Ray Shooting Amidst Fat Convex Polyhedra in 3-Space

Boris Aronov* Mark de Berg† Chris Gray†

Abstract

We present a data structure for ray-shooting queries in a set of disjoint convex fat polyhedra of total complexity n in $\mathbb{R}^3$. The data structure uses $O(n^{2+\varepsilon})$ storage and preprocessing time, and queries can be answered in $O(\log^2 n)$ time. A trade-off between storage and query time is also possible: for any m with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon})$ storage and preprocessing time such that queries take $O((n/\sqrt{m})\log^2 n)$ time.

1 Introduction

The ray-shooting problem is to preprocess a set $\mathcal{P}$ of objects in $\mathbb{R}^d$ for the following queries: what is the first object (if any) in $\mathcal{P}$ that is hit by a query ray? Such queries form the basis of ray-tracing algorithms, and they can be used to approximate form factors in radiosity methods. Since ray shooting is an integral part of many graphics applications, it should not be surprising that it has received much attention, both in computer graphics and computational geometry. In fact, after the range-searching problem it is probably one of the most widely studied data-structuring problems in computational geometry. The survey by Pellegrini [12] and the book by De Berg [3] discuss several of the data structures that have been developed within computational geometry for the ray-shooting problem (although there is also much work that is not covered there, for example, research concerning ray shooting in 2-dimensional scenes, or in d-dimensional space for $d > 3$). In the discussion below, we will restrict our attention to results on ray shooting in $\mathbb{R}^3$. Furthermore, we focus on the general ray-shooting problem, where the origin and direction of the query ray are unrestricted.

If the set $\mathcal{P}$ consists of n arbitrary triangles, the best known structures with $O(\log n)$ query time use $O(n^{4+\epsilon})$ storage [3, 11], whereas the best structures with near-linear storage have roughly $O(n^{3/4})$ query time [2]. More generally, with $O(m^{1+\varepsilon})$ storage, for any m with $n < m < n^4$, one can obtain $O((n/m^{1/4})\log n)$ query time using $O(m^{1+\varepsilon})$ storage [2]. Better results have been obtained for several special cases. When the set $\mathcal{P}$ is a collection of n axis-parallel boxes, one can achieve $O(\log n)$ query time with a structure using $O(n^{2+\varepsilon})$ storage [3]. Again, a trade-off between query time and storage is possible: with $O(m^{1+\varepsilon})$ storage, for any m with $n < m < n^2$, one can achieve $O((n/\sqrt{m})\log n)$ query time using $O(m^{1+\varepsilon})$ storage. If $\mathcal{P}$ is a set of n balls, then it is possible to obtain $O(n^{2/3})$ query time with $O(n^{1+\varepsilon})$ storage [14], or $O(n^{\varepsilon})$ query time with $O(n^{3+\varepsilon})$ storage [10].

Both axis-parallel boxes and balls are very special objects, and in most graphics applications the scene will not consist of such objects. The question thus becomes: is it possible to improve upon the ray-shooting bounds for arbitrary triangles for classes of objects that are more general than axis-parallel boxes or spheres? This is the problem we tackle in this paper. More precisely, we study the ray-shooting problem for disjoint convex polyhedra that are *fat*—see Section 2 for a formal definition.

Related work. Given the prominence of the ray-shooting problem and the interest in efficient algorithms and data structures for fat objects and other realistic input models in the past decade, it is not surprising that the ray-shooting problem for fat objects has been studied already. The results achieved so far are, however, quite limited. Most of the work on ray shooting among fat objects has dealt with shooting rays in a fixed direction [4, 5, 8]. When it comes to arbitrary rays, there are only a few results. For the case of *horizontal* fat triangles, there is a structure that uses $O(n^{2+\varepsilon})$ storage and has $O(\log n)$ query time [3], but the restriction to horizontal triangles is quite severe. Another related result is by Mitchell *et al.* [9]. In their solution, the amount of storage depends on the so-called *simple-cover complexity* of the scene, and the query time depends on the simple-cover complexity of the query ray. Unfortunately the simple-cover complexity of the ray—and, hence, the worst-case query time—can be $\Theta(n)$ for fat objects. In fact, this can happen even when the input is a set of cubes. The first (and so far only, as far as we know) result that works

*Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840 USA; http://cis.poly.edu/~aronov. Research supported in part by NSF grant ITR-0081964 and by a grant from the US-Israel Binational Science Foundation. Part of the research was performed when B.A. visited TU Eindhoven in June 2005.

†Department 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.

for arbitrary rays and rather arbitrary fat objects was recently obtained by Sharir and Shaul [13]. They present a data structure for ray shooting in a collection of fat triangles that has $O(n^{2/3+\varepsilon})$ query time and uses $O(n^{1+\varepsilon})$ storage. Curiously, their method does not improve the known bounds at the other end of the query-time–storage spectrum, so for logarithmic-time queries the best known storage bound is still $O(n^{4+\varepsilon})$.

Our results. We present a data structure for ray shooting with arbitrary rays in a collection $\mathcal{P}$ of disjoint convex fat polyhedra with n vertices in total. Our structure requires $O(n^{2+\varepsilon})$ storage and has query time $O(\log^2 n)$. A trade-off between storage and query time is also possible: for any m with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon})$ storage and has $O((n/\sqrt{m})\log^2 n)$ query time. Thus we improve upon the result by Sharir and Shaul in two ways: we reduce the query time for near-linear storage from $O(n^{2/3+\varepsilon})$ to $O(\sqrt{n}\log^2 n)$ and improve the bounds at the other end of the spectrum as well.

Of course, the two settings are not the same: Sharir and Shaul consider fat triangles, whereas we consider fat polyhedra. Indeed, our solution makes crucial use of the fact that fat polyhedra have a relatively large volume. Note that neither setting implies the other: fat triangles need not form fat polyhedra, and fat polyhedra do not necessarily have fat facets.

Our solution is based on the following idea. For each polyhedron P we construct a constant number of so-called "towers" that lie inside P and together cover the boundary of P. The towers are in some canonical form, which makes it easy to detect intersections of such a tower with a line segment. We believe that this technique, described in detail in Section 3, is of independent interest, and we expect it will find other applications in problems on fat polyhedra.

2 Preliminaries

Definition and basic properties of fat objects. We will use the definition of fatness introduced by Van der Stappen [15]. For a 3-dimensional object o, we use $\mathrm{vol}(o)$ to denote its volume.

Definition 1 *An object o is β-*fat *if for any ball b whose center lies in o and which does not completely contain o, $\mathrm{vol}(b \cap o) \geq \beta \cdot \mathrm{vol}(b)$.*

It is well known that any fat convex object o admits two concentric balls, one containing o and one contained in o, whose size ratio is bounded. Here we need a similar property, but for cubes instead of balls. For a cube C, let $\mathrm{size}(C)$ be the edge-length of C.

Lemma 1 *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 $\frac{\mathrm{size}(C^+(o))}{\mathrm{size}(C^-(o))} = \sigma$.*

Ray shooting and parametric search. Agarwal and Matoušek [1] described a technique that reduces the ray-shooting problem on a set $\mathcal{P}$ of objects to the segment-emptiness problem, *i.e.*, testing whether a query segment intersects any of the objects in $\mathcal{P}$. Since then their technique has been used in several papers dealing with ray shooting [10, 13, 14]. We will also use this technique. In our setting, it implies the following: if we have a data structure for segment-emptiness queries, then we can use that same structure for ray-shooting queries at the cost of an extra (multiplicative) $O(\log n)$ factor in query time.

3 The data structure

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be the set of convex fat polyhedra that we wish to preprocess for ray-shooting queries. We use n to denote the total number of vertices of the polyhedra. Our global strategy is roughly as follows.

We first present a decomposition of the boundary of each polyhedron into a constant number of pieces that are monotone in some canonical direction. Each such piece is extended into the polyhedron to obtain an object which we will call a *tower*. Next, we present a data structure to efficiently perform segment-emptiness queries on the towers. Using Agarwal and Matoušek's parametric-search technique mentioned above, we then convert this structure into a structure for ray shooting.

The decomposition. We first define the canonical directions that we will use in our decomposition. Let C^+ and C^- be two concentric axis-aligned cubes such that $\frac{\mathrm{size}(C^+)}{\mathrm{size}(C^-)} = \sigma$, where σ is defined as in Lemma 1. Since σ is an integer, we can partition each face of C^+ into σ^2 squares of the same size as the facets of C^-. We use this to define a set $\mathcal{D}$ of $O(1/\beta^2)$ canonical directions, as follows. For each square s on the top facet of C^+, we add to $\mathcal{D}$ the direction into which the top facet of C^- must be translated to make it coincide with s. The remaining five facets of C^+ are treated similarly. The resulting set $\mathcal{D}$ of canonical directions has size $6\sigma^2 = O(1/\beta^2)$.

Next, we define the towers. A *tower in the direction* $\vec{d} \in \mathcal{D}$ is a convex polyhedron t with the following properties:

(i) One of the facets of t is an axis-parallel square; this facet if called the *base* of t and it is denoted by $\mathrm{base}(t)$. We require that the orientation of

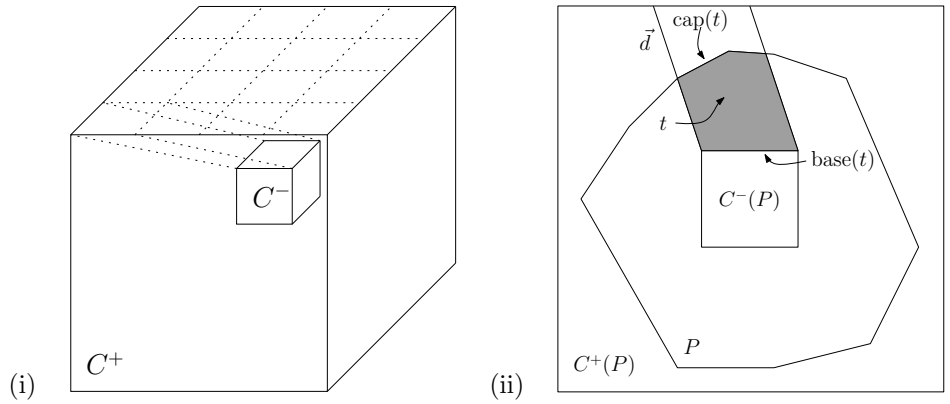

Figure 1: (i) Swept volume defining a tower. (ii) Two-dimensional analogue of a tower t.

the base—whether it is parallel to the xy-plane, to the xz-plane, or to the yz-plane—be uniquely determined by the direction $\vec{d}$. Hence, all towers in a given direction $\vec{d}$ have parallel bases.

(ii) The remaining facets of t form a terrain in direction $\vec{d}$, that is, any line parallel to $\vec{d}$ and intersecting the base intersects the remaining facets either in a single point or in a line segment. We call the union of these remaining facets, excluding facets parallel to $\vec{d}$, the *cap* of the tower, denoted cap(t).

Let $P \in \mathcal{P}$ be a β-fat convex polyhedron. The decomposition of P is performed in a manner similar to the way we constructed the canonical directions. Let $C^-(P)$ and $C^+(P)$ be cubes with the properties given in Lemma 1. Partition each facet of $C^+(P)$ into σ^2 equal-sized squares. For each such square s we construct a tower by sweeping s towards the corresponding facet of $C^-(P)$, and taking the intersection of the swept volume and the polyhedron P. This way we obtain for each polyhedron P one tower for each of the $|\mathcal{D}|$ canonical directions. We denote the set of towers constructed for P by $T(P)$. The union of the towers in $T(P)$ is contained in P; the boundary of this union consists of the boundaries of P and of $C^-(P)$.

Testing for segment emptiness. Before we describe the data structure for segment-emptiness queries, we describe necessary and sufficient conditions for a segment to intersect a polyhedron P. In the lemma below and in the rest of the paper, whenever we speak of "above" and "below" when referring to a specific tower, this is always with respect to the canonical direction $\vec{d}$ of that tower. More precisely, we say that a object o is *below* an object o' whenever there exists a (directed) line with orientation $\vec{d}$ that first intersects o and then o'. A point is inside a tower, for instance, if and only if it is above the base and below the cap. Finally, we use proj(o) to denote the projection of an object o in direction $\vec{d}$ onto a plane orthogonal to $\vec{d}$.

Lemma 2 *A segment $s = \overline{pq}$ intersects a polyhedron $P \in \mathcal{P}$ if and only if one of the following conditions holds:*

1. *p or q is inside P, or*
2. *there is a tower $t \in T(P)$ such that*
   (a) *$\overline{pq}$ intersects* base(t)*, or*
   (b) *$\overline{pq}$ passes below an edge of* cap(t) *and above an edge of* base(t)*, or*
   (c) *$\overline{pq}$ passes below an edge of* cap(t) *and p or q is above* base(t)*.*

Lemma 2 allows us to treat a segment-emptiness query as the disjunction of several different conditions and test separately for each of these conditions. Developing data structures for each of these condition is relatively routine; they can be implemented using standard multi-level range-searching data structures. Below we provide some of the details.

Lemma 3 *Let $\mathcal{P}$ be a set of disjoint β-fat convex polyhedra in $\mathbb{R}^3$ of total complexity n. We can detect whether there is an endpoint of a query segment s inside a polyhedron in $\mathcal{P}$ using a data structure that requires $O(n/\beta)$ storage and preprocessing time and has $O((1/\beta)\log n)$ query time.*

Proof. In order to detect whether a point p is inside a polyhedron in $\mathcal{P}$, we use the so-called *object BAR-tree* designed by De Berg and Streppel [6]. This is a BSP tree with $O(n)$ nodes and depth $O(\log n)$, such that every leaf region intersects at most $O(1/\beta)$ objects. Therefore, assuming the polyhedra have constant complexity, we can test whether p is inside any of the polyhedra in $\mathcal{P}$ simply by finding the cell containing p in $O(\log n)$ time and then testing whether p is inside any of the polyhedra in the cell. If the polyhedra do not have constant complexity, we apply the Dobkin-Kirkpatrick hierarchy [7] to each polyhedron. In either case, the test takes $O(\log n)$ to determine which cell p is in and $O((1/\beta)\log n)$ to test

if p is inside any of the $O(1/\beta)$ polyhedra meeting that cell. □

Lemma 4 *Assuming there is no endpoint of query segment s inside any polyhedron in $\mathcal{P}$, we can detect whether s intersects any polyhedron in $\mathcal{P}$ using a data structure which requires $O(n^{2+\varepsilon}/\beta^2)$ storage and preprocessing time and has query time $O((\log n)/\beta^2)$. Furthermore, for any m with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon}/\beta^2)$ storage and preprocessing time and has $O((n/(\beta^2\sqrt{m}))\log n)$ query time.*

Proof. There are three cases to consider, according to Lemma 2. We will design a different structure for each of them, and in each case we will need a separate structure for each of the $|\mathcal{D}|$ canonical tower directions. Let us fix one of the canonical directions $\vec{d}$, and let $\mathcal{T} = \mathcal{T}(\vec{d})$ be the set of all towers of that direction. Without loss of generality, assume that the base of the towers in $\mathcal{T}$ is parallel to the xy-plane.

Condition 2a: s intersects base(t) *for some tower $t \in \mathcal{T}$:* A segment s intersects base(t) if and only if s intersects base(t) both in the projection onto the yz-plane and in the projection onto the xz-plane. Hence, we can test whether there is an intersected base using a four-level partition tree: the first two levels are used to select the bases that are intersected in the projection onto the xz-plane, and the last two levels are used to test whether any of these bases are also intersected in the projection onto the yz-plane.

Conditions 2b and 2c can be handled similarly. □

Putting it all together. By combining the data structure for segment-emptiness queries described above with Agarwal and Matoušek's parametric search technique [1] mentioned earlier, we obtain our final result.

Theorem 5 *Let $\mathcal{P}$ be a set of β-fat convex and disjoint polyhedra in $\mathbb{R}^3$ of total complexity n. We can preprocess $\mathcal{P}$ using $O(n^{2+\varepsilon}/\beta^2)$ storage and preprocessing time, such that ray-shooting queries can be answered in $O((\log^2 n)/\beta^2)$ time. Moreover, for any m with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon}/\beta^2)$ preprocessing time and storage such that queries take $O((n/\beta^2\sqrt{m})\log^2 n)$ time.*

References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[2] P. K. Agarwal and J. Matoušek. On range-searching with semi-algebraic sets. *Discrete and Computational Geometry*, 11:393–418, 1993.

[3] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal.* Springer-Verlag New York, LNCS 703, 1993.

[4] M. de Berg. Vertical ray shooting for fat objects. In *Proc. 21st Annual Symposium on Computational Geometry*, pages 288–295, 2005.

[5] M. de Berg and C. Gray. Vertical ray shooting and computing depth orders for fat objects. In *Proc. 17th Annual Symposium on Discrete Algorithms*, 2006. To appear.

[6] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 110–121, 2004.

[7] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[8] M. J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry: Theory and Applications*, 8:299–316, 1997.

[9] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *International Journal of Computational Geometry and Applications*, 7(4):317–347, 1997.

[10] S. Mohaban and M. Sharir. Ray shooting amidst spheres in three dimensions and related problems. *SIAM Journal on Computing*, 26(3):654–674, 1997.

[11] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

[12] M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 599–614. CRC Press, Boca Raton-New York, 1997.

[13] M. Sharir and H. Shaul. Ray shooting and stone throwing. In *Proc. 11th European Symposium on Algorithms*, pages 470–481. Springer-Verlag, LNCS 2832, 2003.

[14] M. Sharir and H. Shaul. Ray shooting amid balls, farthest point from a line, and range emptiness queries. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 525–534, 2005.

[15] A. F. van der Stappen. *Motion Planning Amidst Fat Obstacles.* PhD thesis, Dept. of Computer Science, Utrecht University, 1994.