

Vertical Ray Shooting and Computing Depth Orders for Fat Objects

Mark de Berg¹

Chris Gray¹

¹Department of Computing Science, TU Eindhoven. P.O. Box 513, 5600 MB Eindhoven, the Netherlands.
Email: {mdberg,cgray}@win.tue.nl. This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

Abstract

We present new results for three problems dealing with a set \mathcal{P} of n constant-complexity fat objects 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 $O(n \log^4 n)$ time whether a given order on \mathcal{P} is a valid depth order. All three results improve on previous results.

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 algorithms 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 is $\Theta(n^f)$. 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* [7]. 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 proven to be quite successful: the free space for a not-too-large robot moving amidst a set of fat obstacles has only linear complexity [24], which has enabled the development of motion-planning algorithms with near-linear running times in this setting [23].

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 .

The first problem we study is the *ray-shooting problem*: preprocess the set \mathcal{P} such that ray-shooting queries—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, both in computer graphics and in computational geometry; the book by De Berg [4] and the survey by Pellegrini [22] discuss many of the (computational-geometry) solutions. Like 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 [4, 21], 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 $O(\sqrt{n})$ [4].

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 [17] 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 3D polytopes as well.) Using the techniques of Efrat *et al.* [15] it is possible to improve the storage bound to $O(n \log^2 n)$ and the query time to $O(\log^3 n)$ [18]. Recently De Berg [5] presented a structure with $O(\log^2 n)$ query time; his structures 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 [16] or to do hidden-surface removal with the algorithm of Katz *et al.* [19]. 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 [9]. This has lead researchers to also study this problem for fat objects. Agarwal *et al.* [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 [17] 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. In other words, 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 constant-complexity fat polyhedra¹ in \mathbb{R}^3 . Our structure uses $O(n \log^2 n)$ storage and has $O(\log^2 n)$ query time. Compared to Katz’s structure [18] it has a better query time (while the storage is the same) and compared to the De Berg’s structure [5] 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 constant-complexity fat polyhedra in \mathbb{R}^3 . Our algorithm runs in $O(n \log^3 n)$ time, thus improving the result of Agarwal *et al.* [2] by two logarithmic factors. Like the algorithm of Agarwal *et al.*, 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 $O(n \log^4 n)$ time whether a given ordering for a set of fat 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 [9], which runs in $O(n^{4/3+\varepsilon})$ time.

2 Preliminaries

In this section we introduce some basic definitions and terminology.

For a 3-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 2-dimensional object, we use $area(o)$ to denote its area. We use the following definition of fatness [7].

Definition 2.1. *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 made precise in the following lemma.

Lemma 2.2. [5] *If P_i is a β -fat object in three dimensions, then $proj(P_i)$ has fatness $\Omega(\beta)$.*

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.

¹Though 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.

Finally, we will 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 already been proved earlier [6]. For completeness we give a proof in the Appendix.

Lemma 2.3. *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 $\text{size}(o) \geq c \cdot \text{size}(R)$ that intersects R contains at least one point from Q .*

3 Vertical ray shooting

Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a collection of n constant-complexity 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 *et al.* [2] already described a data structure for the case where all objects are intersected by a common vertical line and project to fat triangles. We observe that it is possible to apply fractional cascading to their structure to obtain the following result. A description of the structure of Agarwal *et al.* and how to apply fractional cascading to it is given in the Appendix.

Lemma 3.1. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n disjoint 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 it can be built in $O((1/\beta)n \log n)$ time.*

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 Van Kreveld [20], so we can assume without loss of generality that all objects project to fat triangles. We shall make use of BAR-trees. *BAR-trees* (or *balanced aspect ratio trees*) are a special type of BSP trees for point sets. They were introduced by Duncan *et al.* [13, 14], 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 $\text{region}(\nu) \subset \sigma$, which is defined recursively as follows. The region $\text{region}(\text{root}(\mathcal{T}))$ is the whole square σ . Furthermore, if the splitting line stored at a node ν is $\ell(\nu)$, then $\text{region}(\text{leftchild}(\nu)) = \text{region}(\nu) \cap \ell(\nu)^-$, where $\ell(\nu)^-$ is the half-plane below $\ell(\nu)$. Similarly, $\text{region}(\text{rightchild}(\nu)) = \text{region}(\nu) \cap \ell(\nu)^+$, 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 [10] that this implies the following.

Lemma 3.2. [10] *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 $\text{size}(o) = \Omega(\text{size}(R))$.*

De Berg and Streppel [10] 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 $G_i := G(\text{proj}(P_i))$ be a set of guards for the projection of P_i , as in Lemma 3.2. 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 $G := G_1 \cup \dots \cup G_n$.
- Let ν be a node in \mathcal{T} . We say that an object P_i is *large* at ν if (i) $proj(P_i)$ intersects $region(\nu)$, and (ii) $region(parent(\nu))$ contains a guard from G_i in its interior but $region(\nu)$ does not. Note that Lemma 3.2 implies that $size(P_i) = \Omega(size(region(\nu)))$ 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 $q \in proj(P_i)$. By Lemma 2.3 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 3.1. Thus each node ν has $|Q(\nu)|$ associated structures.

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

Lemma 3.3. *A vertical ray-shooting query can be answered in $O((1/\beta^2) \cdot \log^2 n)$ 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 $q \in Q(\nu)$. Since the depth of tree is $O(\log n)$, a query in an associated structure takes $O(\log n)$ time by Lemma 3.1, and $|Q(\nu)| = O(1/\beta^2)$, the query time follows. 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 $region(root(\mathcal{T}))$ 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 3.4. *Let \mathcal{P} be a collection of n 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 $O((1/\beta^2) \cdot \log^2 n)$ time. The structure uses $O((1/\beta)n \log^2 n)$ storage and it 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 3.3.

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 [14]. 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 only be large at a node 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

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

by Lemma 3.1. Hence, the overall construction time is

$$\sum_{\nu} O(|\mathcal{P}(\nu)| \cdot (|Q(\nu)| + (1/\beta) \log |\mathcal{P}(\nu)|)) = O((1/\beta)n \log^2 n + (1/\beta^2)n \log n) = O((1/\beta)n \log^2 n).$$

\square

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 $(P_i, P_j) \in E$ iff $P_i \prec P_j$. 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 segment in the plane, this can be circumvented by only looking 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. Unfortunately, finding the transitive reduction is difficult. Indeed, in the next section we will compute a superset of the arcs in the transitive reduction. Nevertheless, the results obtained in this section give some interesting insights that will help us later on.

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 $\mathcal{G}(\mathcal{P})$.

Lemma 4.1. *If $\mathcal{G}(\mathcal{P})$ is acyclic, then 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 $P_i \rightarrow x \rightsquigarrow P_j$ 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. Here we concentrate on the case where the objects in the given set \mathcal{P} project onto fat convex polygons. We show that in this case the number of arcs is also linear. Since fat convex polyhedra project to fat polygons, 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 polyhedra. We start with an auxiliary lemma (proved in the Appendix).

Lemma 4.2. *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.*

Theorem 4.3. *Let \mathcal{P} be a collection of n disjoint objects in \mathbb{R}^3 that project to β -fat polygons. 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 fat polygons. In this setting, we say that one object is above another if the original objects satisfy this relationship.

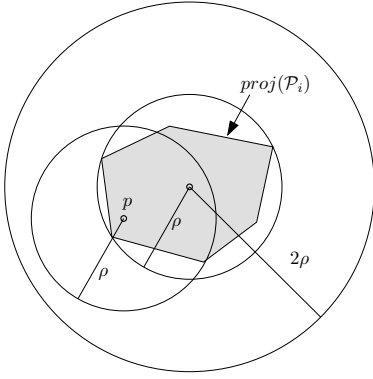


Figure 1: Illustration of the packing argument.

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 $\text{size}(P_i) < \text{size}(P_j)$ 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 $\rho := \text{size}(\text{proj}(P_i))$. If there is an arc in $\mathcal{G}^{(1)}(\mathcal{P})$ between P_i and P_j , then $\text{proj}(P_j)$ intersects $\text{proj}(P_i)$. Let p be a point in this intersection. Then a circle centered at p with radius ρ is centered in $\text{proj}(P_j)$ and does not fully enclose $\text{proj}(P_j)$, or else $\text{proj}(P_j)$ would have a smallest enclosing circle that is smaller or equal to that of $\text{proj}(P_i)$. Thus, this circle contains at least $\beta\pi\rho^2$ units of area of $\text{proj}(P_j)$ by the definition of fatness. Also, it is completely enclosed in a circle of radius 2ρ centered at the center of the smallest enclosing disk of $\text{proj}(P_i)$. This is illustrated in Figure 1.

Since all polygons $\text{proj}(P_j)$, where P_j is above P_i and $\text{sep}(P_i, P_j) = 1$, must be disjoint by Lemma 4.2, 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 only be $4/\beta$ 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 $O(n/\beta)$. \square

5 Computing depth orders

We now present the algorithm to find 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 4.3, 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 $\{\text{proj}(P_i) | P_i \in \mathcal{P}\}$ is β' -fat. By Lemma 2.2, 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 be the set of witness edges constructed for P_i .

- (i) Each witness edge has one of the canonical directions.
- (ii) For any pair of polyhedra P_i and P_j , we have that $\text{proj}(P_i)$ intersects $\text{proj}(P_j)$ if and only if at least one of the following is true:
 - A vertex of $\text{proj}(P_i)$ is inside $\text{proj}(P_j)$, or a vertex of $\text{proj}(P_j)$ is inside $\text{proj}(P_i)$.
 - 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 $\text{proj}(P_i)$ 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 (with the proof in the Appendix) that the witness edges lie completely inside $\text{proj}(P_i)$, which implies that the “if”-part of the second property holds.

Lemma 5.1. *The witness edges in W_i lie completely inside $proj(P_i)$.*

The following lemma finishes the proof that the witness edges have the required properties.

Lemma 5.2. *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 intersects a witness edge from Q .*

Proof. 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 there must be a pair of intersecting edges, e_i from $proj(P_i)$ and e_j from $proj(P_j)$. Let T_i (resp. T_j) be the triangles formed by e_i (resp. e_j) and the witness edges added for e_i (resp. e_j). By Lemma 5.1, T_i is contained within $proj(P_i)$ and T_j is contained within $proj(P_j)$. Since there is no vertex of $proj(P_i)$ in $proj(P_j)$, there cannot be an endpoint of e_i in T_j . Likewise, there cannot be an endpoint of e_j in T_i . This implies that a witness edge from T_i intersects a witness edge from T_j . \square

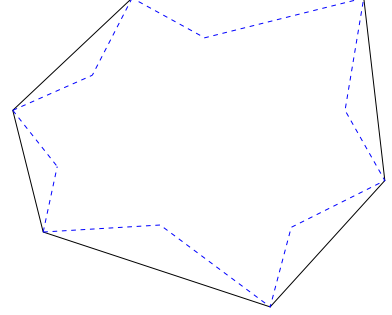


Figure 2: A projection of a polyhedron with witness edges added

The algorithm. The general idea of our algorithm is as follows. By Lemma 4.1 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 ones 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 5.2 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 a part that finds 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 $size(P_1) \geq \dots \geq size(P_n)$, and define $\mathcal{S}_i := \{P_1, \dots, P_i\}$.
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 $\{(P_i, P_j) : P_j \in \mathcal{P}(e) \text{ and } e \text{ is a witness edge of } P_i\}$ to A .

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

Lemma 5.3. *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. 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, \dots, 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, let's 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 multi-level 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 segment in the plane [8].
- 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 rectangular² region $R(\mu)$ that is bounded by two slab boundaries and the topmost and 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: $P_j \in \overline{\mathcal{P}(\mu)}$ iff $P_j \in \mathcal{P}(\mu)$ and $size(P_j) \geq size(R(\mu))/2\sqrt{2}$.

By Lemma 2.3 we can find a set $Q(\mu)$ consisting of $O(1/\beta^2)$ points such 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 perform a query in the structure and to insert an object.

²This 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.

Lemma 5.4. *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 $\text{sep}(P_i, P_j) > 1$. Hence, to prove (P3) it is sufficient to argue that any P_j satisfying (P1) and (P2) and with $\text{sep}(P_i, P_j) = 1$ 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 $P_j \notin \overline{\mathcal{P}(\mu)}$. Then $\text{size}(P_j) < \text{size}(R(\mu))/2\sqrt{2}$. This can only happen when $\text{size}(P_j)$ is less than $d/2$, where d is the distance between the top and bottom edge 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 $\text{size}(P_i) \geq \text{length}(e)/2 \geq d/2$. This contradicts that when we query with a witness edge e of P_i , all objects P_j in the data structure have $\text{size}(P_j) \geq \text{size}(P_i)$. \square

Lemma 5.5. *An object P_i can be inserted into the structure in $O((1/\beta) \log^2 n (\log n + 1/\beta^2))$ time.*

From the two lemmas above, 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 5.6. *Let \mathcal{P} be a collection of n disjoint constant-complexity β -fat 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 *et al.* [9] for verifying depth orders. Let $L = P_1, \dots, P_n$ be the given order. Define $L_1 = P_1, \dots, P_{\lfloor n/2 \rfloor}$ and $L_2 = P_{\lfloor n/2 \rfloor + 1}, \dots, P_n$. The algorithm constructs a data structure on L_2 that can answer the following queries: is a query object in $P_i \in L_1$ above any of the objects in L_2 ? We query the structure with each object in L_1 . Clearly, if the answer to any of the queries is “yes” then the given ordering is not valid. Otherwise, we verify the lists L_1 and L_2 recursively.

If $T(n)$ is the amount of time to build the data structure and $Q(n)$ is the query time, then the overall algorithm takes $O((T(n) + nQ(n)) \log n)$ time. We shall see that $T(n) = O((1/\beta)n \log^3 n)$ and that $Q(n) = O((1/\beta) \log^3 n)$, so the algorithm for verifying the depth order takes $O((1/\beta)n \log^4 n)$ time.

Next we describe the data structure.

Once again, we will make use of witness edges. This time, however, we will “lift” the witness edges to 3-space. That is, after constructing the witness edges as in the previous section, we replace each witness edge e of P_i with a segment e' whose projection is e and that lies inside P_i . As before we apply Lemma 5.2, which says that we can treat the vertices of the objects in L_1 separately from the witness edges. Thus, we will create two data structures: one for ray shooting and one for detecting whether a query edge is above any of the witness edges of an object in L_2 . Since the data structure for ray shooting has been described before, we will concentrate on the latter data structure.

As before, we build $|\mathcal{C}| \cdot (|\mathcal{C}| - 1)$ data structures. Consider a subset W of the witness edges whose projections have the same canonical direction, say parallel to the x -axis, and assume we are building a structure for querying with an edge whose projection is parallel to the y -axis. Again, we have a multi-level structure whose first two levels are a segment tree on the x -intervals of the edges of W and a binary tree on the y -values of the edges.

A query on this part of the data structure results in $O(\log^2 n)$ canonical subsets of witness edges whose projections intersect the projection of the query witness edge. The projections of the edges in a canonical subset are all parallel to the x -axis, and the projected query edge is parallel to the y -axis.

Consider a canonical subset S . Since the query edge intersects all edges in S in the projection, we may as well work with the line ℓ through the query edge. Consider the vertical plane h through ℓ . On the plane h we see the line ℓ and a collection of points, which are the intersections of the edges in S with h . If ℓ is above any of the points, then we know that the object from L_1 is above an object from L_2 , and thus that the depth order is incorrect. In order to test this in $O(\log n)$ time, we can dualize (in the standard manner) all the points and ℓ . If the point representing the query edge is below the upper envelope of the arrangement of lines, then the query edge is above one of the points. Otherwise, it is not.

This works fine if the plane h would be the same for every query edge. However, this is not the case— h can be placed anywhere between the left edge of the slab and the right edge of the slab. Let $t \in [0, 1]$ be the parameter representing the placement of h . As we vary t , the cross section is always of a collection of points, moving vertically with a linear dependence on t . This implies that the duals of each of these points are planes in xyt -space. We can find the upper envelope of this arrangement and preprocess it for point location in $O(|S| \log |S|)$ time. Then we can query the canonical subset in $O(\log n)$ time. The total query time is therefore $O((1/\beta) \log^3 n)$, and the preprocessing time is $O((1/\beta)n \log^3 n)$. We get the following theorem.

Theorem 6.1. *We can verify whether a given order on a set of n disjoint constant-complexity β -fat polyhedra in \mathbb{R}^3 is a valid depth order in $O((1/\beta)n \log^4 n)$ time.*

References

- [1] P. Agarwal, M. de Berg, D. Halperin, and M. Sharir. Efficient generation of multi-directional assembly sequences. In *Proc. Symp. on Discrete Algorithms (SODA)*, pages 122–131, 1996.
- [2] P.K. Agarwal, M.J. Katz, and M. Sharir. Computing depth orders for fat objects and related problems. *Comput. Geom. Theory Appl.* 5:187–206 (1995).
- [3] P.K. Agarwal and J. Matoušek. On range-searching with semi-algebraic sets. *Discr. Comput. Geom.* 11: 393–418 (1993).
- [4] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*. LNCS 703, Springer-Verlag, 1993.
- [5] M. de Berg. Vertical Ray Shooting for Fat Objects. In *Proc. 21st ACM Symp. Comput. Geom.*, pages 288–295, 2005.
- [6] 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. Theory Appl.* 26:99–117 (2003).
- [7] M. de Berg, M. Katz, F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica* 34:81–97 (2002).
- [8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications (2nd edition)*. Springer-Verlag, 2000.
- [9] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.* 23:437–446 (1994).
- [10] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. *Proc. IARCS Conf. Found. Software Tech. Theoret. Comput. Sci. (FSTTCS)*, pages 110–121, 2004.
- [11] B. Chazelle and L. J. Guibas. Fractional Cascading: I. A Data Structuring Technique. *Algorithmica* 1:133–162 (1986)
- [12] B. Chazelle and L. J. Guibas. Fractional Cascading: II. Applications. *Algorithmica* 1:163–191 (1986)
- [13] C.A. Duncan, Balanced Aspect Ratio Trees, Ph.D. Thesis, John Hopkins University, 1999.
- [14] C.A. Duncan, M.T. Goodrich, S.G. Kobourov, Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees, In *Proc. 10th Ann. ACM-SIAM Sympos. Discrete Algorithms*, pages 300–309, 1999.
- [15] A. Efrat, M.J. Katz, F. Nielsen, and M. Sharir. Dynamic Data Structures for Fat Objects and Their Applications. *Comput. Geom. Theory Appl.* 15:215–227 (2000).
- [16] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, 1990.
- [17] M.J. Katz. 3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting amidst convex fat objects. *Comput. Geom. Theory Appl.* 8:299–316 (1998).
- [18] M.J. Katz. Personal communication, 2005.

- [19] M.J. Katz, M. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.* 2:223–234 (1992).
- [20] M. van Kreveld. On fat partitioning, fat covering and the union size of polygons. *Comput. Geom. Theory Appl.* 9(4):197–210, (1998).
- [21] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica* 9: 471–494 (1993).
- [22] M.Pellegrini. Ray shooting and lines in space. In: J.E. Goodman and J. O’Rourke (eds.), *Handbook of Discrete and Computational Geometry*, CRC Press, 1997.
- [23] A.F. van der Stappen. *Motion planning amidst fat obstacles*. Ph.D. thesis, Utrecht University, Utrecht, the Netherlands, 1994.
- [24] 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. Theory Appl.* 3:353–373, 1993.

Appendix

This appendix contains some of the proofs that have been omitted.

Lemma 2.3. 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 \hat{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 $area(o \cap \hat{U}) \geq c'c\beta \cdot area(\hat{U})$ for a suitable constant c' (cf. Van der Stappen's thesis [23], Theorem 2.9). Hence, a regular grid on \hat{U} with $\lceil M \rceil^2$ cells, where $M = 2/(c'\beta)$, must have at least one grid point inside P , because the area of any convex object missing all grid points is less than $2 \cdot area(\hat{U})/M$. \square

Lemma 3.1. Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n disjoint 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 $O((1/\beta) \cdot n \log n)$ storage and it can be built in $O((1/\beta) \cdot n \log n)$ time.

Proof. All we need to do is apply fractional cascading to the structure of Agarwal *et al.* [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 non-intersecting, 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 non-leaf 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 are triangles that contain p_ℓ , and since they are convex $U(\nu)$ is star-shaped with p_ℓ in the kernel. Furthermore, the complexity of $U(\nu)$ is $O((1/\beta)n_\nu)$ where n_ν is the number of triangles [2]. 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 *et al.* 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 q lies. Finding the right cone can be done in $O(\log n)$ time by binary search, but this can be sped up: using fractional cascading [11, 12] finding the cones can be done in $O(1)$ time, except 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 list 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 $\sum_{\nu} O((1/\beta) \cdot |\mathcal{P}(\nu)|) = O((1/\beta) \cdot n \log n)$. Adding the additional pointers for the fractional cascading does not increase the preprocessing time or the amount of storage asymptotically. \square

Lemma 4.2. 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.

Proof. Suppose not. Then there are polyhedra $P_j, P_k \in \mathcal{P}^+(i)$ such that $proj(P_j) \cap proj(P_k) \neq \emptyset$. 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. \square

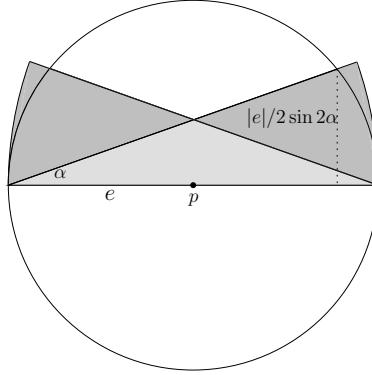


Figure 3: The important region.

Lemma 5.1. 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 $proj(P_i)$ is in the region bounded by e' and e'' . Note that this region must be inside the lighter region in Figure 3 by the minimal-angle condition. 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. \square

Lemma 5.5. An object P_i can be inserted into the structure in $O((1/\beta) \log^2 n (\log n + 1/\beta^2))$ time.

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 $size(P_j) \geq size(R(\mu))/2$ and, if so, find a point $q \in Q(\mu)$ that is contained in $proj(P_i)$ and insert P_i into the tree $\mathcal{T}(q)$. This takes $O(\log^2 n (\log n + 1/\beta^2))$ time per structure, so $O((1/\beta) \log^2 n (\log n + 1/\beta^2))$ time in total. \square