $proj(P_i)$ intersects $proj(P_j)$ if and only if at least one of the following is true:*
- *A projected witness or silhouette vertex of P_i is inside $proj(P_j)$, or a projected witness or silhouette vertex of P_j is inside $proj(P_i)$.*
- *A projected witness edge in W_i crosses a projected witness edge in W_j.*

Proof. Property (i) is satisfied by construction. Also, if one of the two conditions in property (ii) is satisfied, then the projections of P_i and P_j must intersect since they share a point. Therefore, we will concentrate on proving that a projected witness edge in W_i crosses a projected witness edge in W_j assuming that $proj(P_i) \cap proj(P_j) \neq \emptyset$, and that no projected witness or silhouette vertex of P_i is contained in $proj(P_j)$ (or vice versa).

Since $proj(P_i)$ intersects $proj(P_j)$ and no projected silhouette vertex of one is inside the projection of the other, there must be silhouette edges of $proj(P_i)$ and $proj(P_j)$ that cross. Take one such pair of edges and call them e_i and e_j. Consider the arrangement induced by the projections of the silhouette edges and the witness edges of P_i, and let $f(e_i)$ denote the (bounded) face in this arrangement with e_i on its boundary; see Figure 4(ii). Define $f(e_j)$ similarly for the arrangement induced by the projections of the silhouette edges and the witness edges of P_j. By Lemma 4, there must be an intersection between a pair of edges from $f(e_i)$ and $f(e_j)$, neither of which is $proj(e_i)$ or $proj(e_j)$. Hence, there must be an intersection between two projected witness edges. ∎

It follows from Lemma 16 that there is an object from L_1 below an object from L_2 when at least one of the following conditions holds for some pair P_i, P_j with $P_i \in L_1$ and $P_j \in L_2$: a witness or silhouette vertex of P_i is below P_j, or a witness or silhouette vertex of P_j is above P_i, or a witness edge of P_i is below a witness edge of P_j. To test for the first condition, we construct a data structure for vertical ray shooting on the objects in L_2 and query it with upward rays from the witness and silhouette vertices of the objects in L_1. The second condition can be tested similarly, namely, by constructing a data structure for vertical ray shooting on the objects in L_1 and querying it with downward rays from the witness and silhouette vertices of the objects in L_2. By Theorem 1 and the fact that the total number of witness and silhouette vertices is $O(n)$, this will take $O((1/\beta^2)\log^2 n)$ in total. To test the third condition we proceed as follows. Let $W(L_1)$ and $W(L_2)$ denote the set of all witness edges defined for the objects in L_1 and L_2, respectively. We will preprocess $W(L_2)$ into a data structure for querying with witness edges from $W(L_1)$, according to the following lemma.

LEMMA 17. *We can preprocess the set $W(L_2)$ in $O((1/\beta^2)n\log n)$ time into a data structure of size $O((1/\beta^2)n\log n)$ such that, for any witness edge $e \in W(L_1)$, we can determine in $O((1/\beta^2)\log^2 n)$ time whether any witness edge from $W(L_2)$ is above e.*

Proof. Let $W_{\vec{d}}(L_2) \subset W(L_2)$ denote the subset of witness edges of canonical direction $\vec{d}$. Note that

$$\sum_{\vec{d}} |W_{\vec{d}}(L_2)| = |W(L_2)| = O(n).$$

Define $W_{\vec{d}}(L_1)$ similarly. For each pair of directions $\vec{d_1}, \vec{d_2}$ we build a data structure on $W_{\vec{d_1}}(L_2)$ for querying with edges from $W_{\vec{d_2}}(L_1)$. (In fact, the structure can be queried with any segment with direction $\vec{d_2}$.) Assume without loss of generality that $\vec{d_1}$ is parallel to the x-axis and $\vec{d_2}$ is parallel to the y-axis. The structure is a multi-level data structure similar to the structure of section 5. The first two levels are exactly the same as for the structure in section 5: the first level is a segment tree on the x-ranges of the witness edges, and the second level is a balanced binary search tree on their y-values (in section 5 this was called the slab tree). For each canonical subset of a node in the slab tree, we store its highest witness edge. Note that the concept of "highest" is well defined since the witness edges in the canonical subset all have the same direction and the query edge will have a fixed direction as well.

A query with a witness edge $e \in W_{\vec{d_2}}(L_1)$ can be answered in $O(\log^2 n)$ time, as follows: query with the x-coordinate of e in the segment tree; for each node ν on the path, query with the y-range of e in the associated slab tree $\mathcal{T}(\nu)$; and for each selected node μ in $\mathcal{T}(\nu)$, check if the witness stored there is above e.

When we are querying with an edge e, we actually have to query in the sets $W_{\vec{d}}(L_2)$ for each canonical direction $\vec{d}$. Since there are $O(1/\beta^2)$ canonical directions, this means that the total query time is $O((1/\beta^2)\log^2 n)$.

Building the structure on $W_{\vec{d_1}}(L_2)$ for a given query direction $\vec{d_2}$ can be done in $O(|W_{\vec{d_1}}(L_2)|\log|W_{\vec{d_1}}(L_2)|)$ time. This is because the associated structures of the segment tree (the slab trees) can be built in linear time if we presort the witness edges by y-coordinate. After that we compute the highest edge for each node in a slab tree in a bottom-up fashion (the highest edge for a node is the higher of the highest edges of its two children) in linear time. Hence, the overall preprocessing time is the same as the amount of storage, which is $O(|W_{\vec{d_1}}(L_2)|\log|W_{\vec{d_1}}(L_2)|)$. Overall, the preprocessing is therefore

$$\begin{aligned}
&\textstyle\sum_{\vec{d_1},\vec{d_2}} O(|W_{\vec{d_1}}(L_2)|\log|W_{\vec{d_1}}(L_2)|)\\
=\ & O(1/\beta^2)\cdot \textstyle\sum_{\vec{d_1}} O(|W_{\vec{d_1}}(L_2)|\log|W_{\vec{d_1}}(L_2)|)\\
=\ & O((1/\beta^2)n\log n). \qquad \square
\end{aligned}$$

Putting everything together, we get the following theorem.

THEOREM 4. *We can verify whether a given order on a set of n disjoint convex constant-complexity β-fat polyhedra in $\mathbb{R}^3$ is a valid depth order in $O((1/\beta^2)n\log^3 n)$ time.*

7. Concluding remarks.

We have presented new and improved solutions to three problems on fat convex polyhedra in 3-space: vertical ray shooting, computing depth orders, and verifying depth orders. One open problem is to see if the results can be extended to fat nonconvex polyhedra or fat curved objects.

Our algorithm for verifying depth orders uses a collection of witness edges that have canonical directions in three dimensions and allow us to capture (together with a certain set of points in the objects) the above-below relation between the objects. It would be interesting to investigate if these witness edges can be useful for other problems on convex fat objects as well.

Appendix.

This appendix contains a proof that was omitted.

LEMMA 3. *Let P be a β-fat convex polygon with n vertices. There is a set T of α-fat triangles that cover P where $|T| = O(n)$ and $\alpha \geq \beta/128$.*

Proof. Recall that for triangles, we use the definition that the fatness is given by the smallest angle in the triangle.

Let S be the largest possible square contained in P. Any convex subset of P that contains all of S is at least β'-fat, where $\beta' = \Theta(\beta)$, by Lemma 19 below.

We extend the edges of S until they intersect P and add vertices to P at the intersection points (see Figure 5). We let P_a denote the part of P above the (extended) top edge of S, let P_b denote the part below the bottom edge of S, let P_c denote the part to the right of the right edge of S, and let P_d denote the part to the left of the left edge of S. We will show how to cover P_a. The three other parts of P are covered similarly, and S is covered with two triangles that each have a fatness of 45°.

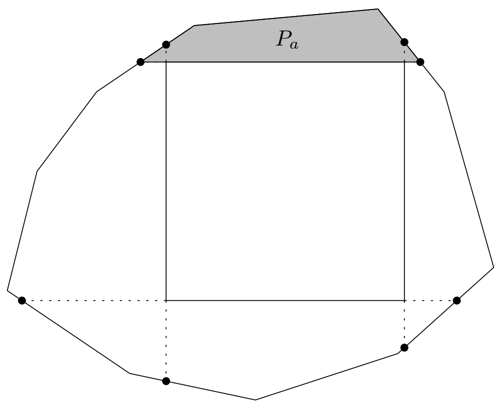

FIG. 5. *One of the subpolygons of P induced by S.*

An ear of a polygon P consists of two consecutive edges of P that have the property that a straight edge connecting the first and last vertex of the edges stays completely inside the polygon. In a convex polygon, any two consecutive edges are ears.

We cover P_a by choosing an arbitrary ear from it (except any ear that also contains the top edge of S), covering it using Lemma 18, and then replacing P by P with that ear removed. Since no part of S is ever removed, P remains fat. Thus we keep removing ears from P_a until it exactly coincides with the extended edge of S.

Since we cover the ears that we remove using the procedure from Lemma 18, we add a constant number of triangles to T per vertex, implying that $|T| = O(n)$. The exact bound on α is given by combining Lemmas 18 and 19. $\square$

LEMMA 18. *An ear of a β-fat polygon P can be covered with at most four α-fat triangles that all stay inside P where $\alpha = (\beta\pi)/16$.*

Proof. In a convex polygon, an ear is a triangle formed by three consecutive vertices. Consider the ear defined by vertices v_{i-1}, v_i, and v_{i+1}. Let ϕ_{i-1}, ϕ_i, and ϕ_{i+1} be the angles at the respective vertices—see Figure 6(i). Because P is β-fat, we know that the angle between any two adjacent edges of P, and in particular the angle ϕ_i, is at least $\beta/(2\pi)$. There are three possibilities for the other two angles, ϕ_{i-1} and ϕ_{i+1}: either they are both at least α, they are both less than 2α, or one is larger than 2α and one is smaller than α. Note that these cases overlap.

Case (i). $\phi_{i-1} \geq \alpha$ and $\phi_{i+1} \geq \alpha$. In this case, the ear is trivial to cover: it is already an α-fat triangle that can be covered by a copy of itself.

Case (ii). $\phi_{i-1} < 2\alpha$ and $\phi_{i+1} < 2\alpha$. First, we add triangles to the edges $v_{i-1}v_i$ and v_iv_{i+1} where the angles of the edges of the triangles with respect to the boundary edges are at least 2α. By Lemma 10, these triangles must stay inside P as long as $\alpha \leq (\beta\pi)/16$. However, it is clear that the nonboundary vertex of these triangles must be outside the ear that we are covering. Therefore, we can place a triangle at the middle vertex of the ear with two sides that correspond to the sides of the two triangles that we just added and whose third side is the edge of the ear that goes between these two edges. This triangle completes the covering of the ear.

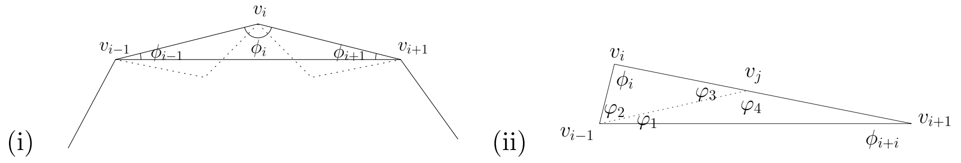

FIG. 6. (i) *Case* (ii). (ii) *Case* (iii).

Case (iii). $\phi_{i-1} > 2\alpha$ and $\phi_{i+1} < \alpha$ (or the symmetric case). See Figure 6(ii). In this case, we add an edge between the vertex that is at the large angle (v_{i-1}, without loss of generality) and the edge across from it, making vertex v_j. This splits ϕ_i into two angles φ_1 and φ_2. We place v_j such that φ_1 is exactly α. Thus, $\varphi_3 = \alpha + \phi_{i+1} > \alpha$. By assumption, $\varphi_2 > \alpha$. Thus, we can cover the triangle $v_{i-1}v_iv_j$ with a copy of itself. Triangle $v_{i-1}v_jv_{i+1}$ can be covered according to the procedure outlined for Case (ii) above. Note that in all cases, we have covered the ear with at most four α-fat triangles. $\square$

LEMMA 19. *Let P be a convex β-fat polygon in $\mathbb{R}^2$ and S be the largest square contained in P. Then any convex subset P' such that $S \subseteq P' \subseteq P$ is β'-fat where $\beta' \geq \beta/(8\pi)$.*

Proof. By the results of section 3.2.1 of Van der Stappen's thesis [25], the side length of S is at least $\beta\rho/(2\sqrt{2})$, where ρ is the diameter of P.

Let $d = \overline{p_1p_2}$ be the diameter of P'. Let $S' \subseteq S$ be the largest square contained in S that has an edge parallel to d. The side length of S' is at least $\sqrt{2}/2$ times the side length of S. Let p_3 and p_4 denote the midpoints of the sides of S' parallel to d; see Figure 7.

We will make two triangles: $p_1p_3p_4$ and $p_2p_3p_4$. By convexity, both of these triangles must be completely inside P'. The sum of the area of these triangles is not

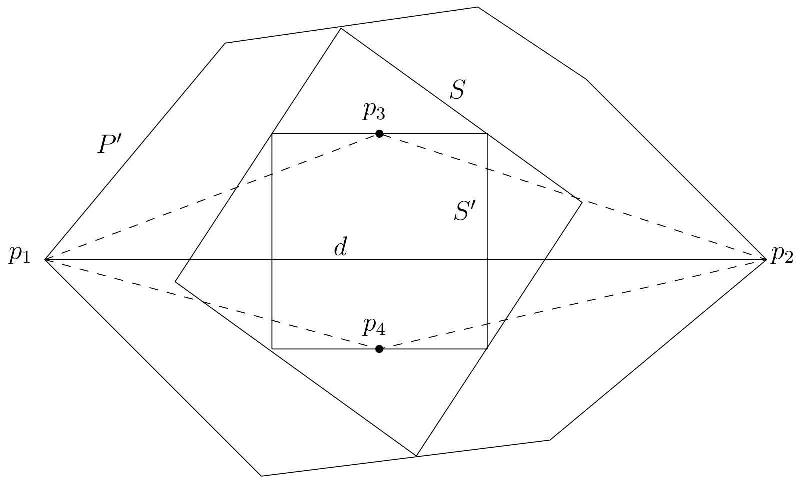

FIG. 7. *P' must be fat.*

dependent on the placement of S'—it is always $d \cdot s/2$, where s is the side length of S'.

Since P' is convex, the fatness of P' is determined by a circle placed at p_1 with radius d [25]. The area of that circle is πd^2. Thus the fatness of P' is

$$\beta' = \frac{\frac{d \cdot s}{2}}{\pi d^2} = \frac{s}{2d\pi} \geq \frac{\beta\rho}{8d\pi} \geq \frac{\beta}{8\pi}$$

since $d \leq \rho$. □

REFERENCES

[1] P. K. AGARWAL, M. DE BERG, D. HALPERIN, AND M. SHARIR, *Efficient generation of k-directional assembly sequences*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1996, pp. 122–131.

[2] P. K. AGARWAL, M. J. KATZ, AND M. SHARIR, *Computing depth orders for fat objects and related problems*, Comput. Geom., 5 (1995), pp. 187–206.

[3] P. K. AGARWAL AND J. MATOUŠEK, *On range-searching with semi-algebraic sets*, Discrete Comput. Geom., 11 (1993), pp. 393–418.

[4] B. ARONOV, M. DE BERG, AND C. GRAY, *Ray shooting and intersection searching amidst fat convex polyhedra in 3-space*, in Proceedings of the 22nd Annual ACM Symposium on Computational Geometry, 2006, pp. 88–94.

[5] M. DE BERG, *Ray Shooting, Depth Orders and Hidden Surface Removal,* Lecture Notes in Comput. Sci. 703, Springer-Verlag, Berlin, 1993.

[6] M. DE BERG, *Vertical ray shooting for fat objects*, in Proceedings of the 21st Annual ACM Symposium on Computational Geometry, 2005, pp. 288–295.

[7] M. DE BERG, H. DAVID, M. J. KATZ, M. OVERMARS, A. F. VAN DER STAPPEN, AND J. VLEUGELS, *Guarding scenes against invasive hypercubes*, Comput. Geom., 26 (2003), pp. 99–117.

[8] M. DE BERG AND C. GRAY, *Vertical ray shooting and computing depth orders for fat objects*, in Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2006, pp. 494–503.

[9] M. DE BERG, M. KATZ, F. VAN DER STAPPEN, AND J. VLEUGELS, *Realistic input models for geometric algorithms*, Algorithmica, 34 (2002), pp. 81–97.

[10] M. DE BERG, M. VAN KREVELD, M. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[11] M. DE BERG, M. OVERMARS, AND O. SCHWARZKOPF, *Computing and verifying depth orders*, SIAM J. Comput., 23 (1994), pp. 437–446.

[12] M. de Berg and M. Streppel, *Approximate range searching using binary space partitions*, in Proceedings of the IARCS Conference on Foundation Software Technology and Theoretical Computational Science (FSTTCS), 2004, pp. 110–121.

[13] B. Chazelle and L. J. Guibas, *Fractional cascading*: I. *A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.

[14] B. Chazelle and L. J. Guibas, *Fractional cascading*: II. *Applications*, Algorithmica, 1 (1986), pp. 163–191.

[15] C. A. Duncan, *Balanced Aspect Ratio Trees,* Ph.D. thesis, The Johns Hopkins University, Baltimore, MD, 1999.

[16] C. A. Duncan, M. T. Goodrich, and S. Kobourov, *Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 300–309.

[17] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir, *Dynamic data structures for fat objects and their applications*, Comput. Geom., 15 (2000), pp. 215–227.

[18] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics. Principles and Practice*, 2nd ed., Addison-Wesley, Bonn, Germany, 1990.

[19] M. J. Katz, 3-*D vertical ray shooting and* 2-*D point enclosure, range searching, and arc shooting amidst convex fat objects*, Comput. Geom., 8 (1998), pp. 299–316.

[20] M. J. Katz, *personal communication*, 2005.

[21] M. J. Katz, M. Overmars, and M. Sharir, *Efficient hidden surface removal for objects with small union size*, Comput. Geom., 2 (1992), pp. 223–234.

[22] M. van Kreveld, *On fat partitioning, fat covering and the union size of polygons*, Comput. Geom., 9 (1998), pp. 197–210.

[23] M. Pellegrini, *Ray shooting on triangles in* 3-*space*, Algorithmica, 9 (1993), pp. 471–494.

[24] M. Pellegrini, *Ray shooting and lines in space*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC, Boca Raton, FL, 1997, pp. 599–614.

[25] A. F. van der Stappen, *Motion Planning Amidst Fat Obstacles*, Ph.D. thesis, Utrecht University, Utrecht, The Netherlands, 1994.

[26] A. F. van der Stappen, D. Halperin, and M. H. Overmars, *The complexity of the free space for a robot moving amidst fat obstacles*, Comput. Geom., 3 (1993), pp. 353–373.