International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

QUERY-SENSITIVE RAY SHOOTING

JOSEPH S. B. MITCHELL[*]
Department of Applied Mathematics and Statistics
University of Stony Brook, Stony Brook, NY 11794–3600, USA

DAVID M. MOUNT[†]
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA

and

SUBHASH SURI[‡]
Deptartment of Computer Science
Washington University, St. Louis, MO 63130–4899, USA

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Ray (segment) shooting is the problem of determining the first intersection between a ray (directed line segment) and a collection of polygonal or polyhedral obstacles. In order to process queries efficiently, the set of obstacle polyhedra is usually preprocessed into a data structure. In this paper we propose a query-sensitive data structure for ray shooting, which means that the performance of our data structure depends on the local geometry of obstacles near the query segment. We measure the complexity of the local geometry near the segment by a parameter called the *simple cover complexity*, denoted by $scc(s)$ for a segment s. Our data structure consists of a subdivision that partitions the space into a collection of polyhedral cells, each of $O(1)$ complexity. We answer a segment shooting query by walking along the segment through the subdivision. Our first result is that, for any fixed dimension d, there exists a simple hierarchical subdivision in which no query segment s intersects more than $O(scc(s))$ cells. Our second result shows that in two dimensions such a subdivision of size $O(n)$ can be constructed in time $O(n \log n)$, where n is the total number of vertices in all the obstacles.

1. Introduction

Ray shooting is the problem of determining the first intersection of a ray with a set of obstacles. This is a fundamental problem in computational geometry because

[*]`jsbm@ams.sunysb.edu`; Partially supported by grants from Boeing Computer Services, Hughes Aircraft, Air Force Office of Scientific Research contract AFOSR–91–0328, and by NSF Grants ECSE–8857642 CCR–9204585, and CCR–9504192.

[†]`mount@cs.umd.edu`; Partially supported by NSF Grant CCR–9310705.

[‡]`suri@cs.wustl.edu`.

of its important applications in areas such as ray tracing[21] and Monte Carlo approaches to radiosity in computer graphics,[33] and motion tracking in virtual reality applications.[23] Since many ray shooting queries may be applied to a given set of obstacles, it is desirable to preprocess the obstacles so that query processing is as efficient as possible. Throughout this paper, we assume that obstacles are modeled by disjoint polyhedra.

1.1. Previous Results

The problem of ray shooting in a simple polygon with n vertices has been studied by Chazelle and Guibas,[16] Chazelle et al.,[15] Goodrich and Tamassia,[22] and Hershberger and Suri.[24] The main result for this special case of the problem is that ray shooting queries can be answered in $O(\log n)$ time with an $O(n)$ size data structure. The recent approaches of Refs. 15,24 are based on constructing triangulations of low "stabbing number".

The problem appears to be significantly more difficult for multiple polygons. All of the known methods for achieving $O(\log n)$ query time require $\Omega(n^2)$ space. The best results known using linear or near-linear space give a query time of roughly $O(\sqrt{n}\log n)$; see Chazelle et al.,[15] and Hershberger and Suri.[24] Several other papers also achieve sublinear query time with sub-quadratic space; Agarwal,[1], Cheng and Janardan,[17,18] and Overmars, Schipper, and Sharir.[29]

Predictably, the quality of the results degrades quickly as the dimension of the underlying space increases beyond two. In three dimensions, the best methods to achieve $O(\log n)$ query time require $O(n^{4+\delta})$ space, for any positive $\delta > 0$.[6,10] Several tradeoffs between space and query time are possible.[3,5,9,10,30,32] In particular, the best known query time with near-linear ($O(n^{1+\delta})$) space is $O(n^{3/4})$ time, for any $\delta > 0$.[4]

In contrast with the two-dimensional case, there are no ray shooting results in dimensions three or higher based on subdivisions of low stabbing number. Indeed, some recent lower bound results of Agarwal, Aronov and Suri[2] indicate that such subdivisions are not possible in higher dimensions. Specifically, they show that there exist (nonconvex) n-vertex polyhedra in three dimensions whose *every* (Steiner) triangulation intersects some ray in $\Omega(n)$ points. More generally, for every $k > 0$, there exists a collection of k disjoint convex polyhedra such that no matter how one triangulates their common exterior, some ray always intersects $\Omega(k + \log n)$ simplices.

1.2. A Query-Sensitive Approach

One of the shortcomings of the type of analysis used in the results above is that it fails to account for the geometric structure of the obstacle set and query ray. Analyses based solely upon input size cannot account for the fact that there exist configurations of obstacles and rays for which solving ray shooting queries is intuitively very easy, and others for which it is quite a bit more complicated. Practical experience suggests that worst-case scenarios are rare for many applications.

By concentrating on worst-case asymptotic performance, algorithm designers may overlook simple approaches that perform very well in the vast majority of practical situations, even though their worst-case performance can be quite poor. One approach for dealing with this phenomenon is to design algorithms for special-purpose classes of inputs, e.g. convex, star-shaped, or monotone polygons. However, these approaches may not be useful if inputs do not satisfy these conditions. An alternative approach is to design algorithms with good expected-case performance. However, there seems to be no useful notion of a "random" polygon or "random" polyhedron that captures the structure existing in many geometric applications.

In this paper we suggest a different approach. We present a fully general algorithm for the ray shooting problem, whose performance is not described as a function of the size of the input, but rather as a function of the complexity of a query with respect to an intrinsic parameter of the input. This parameter is called the *simple cover complexity*, and will be defined shortly. We feel that this parameter intuitively captures the geometric complexity of a given query. As in (Refs. 15,24), we construct a carefully designed subdivision of free space that has low "stabbing number" with respect to that portion of a query segment that lies within free space. This allows query processing to proceed by a simple "walk" through the subdivision.

We focus on a generalization of ray shooting, called *directed segment shooting* (or just *segment shooting*, for short). Given a set of obstacles and a directed line segment $\overrightarrow{pq}$, the problem is to determine the first intersection (if any) with an obstacle for a point traveling along this segment from p to q. If there is no intersection, the point q is returned. The reason for this formulation is that one can imagine applications of segment shooting arising in motion simulation where a number of short segment queries are used to approximate more complex curved motion. Furthermore, in some interference-detection algorithms for virtual reality applications, one tracks the motion of an object through free-space by performing segment-shooting queries (as well as "triangle queries") for each boundary edge and facet of the moving object.[23]

1.3. Defining Cover Complexity

Before defining the complexity parameter referred to above, we give some definitions. Let P denote a finite collection of pairwise-disjoint polygonal obstacles in the plane, or more generally polyhedral obstacles in d-space. Because our algorithms are based on a recursive subdivision of space into sufficiently simple regions, we need to make the *bounded incidence assumption* that each point is incident to at most a constant number β of faces of all dimensions. (A k-face is a boundary flat of dimension k, where $k \leq d-1$.) Define a *ball* of radius r to be the (open) set of points that are less than distance r from a given center point. In the plane, a ball is *simple* if it intersects at most 2 edges of P. In general, in dimension $d \geq 3$, the value 2 can be replaced with any constant δ with $\delta \geq \beta$. Given any $\epsilon > 0$, we say that a ball of radius r is *ϵ-strongly simple* if the ball with the same center and radius $(1+\epsilon)r$ is simple. Given ϵ, a *strongly simple cover* of a domain D is a collection of ϵ-strongly simple balls whose union contains D. Note that the balls themselves cover D, but each expansion by $(1+\epsilon)$ is simple. Finally, the *simple cover complexity* of D *with respect to* ϵ, denoted $scc_\epsilon(D)$, is defined to be the cardinality of the smallest strongly simple cover of D. We will use the term *complexity* for short. When used in asymptotic bounds, we will omit the "ϵ" subscript, since, as will be shown in Lemma 1, it affects the complexity by only a constant factor.

Fig 1(a) shows a strongly simple cover of size 6 for a directed line segment (the heavy line). The ϵ value is 1/4. Solid circles are the balls of the cover, and dashed circles are balls of radius $(1+\epsilon)r$.

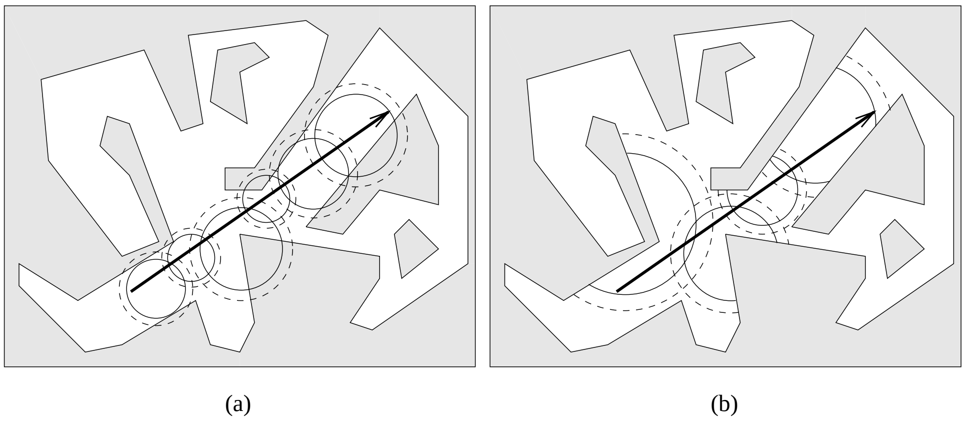


Figure 1: Strongly simple cover complexity and C-complexity.

The reason that we use a $(1+\epsilon)$ factor in the definition of simple cover complexity is that a query segment could be covered by only two balls, while there may be $\Omega(n)$ obstacle vertices arbitrarily close to the segment. Consider, for example, two open unit balls whose centers are infinitessimally less than distance 2 apart. The segment joining their centers is covered by the balls, but there may be many obstacles that are arbitrarily close to the region where these two balls intersect. Thus the value ϵ serves a role in making the size of the cover complexity insensitive to small geometric perturbations.

Intuitively, it is easy to see that a segment that is short and well separated from all obstacles will have a smaller complexity than a segment that grazes the boundary of the obstacles. In this sense, we feel that this complexity measure is a reasonable reflection of the intrinsic difficulty of solving a segment shooting query. Observe that this complexity measure is invariant under rigid transformations of the plane (translation, rotation, uniform scaling); however, it is not invariant under general nonsingular affine transformations (for example, nonuniform scaling and shearing), since balls may not be preserved under these transformations.

It is possible to strengthen the notion of cover complexity to simple connected regions of space. First, observe that given any ball B, the obstacle set subdivides the points lying within this ball into various connected components, which we call C-balls (where "C" stands for connected). More formally, consider points p and q lying within B. We say that p can *reach* q within B if there is a path from p to q

that lies entirely within B and does not intersect any obstacle. Reachability within B defines an equivalence partition of B into *connected components*. Define a *C-ball* to be any such connected component of any ball. A C-ball is *simple* if its boundary intersects at most 2 edges of P (or, more generally, at most δ faces, in higher dimensions). Because each C-ball is associated with a ball, the center and radius of a C-ball are defined to be the center and radius, respectively, of the associated ball. Note that if one ball contains another, then its connected components contain the connected components of the other. Given any $\epsilon > 0$, a C-ball of radius r is *ϵ-strongly C-simple* if the C-ball with the same center and radius $(1+\epsilon)r$ is simple. Based on this notion of C-simplicity, we can derive a notion of *C-complexity*, as we did above. We use $cscc_\epsilon(D)$ to denote the C-complexity of a domain D.

Clearly any ball that is simple is also C-simple, and thus the C-complexity of an object is not greater than its complexity. However, C-complexity may be smaller. For example, as shown in Figure 1(b), the C-complexity of the segment shown in part (a) is only 4. Intuitively, C-complexity is a more appropriate measure of the complexity of a ray shooting query, since obstacles that lie on the far side of an obstacle edge should not affect query processing, no matter how close they may be to the query segment.

1.4. Statement of Main Results

We present three principal results. First, for ray shooting queries in arbitrary dimensions, we analyze the performance of a hierarchical spatial subdivision, called a *smoothed k-d subdivision*. This subdivision is related to the well-known quadtree data structure, and is based more specifically on a data structure called a PM *k-d* tree, described by Samet.[31] Samet gives an analysis of the maximum depth of this tree in terms of the ratio between the diameter of the obstacle set and the minimum separation distance between nonadjacent vertices and edges. Various authors have derived relationships between the size of quadtrees for representing shapes and intrinsic properties of the shape (perimeter or, more generally, surface area) and the accuracy of the representation.[20,26] We show that the size of a smoothed *k-d* subdivision is proportional to the simple cover complexity of the obstacle-free space. This result in itself is interesting because this complexity measure, while less intuitive than perimeter, is independent of the accuracy of representation (and unlike standard quadtrees, there is no loss of precision in our representation). We also show that the number of cells of the subdivision that are stabbed by a query segment is proportional to the simple cover complexity of the query segment. We show that the hierarchical subdivision can be used to locate the origin-point of the directed query segment in time $O(\log \frac{R}{r})$, after which we can trace the segment through the subdivision in time $O(scc(s))$ to answer the shooting query, where s is the query segment, $scc(s)$ is the simple cover complexity of the initial obstacle-free portion of s, r is the radius of the largest strongly simple ball containing the origin of s, and R is the diameter of the obstacle set.

Second, we extend the bounds above on the stabbing number to C-complexity by introducing a variation of the smoothed *k-d* subdivision, called a *reduced k-d* *subdivision.*

Our final result involves ray shooting in the plane. We assume we are given a collection of polygonal obstacles with n vertices total. We show that in $O(n \log n)$ time, it is possible to produce a subdivision of the plane, called the *reduced box-decomposition subdivision*, of size $O(n)$, such that segment shooting queries can be answered in time $O(\log n + cscc(s))$, where $cscc(s)$ is the C-simple cover complexity of the obstacle-free portion of the query segment. The $O(\log n)$ term is the time needed for point location, and $O(cscc(s))$ term is the bound on the number of cells of the subdivision stabbed by the query segment.

1.5. A Technical Lemma

Before describing our algorithms we mention one technical fact. The definition of simplicity is based on a parameter $\epsilon > 0$, which is independent of n. The exact choice of ϵ affects only the constant factors in our bounds, as shown by the following result. So, in the remainder of the paper, we omit references to ϵ when deriving asymptotic bounds.

Lemma 1 *Let $0 < \epsilon < \epsilon'$, and let D be some domain in dimension d. There is a constant γ (depending on ϵ, ϵ' and the dimension d) such that*

$$scc_{\epsilon}(D) \;\leq\; scc_{\epsilon'}(D) \;\leq\; \gamma \cdot scc_{\epsilon}(D).$$

The same result holds for C-complexity as well.

Proof. We present the proof in the plane for simplicity, but the generalization to arbitrary dimensions is straightforward. We will derive a very loose bound on γ. The first inequality is trivial, since if the expansion of a ball in any cover by $(1+\epsilon')$ is simple, then an expansion by only $(1+\epsilon)$ is certainly simple. To prove the second inequality, let $\mathcal{B}$ be a simple cover with respect to ϵ. We will show that we can cover each ball $B \in \mathcal{B}$ with at most γ smaller balls whose expansions by $(1+\epsilon')$ are contained within an expansion of B by $(1+\epsilon)$. Let r denote the radius of B. Let

$$r' = \frac{r\epsilon}{2(1+\epsilon')}.$$

Cover B with a minimal set of squares (hypercubes, in general), arranged in a grid, whose diagonals are of length $2r'$. Enclose each square of this grid within a ball of radius r'. Observe that the resulting set of balls covers B. If we grow any one of these balls by $(1+\epsilon')$, every point in the resulting ball lies within distance $2r'(1+\epsilon') = r\epsilon$ of the boundary of B, and hence lies within the expansion of B by $(1+\epsilon)$. Thus each of these balls is $(1+\epsilon)$ strongly simple. Finally observe that the number of newly formed balls is equal to the number of grid squares needed to cover B. This is not more than the number of grid squares of diagonal length $2r'$ (side length $r'\sqrt{2}$) needed to cover a square of side length $2r$. It is easy to verify that this is at most

$$\left\lceil \frac{2r}{r'\sqrt{2}} \right\rceil^2 = \left\lceil \frac{4(1+\epsilon')}{\epsilon\sqrt{2}} \right\rceil^2.$$

In general, the exponent and constant factors depend on the dimension d. Letting γ be this constant completes the proof. Clearly, this construction can be generalized to C-simple balls as well. $\Box$

2. A Simple Approach

In this section we give a query-sensitive analysis of a simple, practical decomposition strategy, which illustrates the essential features of the more complex method and its analysis that will be presented in the following section. The decomposition produces a subdivision of space called a *smoothed k-d subdivision*, and is based on the PM *k-d* tree and PM quadtree decompositions described by Samet.[31] We have chosen the binary *k-d* subdivision over the more well known 2^d-ary quadtree subdivision for the practical reason that it results in a somewhat smaller number of regions, especially in higher dimensions.

We begin by assuming that we are given a set of polyhedral obstacles in d-dimensional space, represented simply by the set of its faces (of all dimensions). As mentioned in the introduction, we assume that faces bounding these obstacles are simple in the sense that each point is incident to at most a constant number β of faces of all dimensions. We also assume that the obstacles are contained within a minimal bounding hypercube. (Such a hypercube is easy to construct in linear time.) This hypercube is our initial *box*. A box is said to be *crowded* if it intersects more than δ faces. (In general any constant threshold at least as large as δ can be chosen.) If a box is crowded, then it is *split* by a hyperplane passing through the center of the box and perpendicular to its longest side. (Ties for the longest side can be broken arbitrarily.)

Each split replaces a box by two smaller boxes. Since we start with a hypercube, and each split cuts through the midpoint of the longest side of a box, it follows that each box is "fat" in the sense that the ratio between its longest and shortest side is at most 2.

The splitting process naturally defines a binary tree called a *k-d* tree. The original box is the *parent* of the resulting two boxes in this tree. The splitting process terminates when no remaining boxes are crowded. The resulting leaf boxes, called *cells*, form a subdivision of space, each of which contains a constant number of obstacle boundary elements. (The cells correspond to the leaves of the binary tree modeling our hierarchical decomposition. The boxes corresponding to internal nodes of the tree are not cells, although they are a useful conceptual device in our proofs.) The internal nodes of the resulting binary tree each contain data describing the splitting plane. Each leaf node contains pointers to the obstacle boundary elements that overlap the associated cell.

The splitting process eventually terminates, because of the bounded incidence assumption. However, the number of cells in the subdivision depends on the geometric placement of the obstacles, and it cannot be bounded only as a function of n, the number of obstacle faces.

To support segment shooting queries, we make one modification, which we call *smoothing.* Intuitively, smoothing involves additional splitting of the rectangles of the decomposition so that neighboring cells are of similar sizes. (Smoothing is a well-known operation on quadtree data structures and is better known under the names of *"restriction"*[25] or *"balancing"*.[12,27,28] We use the term "smoothing" to avoid confusion with depth balancing in binary trees.) We say two cells are *neighbors* if they share a common boundary of dimension $d-1$. Define the *size* of a cell to be the length of its longest side. If there are two neighboring cells whose sizes differ by more than some constant factor (at least 2), then the larger cell is split. Splitting is repeated until all neighboring cells satisfy the size threshold. It is not hard to see that smoothing will eventually terminate since it can only decrease the size of the cells, and no cell can be made smaller than the smallest cell in the tree prior to smoothing. In fact, smoothing may generally increase the size of the tree by no more than a constant factor.[28] The resulting subdivision is called a *smoothed k-d tree subdivision.*

2.1. Query analysis using standard complexity

To analyze the time for query processing, we first provide a relationship between the size of cells in the subdivision and strongly simple balls. For these results, we use the standard notion of simplicity (not C-simplicity).

Lemma 2 *Let $\epsilon > 0$, and let B be an ϵ-strongly simple ball of radius r. If B intersects a cell of the decomposition of size s, then $s \geq \alpha r$ for some constant α (depending on ϵ and the dimension d).*

Proof. For concreteness, assume that in smoothing, we split a cell if a neighbor is of size less than 1/2 of its size. Let $\alpha = \frac{\epsilon}{6\sqrt{d}}$. Suppose to the contrary that there exists a cell that violates the conditions of this lemma. Let c be a smallest violating cell and let s denote its size. That is, c is a smallest cell such that there exists an ϵ-strongly simple ball B of radius r intersecting c such that $s < \alpha r$. Let p be the parent box of c in the smoothed *k-d* tree (see Figure 2). Box p may have been split for one of two reasons. Either (1) p itself is not simple, or (2) p is the neighbor of a cell c' whose size is less than 1/2 the size of p, and we split p for smoothing. The size of p is at most $2s$, and hence the diameter of p is at most $2s\sqrt{d}$.

In case (1), since c intersects B, and hence so does c's parent, then within distance $2s\sqrt{d}$ of the boundary of B there is a crowded box, intersecting more than δ faces of P. But,

$$2s\sqrt{d} < 2\alpha r\sqrt{d} = \frac{2\epsilon r\sqrt{d}}{6\sqrt{d}} \leq \epsilon r.$$

Therefore, the $(1+\epsilon)$ expansion of B intersects more than δ faces of P, and is not simple. This is a contradiction.

In case (2), let s' denote the size of c'. Since $s' < 2s/2 = s$, and since sizes vary by factors of 2, $s' \leq s/2$. We will prove below that there is an ϵ-strongly simple ball B' of radius $r/2$ that intersects c' (see Figure 2). Assuming this for now, since c is a smallest cell violating the conditions of the lemma (for the fixed value of α), we know that c', a cell smaller than c, *must* satisfy the conditions of the lemma; i.e., for *all* choices of an ϵ-strongly simple ball that intersects c', s' must be at least as

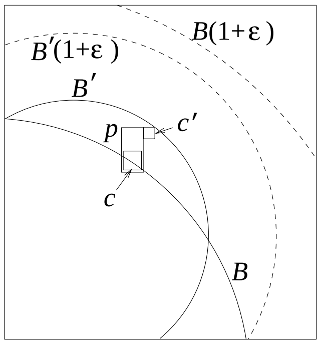


Figure 2: Proof of Lemma 2.1.

large as α times the radius of the ball. Thus, we have

$$\frac{s}{2} \geq s' \geq \alpha \frac{r}{2},$$

implying that $s \geq \alpha r$, contradicting our earlier hypothesis.

To prove the existence of B', first observe that since the size of c' is at most $s/2$, its diameter is at most $s\sqrt{d}/2$. Since c' is a neighbor of p, and since p intersects B, c' lies entirely within distance $2s\sqrt{d} + s\sqrt{d}/2 \leq 3s\sqrt{d}$ of the boundary of B. There exists a ball B' of radius $r/2$ that intersects c' and is at no point further than $3s\sqrt{d}$ from the boundary of B. The expansion of B' by $(1+\epsilon)$ lies within distance $r\epsilon/2$ from the boundary of B'. Hence this expansion of B' lies within distance

$$3s\sqrt{d} + \frac{r\epsilon}{2} \;<\; 3\alpha r\sqrt{d} + \frac{r\epsilon}{2} \;=\; \frac{3r\epsilon\sqrt{d}}{6\sqrt{d}} + \frac{r\epsilon}{2} \;=\; \frac{r\epsilon}{2} + \frac{r\epsilon}{2} \;=\; r\epsilon$$

of the boundary of B. Thus the expansion of B' lies entirely within the expansion of B, implying that B' is simple. □

Lemma 3 *Given an ϵ-strongly simple ball B of radius r, the number of cells that intersect B in the subdivision is bounded above by a constant (depending on ϵ and the dimension d).*

Proof. By Lemma 2, the size of a cell intersecting B is bounded from below by αr. Because the cells of the subdivision are disjoint and fat, the number of such cells intersecting B is bounded above by the number of disjoint hypercubes of one half this side length that can be packed around (inside or touching) B, which by a simple packing argument is on the order of $1/\alpha^d$. □

Lemma 4 *Let q be a point in the enclosing hypercube of the obstacle set of side length R. If q is located in a strongly simple ball of radius r, then we can locate the cell containing q in time $O(\log(R/r))$. (Constant factors depend on d and ϵ.)*

Proof. The search for q is a simple descent through the smoothed *k-d* tree. With every d levels of descent, the size of the associated regions decreases by a factor of $1/2$. Thus, if the search continues for k steps, then the search terminates at a region that contains q and whose size s obeys $s \leq \frac{R}{2^{\lfloor k/d \rfloor}}$. But Lemma 2 implies that $s \geq \alpha r$, for a constant α, so that $k = O(\log(R/r))$. $\square$

Segment shooting queries are processed by first locating the cell containing the origin of the segment, and then walking along the segment, cell by cell, through the subdivision. Because neighboring cells differ in size by a constant amount, and because the ratio of the longest to shortest side of each cell is bounded by 2, it follows that the number of neighbors of a given cell is at most a constant depending on the dimension. From the fact that each cell contains a constant number of obstacle boundary elements, it follows that each cell can be tested for intersection with the query segment in constant time, and hence the time to perform this walk is proportional to the number of cells visited. Let $scc(s)$ denote the simple cover complexity of a segment s (more precisely, the complexity of the obstacle-free initial subsegment of s), and let $\mathcal{B}$ be a strongly simple cover for s. By Lemma 3, the number of subdivision cells intersecting each ball in $\mathcal{B}$ is a constant, and hence, since these balls cover s, the number of subdivision cells traversed by s is $O(|\mathcal{B}|) = O(scc(s))$. By combining this with the previous lemma, it follows that segment shooting queries can be answered in $O((\log R/r) + scc(s))$ time. Constant factors vary with the dimension d.

By a similar argument, the number of cells in the subdivision is $O(scc(P))$, where $scc(P)$ denotes the simple cover complexity of the obstacle-free space. The reason is that each ball of the cover overlaps at most a constant number of cells of the smoothed subdivision. Since each cell contains only a constant number of boundary elements, the size of the final subdivision is $O(scc(P))$. From this we have the following analysis of the smoothed *k-d* tree subdivision.

Theorem 1 *Let P be a collection of polyhedral obstacles satisfying the bounded incidence assumption, and let R denote the side length of the smallest hypercube enclosing P. The smoothed k-d subdivision is a subdivision of size $O(scc(P))$ such that given any directed query segment s, whose origin lies within a strongly simple ball of radius r, the query can be answered in $O((\log R/r) + scc(s))$ time.*

We do not consider preprocessing time in our analysis, because it depends on the implementation of the underlying primitive operation of efficiently determining which obstacle faces intersect the boxes of the decomposition. This depends on the representation and complexity of the faces making up the obstacles. If we assume that obstacle faces are of bounded complexity (e.g. through triangulation) then this primitive operation can be performed in $O(1)$ time for each box/face pair. This provides a trivial preprocessing time bound of $O(n \cdot scc(P))$, where n is the number of obstacle faces. In Section 3, we show that this can be improved considerably in the plane.

2.2. A Modification using C-complexity

In this section we extend the results of the previous section to the stronger notion of C-complexity. We do this by coarsening the smoothed *k-d* subdivision, resulting in a subdivision that we call a *reduced k-d subdivision*. This subdivision will be of interest later in the presentation, because it forms the basis for the analysis of the more complicated data structure to be presented in Section 3.

For simplicity, we describe only the planar case, but the generalization to higher dimensions is straightforward. The appropriate generalization of Lemma 2 that we seek is that if a strongly *C-simple* ball of radius r intersects a cell c, then c is of size at least αr for some constant α. This may not be the case for a *k-d* tree subdivision (even without the additional cells caused by smoothing) as shown in Figure 3(a). A large number of small subdivision cells lie to the right of edge e in this figure. The existence of these cells is justified from the vertices lying to the left of e. However, a point lying just to the right of e will be contained within a large strongly C-simple ball, although its containing cell may be arbitrarily small. For this reason, a ray lying just to the right of e and parallel to e will have a very low C-simple cover complexity, but a high stabbing number with respect to the subdivision.

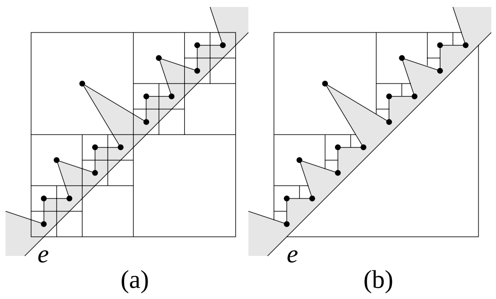


Figure 3: Merging.

To remedy this, we present a method to remove unwanted cells from the subdivision. It will simplify the presentation to make the general position assumption that obstacle edges and *k-d* tree edges are either disjoint or intersect in a single point. This assumption can be overcome by a careful treatment of the various special cases. The first step is to compute the planar subdivision that results from overlaying the smoothed *k-d* subdivision on the obstacle edges. Cells of the subdivision (2-dimensional faces) that lie entirely within obstacle interiors are deleted. (In some of the subsequent figures, these interior cells have been shown to make the hierarchical subdivision structure clearer.) We say that two cells of the subdivision are *neighbors* if they share a common boundary that is not part of any obstacle, that is, it is a segment of the *k-d* tree decomposition.

By the nature of the *k-d* tree decomposition we know that each cell of this subdivision contains at most δ obstacle edges on its boundary. From smoothness we know that each cell of this subdivision has a bounded number of neighbors. The cells of the subdivision may not be convex, but the nonconvexity exists only because of the obstacles. Thus the subdivision possess the following three *basic properties*.

Bounded boundary complexity: Each cell has at most δ obstacle edges on its boundary,

Bounded neighborhood size: Each cell has at most a constant number, ν, of neighboring cells, and

Relative convexity: A line segment that intersects none of the obstacles intersects a cell in a single segment.

We essentially "undo" some of the splits performed by the smoothed *k-d* tree decomposition, by merging neighboring regions, as long as C-simplicity and smoothness are not violated. In particular we apply the following *merging procedure.* Every nonobstacle edge in the subdivision is a subsegment of some splitting segment in the underlying *k-d* tree. The splitting segments of the *k-d* tree are in 1–1 correspondence with the internal nodes of the tree. Working from the leaves of the *k-d* tree to the root, the merging procedure attempts to merge neighboring cells by removing edges of the subdivision that correspond to splitting segments in the *k-d* tree.

Initially all cells are considered *eligible* for merging, and each cell is associated with the leaf box of the *k-d* tree that contains it. Working from the bottom (leaf level) to the top of the *k-d* tree, we consider neighboring cells of the subdivision that are eligible for merging, and whose associated boxes of the *k-d* tree are siblings, separated by a common splitting segment. We consider the connected components of space that would result if this splitting segment of the *k-d* tree were removed, and the cells on either side were merged. This is called a *trial merge.* For each of these merged connected components, if the component satisfies the three basic properties listed above, then the cells are replaced with their union in the subdivision, thus locally undoing the decomposition step. (Only the first two basic properties need be tested explicitly. It is easy to see that relative convexity is preserved by the nature of the merging process.) A single splitting segment may separate many connected components, and the trial merge may succeed for some components and fail for others. When two cells are successfully merged, the merged cell is associated with the parent box in the *k-d* tree, and it becomes eligible for subsequent merging. Otherwise the cells are not merged, and are not eligible for future merging.

If a cell is associated with one of the two *k-d* tree boxes that are being merged, but it is not incident to the splitting segment, then it is treated as though it has succeeded with a trivial merge with an empty cell. In particular, it is associated with the parent box in the *k-d* tree, and continues to be eligible for merging.

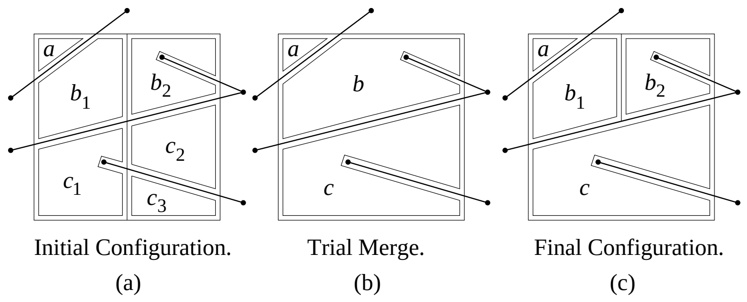


Figure 4: Merging procedure.

For example, Figures 4 (a) and (b) illustrate the effect of a trial merge; the six cells a, b_1, b_2, c_1, c_2, and c_3 are replaced by the three connected components a, b and c. The cell $b = b_1 \cup b_2$ violates the complexity bound (assuming at most 2 obstacle edges are permitted), and so they are not merged. On the other hand, assuming that c_1, c_2, c_3 are all eligible for merging, they are merged into a single cell c. (If any of the three were ineligible, then none of them would be merged.) Cell a is not incident to the splitting segment, and so succeeds in a trivial merge. Consequently, cells a and c become eligible for subsequent merging. The *k-d* tree node associated with cells a and c is the entire square. The result of this merging step is shown in Figure 4(c).

The end result of the merging process is called a *reduced k-d subdivision.* Each cell in this subdivision satisfies the three basic cell properties listed above. Each cell is a connected component of some box of the smoothed *k-d* tree (because we merge all connected neighboring cells or none of them). These cells are maximal in the sense that the corresponding connected component of the parent box in the *k-d* would violate the three properties above. We define the *size* of a cell to be the size (length of the longest side) of the associated box of the *k-d* tree. Because cells are subsets (connected components) of their associated *k-d* boxes, the Euclidean diameter of a cell may be arbitrarily small compared to its size (as seen with cell a in Figure 4(c)). We now give the appropriate generalization of Lemma 2.

Lemma 5 *Let $\epsilon > 0$, and let B be an ϵ-strongly C-simple ball of radius r, and let S be a reduced k-d subdivision. If B intersects a cell of S of size s, then $s \geq \alpha r$ for some constant α (depending on the strong cover bound ϵ and the dimension).*

Proof. The value of α is the same as in Lemma 2. Our proof is based on considering just the obstacle edges incident to the C-simple ball, and then reducing to Lemma 2.

Consider the $(1+\epsilon)$ expansion of B. Let P' denote the obstacle vertices and edges of P that are incident to this connected component. For the sake of analysis, consider an alternative (infinitely large) smoothed *k-d* subdivision S' for P', that starts with the same initial enclosing square that was used for the original *k-d* tree, but in which any box that lies all or partially outside of the $(1+\epsilon)$ expansion of B is split. (Thus merging has been applied to S but not applied to S'.) A cell of S' has nonzero size only if it lies entirely within the $(1+\epsilon)$ expansion of B. (This is illustrated Figure 5, where S is shown in (a), and S' is shown (b). Cells have been split if they intersect more than two obstacle edges.)

We claim that the box associated with any cell c of S that intersects B cannot be properly contained within any of the cells of S'. (And, in particular, the situation illustrated in the figure could not arise.) Suppose that this were so and consider the smallest such cell c. Let b be the associated box of the smoothed *k-d* tree, and suppose that b is properly contained within some box b' of S'. Because both decompositions started with the same bounding square, b' is a proper ancestor of b (in the abstract *k-d* ordering of boxes). Therefore, b' contains not only b, but also the merge of b with its sibling box in the *k-d* tree. But, because b' has nonzero size, it lies entirely within the $(1+\epsilon)$ expansion of B, and hence is not crowded

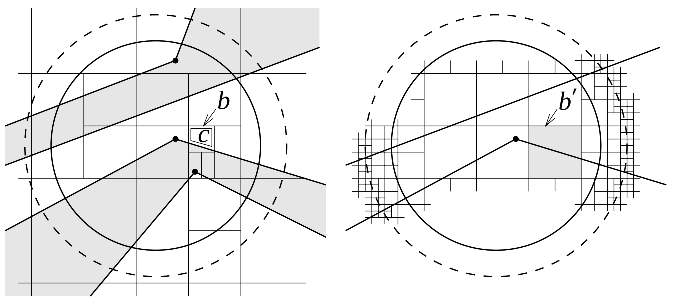


Figure 5: Proof of Lemma 5.

with respect to P'. It follows that the resulting connected component has bounded boundary complexity with respect to P. Therefore, boundary complexity could not be an impediment to the merge of b and its sibling.

We also claim that the sizes (and hence number) of c's neighbors cannot be an impediment to the merger. The reason is that the obstacles of P' located within the $(1+\epsilon)$ expansion of B are the same for the two *k-d* trees, and we have constructed P' so that any cell extending outside of the expansion will be split. S cannot subdivide cells for smoothing purposes any finer than this.

Finally, we claim that relative convexity is not violated by the merger. This is an immediate consequence of the nature of the merging process. Since none of the three merging properties are violated, c would have been merged with its neighbor, yielding the contradiction.

Because the box associated with c cannot be properly contained within any of the cells of S', and because the size of a cell is based on the size of its associated box, we may apply Lemma 2 to cells of S', to establish the same lower bound of αr on the size of c. □

Lemma 6 *Given an ϵ-strongly C-simple ball B of radius r, the number of cells of the reduced k-d subdivision that intersect B is bounded above by a constant (depending on ϵ, δ, and the dimension d).*

Proof. By Lemma 5, the size of a cell intersecting B is bounded from below by αr. This means that the size of the associated box in the *k-d* tree is bounded by this value. It follows from Lemma 3 that there are a constant number of boxes. To complete the proof, it suffices to show that given any box b in the smoothed *k-d* tree, there are at most a constant number of cells of the reduced *k-d* subdivision associated with b that intersect B. This is true because each cell of the reduced *k-d* tree is a connected component of its associated box in the smoothed *k-d* tree. Because B is strongly C-simple, there are a constant number of edges incident to B, and hence a constant number of connected components of b intersect B. □

This leads the main result of this section. Because we have replaced the hierarchical *k-d* structure with a more general subdivision (based on connected components), the point location time bound of $O(\log R/r)$ does not necessarily hold.

Theorem 2 *Let P be a collection of polyhedral obstacles satisfying the bounded incidence assumption. The reduced k-d subdivision subdivides space into $O(cscc(P))$ cells each of bounded complexity, such that a line segment s that does not intersect any obstacle intersects $O(cscc(s))$ cells of the subdivision.*

The method of first constructing the refined *k-d* structure and then coarsening through merging may seem unnecessarily complicated. Obviously a more direct approach is to keep track of the connected components throughout the construction process, and avoid the additional splits. We have presented this particular approach because it more closely parallels the presentation in the next section, where it is not as clear how to implement the direct approach efficiently.

In summary, the smoothed *k-d* subdivision and the reduced *k-d* subdivision have the advantages of simplicity and ease of generalization to higher dimensions. It is unfortunate that it is not possible to bound the space or preprocessing time for these data structures purely in terms of a function of input size. In the next section we show that it is possible to do so when obstacles lie in the plane.

3. The Planar Case

In this section we modify the simple reduced *k-d* subdivision given in the previous section to handle segment shooting for a set P of polygonal obstacles in the plane. We call the resulting structure a *reduced box-decomposition subdivision.* Our goal will be to prove the following main theorem. We assume throughout this section that the bound of at most two obstacle edges is used in the definition of C-simplicity.

Theorem 3 *Let P be a collection of disjoint polygonal obstacles in the plane, with a total of n vertices. A reduced box-decomposition subdivision of size $O(n)$ can be built in $O(n \log n)$ time, such that given this data structure, segment shooting queries for a query segment s can be answered in $O(\log n + cscc(s))$ time, where $cscc(s)$ is the strong C-simple cover complexity of the initial obstacle-free portion of s.*

Interestingly, the subdivision presented here has a number of similarities to the quality mesh triangulations of Bern, Eppstein, and Gilbert,[12] and Mitchell and Vavasis.[27] Since there are no restrictions on the geometric structure of our subdivision, we can provide the space and preprocessing time improvements listed above. (In mesh generation, the size of the triangulation generally depends on the geometric structure of the input.) The construction of the reduced box-decomposition subdivision involves a number of steps. They are presented in each of the following subsections.

3.1. Box-Decomposition

Preprocessing begins with a hierarchical subdivision similar to the PM *k-d* tree, described in the last section. To achieve linear space, it is necessary to modify the decomposition procedure. The decomposition begins as an adaptation of a standard decomposition method, called *box-decomposition.* This basic concept was introduced by Clarkson[19] and has appeared in a number of different varieties.[8,11,13,34] The initial subdivision is based solely on the vertices of the obstacle set. We do not require that our subdivision be a cell complex, since there may be vertices lying in the interior of edges; however, the number of vertices in the interior of any edge will be a constant. Thus, we can easily augment such a subdivision into a cell complex or a triangulation with a linear number of additional Steiner points.

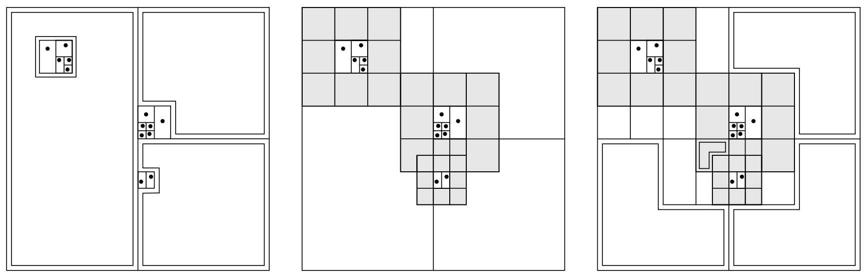
As with the PM *k-d* tree, the decomposition is hierarchical and naturally associated with a binary tree, called the *box-decomposition tree*. Each node of the box-decomposition tree corresponds to one of two types of objects called *enclosures*: (1) a *box*, which is a rectangle with the property that the ratio of the longest to shortest side is at most 2, and (2) a *doughnut*, which is the set theoretic difference of two boxes, an *inner box* contained within an *outer box*. It will simplify the presentation to assume that inner boxes are always square. The *size* of a box is defined to be the length of its longest side, and the *size* of a doughnut is the size of its outer box. We assume that the obstacles have been scaled to lie within a unit square.

The root of the tree is associated with the enclosing unit square. Inductively, we assume that the point set to be decomposed (initially the set of all obstacle vertices) is contained within a box. If there is at most one vertex, then the decomposition terminates here. Otherwise we consider a horizontal or vertical line that bisects the longer side of the box. (Ties can be broken arbitrarily, but in our examples we assume that the vertical split is made first.) If there exists at least one vertex on either side of this splitting line, then this line is used to split the box into two identical boxes. We partition the vertices according to which of the two boxes they lie in and recursively subdivide each. (Vertices that lie on the splitting line may be placed on either side of the splitting plane.) This process is called *splitting*.

Without the provision above that there exists a vertex on each side of the splitting line, the splitting process could result in an arbitrarily long series of *trivial splits*, in which no vertices are separated from any other, and this in turn could result in a subdivision of size greater than $O(n)$.

To avoid this, a different decomposition step called *shrinking* may be performed. Shrinking consists of first finding the smallest quadtree box that contains all the vertices. By a *quadtree box*, we mean any square that could be generated by some repeated application of the quadtree subdivision rule, which, starting with the initial bounding square, recursively subdivides a square into four identical squares of half the side length. Such boxes have side lengths that are powers of $1/2$, and a box of side length $1/2^k$ has vertex coordinates that are multiples of $1/2^k$. The minimum quadtree box surrounding a given rectangle can be computed in constant time assuming a model of computation where integer logarithm, integer division, and bitwise exclusive-or operations are supported in $O(1)$ time on the coordinates[8]. (The restriction of using quadtree boxes simplifies the presentation of the smoothing operation, to be described later. It can be overcome with a somewhat more flexible definition of boxes and shrinking.[14])

If the smallest quadtree box enclosing the vertices is sufficiently small compared to the current box, we apply a shrinking operation rather than splitting. Shrinking produces two enclosures. One is the inner box containing the vertices, and the other is the surrounding doughnut, which contains no vertices. The doughnut becomes a leaf in the decomposition tree, and the subdivision process is applied recursively to the inner box. See Figure 6(a) for an illustration of the decomposition. Doughnuts are indicated with double lines.



Initial box-decomposition. (a) Buffer zones. (b) Decomposition with zones. (c)

Figure 6: Box-decomposition with buffer zones.

This subdivision is modeled by a binary tree, called the *box-decomposition tree*, where each internal node has two children (either the left and right side from a split, or the doughnut and inner box from a shrink). The leaves of this tree define a subdivision of the plane into boxes and doughnuts. These are called *basic enclosures*. Box-decomposition can be performed in $O(n \log n)$ time and produces a tree of size $O(n)$ (see e.g. Refs. 8,13).

Remark: Although shrinking is crucial to proving the desired space bounds in worst-case scenarios, it should be mentioned that it is rarely needed in practical situations. Shrinking is only needed when obstacle vertices form extremely tight clusters, when compared to the distance to the nearest points lying outside the cluster. Furthermore, shrinking complicates the smoothing process, described below. Under the practical assumption that the number of trivial splits does not exceed $O(n)$, all the results of this paper hold without the need for shrinking.

3.2. Smoothing

To handle segment shooting queries we will need to smooth the subdivision as in the Section 2.2. The presence of shrinking requires that this be done carefully, in order to keep the size of the resulting data structure from growing by more than a constant factor. The problem is that shrinking, by its very nature, introduces neighboring enclosures of significantly different sizes. Thus, the operations of shrinking and smoothing are incompatible in some sense. Although the result may consist of boxes of significantly different sizes, the goal of smoothing is that each box has a constant number of neighboring boxes.

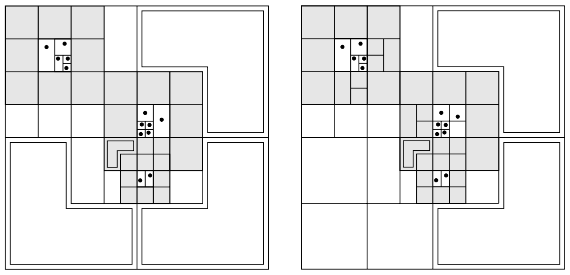
Smoothing is performed in two phases. The first phase of smoothing is to surround the inner box of each doughnut with a *buffer zone* of eight identical boxes (these are the shaded boxes in Figure 6(b)). If the inner box touches the boundary of the outer box, then these boxes may spill over into neighboring enclosures. In the case of spill-over, these boxes may already be a part of the decomposition, and no further action is needed. If spill-over occurs, and these boxes are not part of the decomposition, then this neighbor will need to be decomposed by applying either splitting or shrinking appropriately, until we have produced a legal box-decomposition that contains both the boxes of the original box-decomposition and these new buffer boxes.

This further decomposition can be implemented in the same $O(n \log n)$ time as box-decomposition. In particular, think of each newly created buffer box as a new "fat" obstacle vertex. Then a simple modification of box-decomposition on the union of the obstacle vertices and these new fat points can be applied. Observe that by our choice of buffer boxes as quadtree boxes, if a buffer box is properly contained within a box of the decomposition, and this box is split, then the buffer box will lie entirely to one side or the other of the splitting line. If a buffer box contains more than one obstacle vertex, then the buffer box will be created as an internal node in the decomposition tree, and it will continue to be subdivided. Because the original tree had $O(n)$ boxes, there are $O(n)$ buffer boxes, and hence the final decomposition will have $O(n)$ total size. The result of the decomposition with buffer zones is illustrated in Figure 6(c).

Define a *neighbor* of a basic enclosure to be any other basic enclosure such that the two share a common line segment on their boundaries. The second phase of smoothing is to determine whenever two neighboring enclosures differ in size by more than some constant factor (at least 2). In such a case, the larger enclosure is repeatedly split (never shrunk), until the difference in size falls below this factor. However, there are two important exceptions. First, inner boxes do not induce neighboring enclosures to be split. The inner boxes of the original decomposition are surrounded by buffer zones, so this does not apply to them. It applies to any inner box that was generated as part of the decomposition with the buffer boxes. Second, splitting that is induced from an inner in the original decomposition is allowed to propagate into its buffer zone, but not beyond there. This exception is important so that the buffer zone enclosures surrounding a small inner box do not induce a much larger surrounding doughnut to split (for otherwise we will lose the space savings achieved by shrinking).

When splitting is applied to a shrinking node, the doughnut is split. As mentioned above, because we have chosen inner boxes to be quadtree boxes, observe that the inner box will lie entirely to one side or the other of each splitting line. Thus, after splitting a doughnut, there will be a new smaller doughnut with the same inner box, and a regular box (which will contain no vertices). As the doughnut is split and becomes smaller, if its size becomes sufficiently close to the size of its inner box, we can just replace the shrink with a small number of splits.

In Figure 7(a) we show the output of the first smoothing phase. (This is the same as Figure 6(c).) Buffer zone enclosures are shaded. In (b) we show the final decomposition produced by the second smoothing phase. Splits are performed if the difference in size between an enclosure and its neighbor exceeds a factor



Decomposition with zones. (a) After smoothing. (b)

Figure 7: Smoothing: Second step.

of 2. As before, vertical splits are performed before horizontal splits when both sides are of equal length. Observe that the shaded enclosures in (b) do not induce larger neighboring enclosures to be split, but unshaded enclosures do induce larger neighbors to be split, whether shaded or not.

To implement this phase of the smoothing process, we assume the algorithm maintains, for each side of each enclosure, a list of pointers to the neighboring enclosures, called a *neighbor list.* This is a doubly-linked list sorted along the side of the enclosure. These lists are cross-indexed, so that an enclosure can locate the entries in its neighbor's neighbor lists that refer back to it.

This phase is implemented in a bottom-up manner, working from small enclosures to larger enclosures. Enclosures can be stored in a fast priority queue according to size, so the smallest current enclosure can be extracted in logarithmic time. When a small enclosure identifies a sufficiently large neighboring enclosure, it induces the neighbor to split. When an enclosure is split, the neighbor lists of the two sides that were split need to be split as well. This is done by applying a dovetailing search simultaneously inward from both sides of both neighbor lists, until finding the splitting points on each of the lists. (Thus there are four searches running simultaneously.) When the splitting points are known, each list is split into two sublists (in constant time). The subenclosure having the larger total number of neighbors along the split edges retains the name of the original enclosure. The other subenclosure is given a new name, and the cross-index links are accessed to update the neighbor lists of all its neighboring enclosures.

The reason for using dovetailing search and renaming the subenclosure with the smaller number of neighbors is to achieve the desired running time. Consider a single split, and let n_1 and n_2 denote the number of neighbors of each the two subenclosures after splitting. Assume without loss of generality that $n_1 \leq n_2$. The important aspect of this splitting process is that this split can be performed in $O(n_1)$ time. This is true because the dovetailing search will identify the splitting point in this time, and because only this subenclosure is renamed, its neighbor's neighbor lists can all be updated in $O(n_1)$ time. It follows from standard arguments (Ref. 7, page 127) that if we start with neighbor lists of total size $O(n)$, and each partition can be performed in time proportional to the smaller side of the partition, then the total time spent in partitioning will be $O(n \log n)$.

Lemma 7 *The smoothed box-decomposition tree can be constructed in $O(n \log n)$ time and $O(n)$ space.*

Proof. The initial box-decomposition can be built in $O(n \log n)$ time and $O(n)$ space.[8,13] The first phase of smoothing is effectively equivalent to running a box-decomposition with at most $8n$ additional objects, and so can be accomplished within the same time bound. To complete the proof it suffices to show: (1) that the second smoothing phase can be performed in $O(n \log n)$ time, and (2) that the final decomposition is of size $O(n)$.

Assuming for now that (2) holds, we prove (1). First observe that if the size of the final decomposition is $O(n)$, then the neighbor lists essentially define a planar graph on the decomposition, which by Euler's formula has size $O(n)$. For each split, in $O(\log n)$ time we can extract the smallest enclosure from the priority queue. A single pass through its neighbor list suffices to locate all larger neighbors. Using the dovetailing search and renaming the subenclosure with fewer neighbors, it follows from the partitioning fact above that the total time spent in splitting over the entire algorithm is $O(n \log n)$. With each split, it is easy to update the decomposition tree in constant time.

To establish (2) we apply a charging argument. As noted earlier, the size of the box-decomposition tree for a set of n points is $O(n)$. Thus there are $O(n)$ boxes in the box-decomposition tree (counting both leaf boxes and boxes corresponding to internal nodes of the decomposition tree) before smoothing. Let B_1 denote this set of boxes. Let B_2 denote the boxes of the tree after smoothing. We will charge each box in B_2 that is split as part of smoothing to a box of B_1. We will show that this can be done so that each box in B_1 is charged at most a constant number of times. Since each split results in the creation of two new boxes, this will complete the proof.

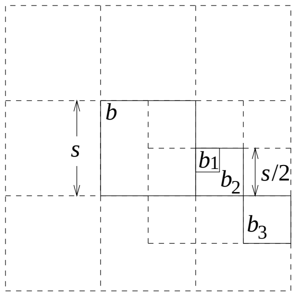


Figure 8: Charging scheme.

Let b be a box of B_2, and let s be the size of b. Consider the nine *local boxes*, consisting of b itself and the eight surrounding boxes whose size and shape are equal to b's. (The local boxes resemble the buffer zone described earlier.) If b is split as a result of smoothing, we make the assertion that b exists in B_2 either because b is a node in B_1 or because at least one of b's local boxes is in B_1 and contains a small box that eventually induced b to split as a part of smoothing.

Assuming the assertion for now, we show how to complete the proof. We charge each box in B_2 that is split to this local box in B_1. Since each box in B_1 can be assessed charges only by itself and the surrounding 8 boxes, it follows that each box of B_1 is charged at most 9 times, and hence $|B_2| \leq 9|B_1|$.

All that remains is to prove the assertion. We prove this by induction on the size of the enclosures, working from smaller to larger enclosures. The reason that b was split is either (1) because b was already split in B_1 or (2) because there is a neighboring enclosure of size no greater than $s/4$ that induced b to split in smoothing. In the first case, the claim is trivially true. In the second case, consider the neighboring box b_1 that induced b to split (see Fig 8).

Recall that if a smaller box is created as part of a buffer zone or as part of shrinking, then the smoothing rules would not allow such a box to induce a larger neighboring box to be split. Thus, starting with the nearest common ancestor of b and b_1, the path to b_1 in the decomposition tree can only consist of splits. At some point along this decomposition path, there must exist an ancestor b_2 of b_1 that is a 1/2-scale copy of b. Clearly b_2 is a neighbor of b.

By the induction hypothesis, either b_2 is a node in B_1 or at least one of b_2's local boxes is in B_1, and contains a small box that eventually induced b_2 to split as a part of smoothing. In the former case, we are done, because then an ancestor of b_2 is a local box of b_1. Otherwise, let b_3 be this local box of b_2. As above, we can argue that b_3 arose along a decomposition path containing only splits. Since b_2 and b_3 are both 1/2-scale copies of c, and they share a common boundary point with each other, it follows that b_3 has an ancestor in B_1 among b's local boxes. This completes the proof of the assertion. $\square$

The bound on the constant factor arising in the previous proof is not tight. Tight bounds on the size of smoothed quadtree subdivisions under various smoothing rules have been established by Moore.[28] However, our result does not follow from Moore's analysis because of shrinking.

Lemma 8 *After smoothing, the number of neighbors of any basic enclosure is* $O(1)$.

Proof. Consider a basic enclosure c of size s. Each basic enclosure is formed from an outer and possibly an inner box. Hence c has at most eight sides. First consider neighbors incident to the outer box. There can be at most a constant number of neighbors whose size is within a constant factor of s. If a neighbor is significantly smaller, then we claim that this neighbor is a buffer box, which was added after the initial point decomposition. To see this, observe that the neighbor could not be generated by a series of splits alone, for otherwise smoothing would induce c to split. Also the neighbor could not be an inner box of the initial decomposition, for otherwise its buffer zone would extend into c, causing c itself to be decomposed. Thus, this neighbor is a buffer box. It is easy to see (Figure 9(a)) that, no matter how densely points are arranged within an inner box, after the smoothing process has terminated, the outer side of each buffer box can be split at most once. Furthermore, an outer box can have at most a constant number of buffer box neighbors. A worst-case example is shown in Figure 9(b), where each of the neighboring enclosures of half the size of the outer box, and each contributes 3 buffer boxes along the common boundary.

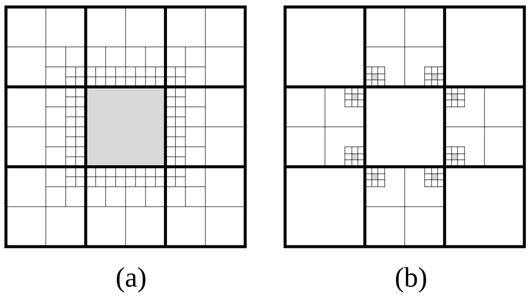


Figure 9: Propagation of smoothing and number of neighboring enclosures.

Second, we consider neighbors of c's inner box. We claim that these neighboring enclosures consist of a constant number of buffer boxes. If c's inner box had any points, then c would have an inner box in the initial decomposition, and so this inner box would have been surrounded by eight buffer boxes. On the other hand, if c had no inner box after the initial decomposition, then the only way that it could acquire an inner box is because the neighbors incident to c's outer box had buffer boxes that spilled over into c. We have argued that c's outer box has at most a constant number of neighbors, and hence there can be at most a constant number of buffer boxes that spilled over into c. As before, the outer edge of each of these boxes can be split at most once, implying that each side of c's inner box is incident to a constant number of enclosures. $\square$

The worst-case bound suggested by this proof is quite pessimistic. The dual graph of the subdivision is a planar graph, implying that each enclosure on average has less than 6 neighbors. From the proof we know that splitting induced from an inner box can cause the outer edge of each buffer box to be split at most once. If the outer edge of a buffer box edge is split more than once in smoothing, then we know that this splitting has originated from some other source, and is allowed to propagate outside the buffer zone.

Because of the doughnuts, basic enclosures may not be convex; indeed, the vertices of an inner box of a doughnut are reflex vertices. We can make the overall subdivision convex by extending the left and right (vertical) sides of each inner box until they contact the boundary of the outer box. This subdivides each doughnut into four rectangles. Observe that this increases the stabbing number of any directed segment by at most a constant factor, and so has no significant effect on the complexity arguments to be made later. This also has a nice effect of reducing each shrink operation to 4 horizontal/vertical split operations (although the splits do not pass through the midpoint of the enclosures). Thus, the resulting subdivision is a recursive binary space partition, using only horizontal and vertical cuts.

Here is a summary of the construction up to this point.

(1) Build a box-decomposition tree for the point set consisting of the obstacle vertices.

(2) For each inner box of a shrinking operation in the box-decomposition, create a buffer zone of 8 identical surrounding boxes. Treating the buffer boxes as fat points, build a box-decomposition tree for the union of the obstacle vertices and buffer boxes.

(3) Given the box-decomposition tree for the point set and buffer boxes, build the cross-indexed neighbor lists for each side of each enclosure by a traversal of the tree. Omit from this list inner boxes from neighboring enclosures and buffer boxes from their neighbors outside the buffer zone.

(4) While the priority queue is not empty, smooth the subdivision by performing the following steps until the priority queue is empty.

    (a) Extract the smallest enclosure $c_{\min}$ from the priority queue.

    (b) For each of its larger neighbors c, do the following until the sizes of c and $c_{\min}$ differ by at most a fixed constant factor (e.g. 2).

        (i) Split c along its longest side.

        (ii) Partition the neighbor lists of c by a dovetailing search.

        (iii) Let c' denote the subenclosure with the fewer neighbors. Create neighbor lists for c' by deleting the appropriate elements from the neighbor lists of c and adding them to c'.

        (iv) Access the neighbors of c', and update their neighbor lists appropriately.

        (v) Insert c' into the priority queue. Adjust the priority of c, according to its new size.

        (vi) If c' is the neighbor of $c_{\min}$, then let $c \leftarrow c'$.

(5) Replace each doughnut by at most four boxes by extending the vertical segments passing through the left and right sides of each inner box.

3.3. Adding Obstacle Edges

To complete the description of the decomposition algorithm, we introduce the obstacle edges. The problem with simply adding the obstacle edges to the box-decomposition is that the number of intersections between obstacle edges and box-decomposition edges may be as large $O(n^2)$. To guarantee that the number of intersections is linear in n, we apply a simple trimming procedure, which trims each edge of the smoothed box-decomposition subdivision at its first and last intersection with an obstacle boundary. Let us assume that both the obstacle set and box-decomposition have been preprocessed (by standard means—e.g., trapezoidization, and a point location data structure) to support horizontal and vertical ray-shooting queries in $O(\log n)$ time. Intuitively, one can imagine each vertex of the box-decomposition subdivision firing a bullet through along each of its incident horizontal and vertical edges, until either reaching the end of this edge, or until hitting an obstacle. The edges of the subdivision are then trimmed so only these bullet-path subedges remain. Trimmed edges that lie entirely within obstacle interiors are deleted.

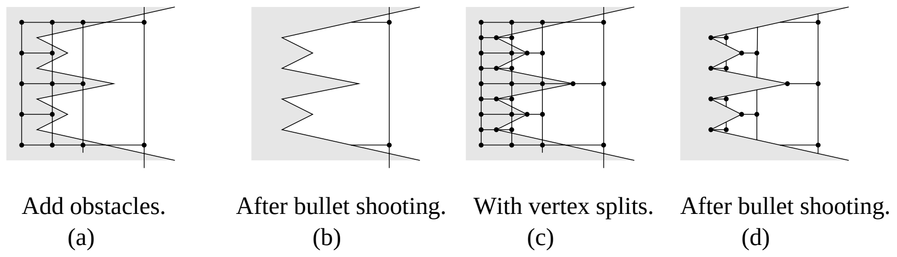


Figure 10: The problem with simple trimming.

Directly applying the simple trimming procedure described above can result in regions of complexity $\Omega(n)$ (see Figure 10(a) and (b)). To deal with this, we apply one additional refinement to the smoothed box-decomposition subdivision, *before* applying the trimming procedure. Recall that each enclosure contains at most one obstacle vertex. For each such enclosure/vertex pair, we split the enclosure by passing a horizontal segment through the vertex. This subdivides the obstacle-free portion of each enclosure into at most 3 convex regions. It also results in the creation of at most two new (subdivision) vertices for each enclosure, and increases the stabbing number of any directed segment by at most a constant factor. This modification of the box-decomposition subdivision can be performed for all n vertices in total time $O(n)$, while increasing the size of the subdivision by only a constant factor. (The resulting transformation is illustrated in Figure 10(c).)

Now apply the above-mentioned trimming procedure for each edge of the modified box-decomposition subdivision. Bullets are shot from both box-decomposition vertices as well as obstacle vertices. (The result is shown in Figure 10(d).) Since this involves shooting $O(n)$ directed (horizontal/vertical) segments, the entire process takes $O(n \log n)$ time. Observe that the regions of the resulting subdivision are simple, and the subdivision has total size $O(n)$, since we have created at most two new edges for each of the $O(n)$ edges in the box-decomposition subdivision.

Within this same time bound we can convert this collection of segments (which includes box-decomposition edges as well as obstacle edges) into a planar subdivision data structure. The 2-faces of the resulting subdivision are called *cells*. Cells are necessarily convex, because all reflex angles have been resolved, and trimming introduces no new reflex angles. Because of the addition of horizontal vertex splitting segments, a cell cannot have two or more consecutive obstacle edges on its boundary except at its topmost and bottommost vertices. The nonobstacle sides of each cell are either horizontal or vertical, and by convexity there can be at most four such sides (although these sides may have embedded vertices from the box decomposition). Therefore each cell is of bounded boundary complexity.

We say that two cells are *neighbors* if they share a common nonobstacle (box-decomposition) edge. Prior to trimming, each enclosure of the smoothed box-decomposition subdivision had a constant number of neighbors. Because the nonobstacle edges of each cell are made up of connected components of box-decomposition edges, it follows that each cell has at most a constant number of neighbors.

A complete example of the process up to now is shown in Figure 11(a) and (b). Part (a) shows the initial smooth box-decomposition of the obstacle vertex set. (Object boundaries have not been shown explicitly, since they play no role up to this stage. Dashed segments show how obstacle vertices have been split.) Figure 11(b) shows the decomposition after trimming. Vertices from which bullet shooting has originated are drawn with solid dots.

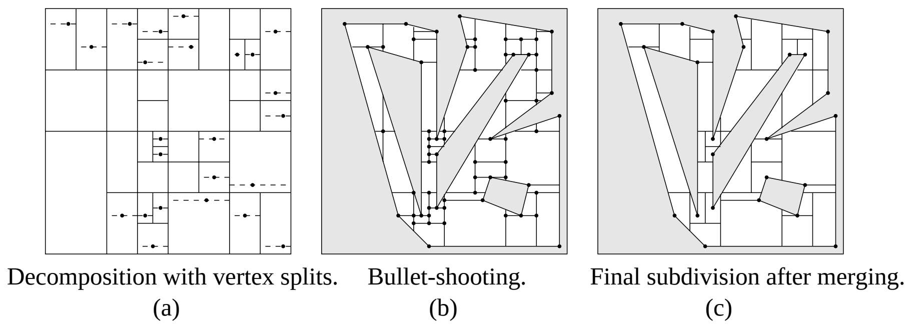

Decomposition with vertex splits.
(a)
Bullet-shooting.
(b)
Final subdivision after merging.
(c)

Figure 11: The final trimming procedure.

If we stop at this point, we claim that the subdivision constructed so far has the property that any segment s stabs as many cells of the subdivision as the simple cover complexity of s. This follows from arguments presented in Section 2.2, combined with the observations that shrinking and trimming do not increase the stabbing number of a segment, and the various refinements made to the subdivision to establish convexity increase stabbing number by at most a constant factor. To strengthen these results to hold for C-simple cover complexity, we present a modification of the merging procedure given in Section 2.2.

At this stage of the algorithm, the cells satisfy the three basic properties introduced in Section 2.2: (1) bounded boundary complexity, (2) bounded neighborhood complexity, and (3) relative convexity. (The actual bounds on the boundary and neighborhood complexity will differ from those in Section 2.2, but they are still constants.) Because of the addition of horizontal vertex splitting segments, cells are actually convex at this point, but we will need to sacrifice this property in order to perform merging. It is easy to restore convexity after merging has been completed.

Recall that prior to the additional obstacle edges, the decomposition is essentially a hierarchical binary space partition, using horizontal and vertical cuts. After trimming, every nonobstacle edge in the current subdivision results from trimming an edge in this subdivision. The process of trimming has disturbed the natural association between cells and boxes of the box-decomposition. For example, because of trimming, a single corridor-like cell lying between two long parallel obstacle edges may cut through an arbitrarily large number of boxes of the box decomposition, and similarly a single box of the decomposition may be intersected by an arbitrarily large number of such thin corridors. In general, each cell is a connected component of the union of some number of boxes in the box-decomposition

The merging procedure operates in a similar manner as the one presented in Section 2.2. Initially all cells are eligible for merging. The box-decomposition splitting segments are traversed in a bottom-up fashion (starting with the leaves of the hierarchy). Each splitting segment has at most two components in the tree (one shot from each endpoint). For each component of each splitting segment consider the two incident cells on either side of this segment. If both cells are eligible for merging, and the union of the two cells satisfies the three basic properties, then the cells are merged into a single cell (effectively erasing the splitting segment). As mentioned earlier, it suffices to test only the first two basic properties, because the third follows as a consequence of the merging process. Otherwise the cells are not merged, and neither cell is eligible for subsequent merging. Because cells are of constant complexity, merging can be performed in time proportional to the size of the subdivision, which is $O(n)$.

We call this final subdivision the *reduced box-decomposition subdivision* for the set of obstacles. (See Figure 11(c) for an illustration of this subdivision.) The final step of preprocessing is to compute a point location data structure for this subdivision. Here is a summary of the construction described in this section.

(1) Given the box-decomposition tree for the obstacle vertex set, augment this decomposition by adding a horizontal splitting segment through each vertex.

(2) Preprocess the obstacle set for horizontal and vertical bullet-shooting queries by computing a horizontal and vertical trapezoidization and a point location structure for each.

(3) For each edge of the augmented subdivision, shoot a bullet along the edge inward from each of its endpoints. Create a subdivision consisting only of the obstacle edges, and the obstacle-free bullet-path subedges.

(4) Mark every cell of this subdivision as eligible for merging.

(5) Working from the bottom of the decomposition hierarchy to the top, for each component of each splitting segment, if the two cells incident to the splitting segment are both eligible for merging, then consider the union of these two cells.

    (a) If this merged cell is incident to at most a given constant number of obstacle edges, and to a given constant number of neighboring cells then replace the two cells with their union.

    (b) Otherwise, mark both cells ineligible for further merging.

(6) Compute a point location data structure for the resulting subdivision.

3.4. Analysis

In this section we show that the number of cells that a query segment stabs in the reduced box-decomposition subdivision is proportional to the C-simple cover complexity of the segment. The absence of the hierarchical *k*-*d* structure makes it difficult to generalize Lemma 5, which was used for the reduced *k*-*d* subdivision. Our analysis will be based on showing that each cell of the reduced *k*-*d* subdivision presented in Section 2.2 intersects at most a constant number of cells in the reduced box-decomposition subdivision presented above. Then, the desired result will follow from the stabbing bounds on the reduced *k*-*d* subdivision established in Theorem 2.

Lemma 9 *Consider a set P of disjoint polygonal obstacles in the plane, and two subdivisions: the reduced k-d subdivision S, and the reduced box-decomposition subdivision S', both starting with the same bounding square. Then each cell of S intersects a constant number of cells S'.*

Proof. We assume for concreteness that in both subdivisions, cells are merged only if the number of incident obstacle edges is at most two. A cell may generally have (a constant number) more than two incident edges before merging, but such a cell may not acquire any more incident edges through merging.

Consider a cell c of S. Let b be the box in the *k*-*d* tree associated with c. Recall from Section 2.2 that c is a connected component of b. (Figure 12(a) shows the reduced *k*-*d* subdivision S and (b) shows the reduced box-decomposition subdivision S'. We have indicated that box b has been further subdivided due to the presence of obstacles that are not shown in the figure. Note that in general the two subdivisions will not be the same.)

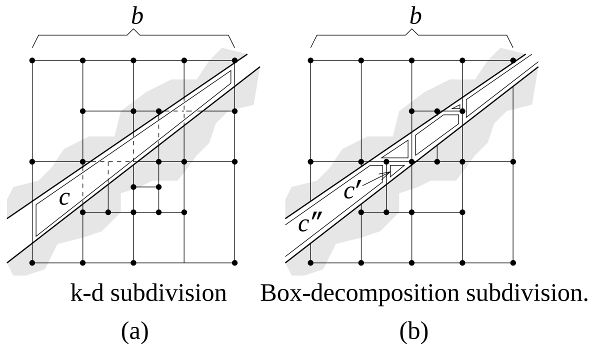


Figure 12: Proof of Lemma 9.

First we consider whether box b exists in S'. Because both subdivisions started with the same bounding square, either (1) b is a box in the box-decomposition, (2) b is contained within the outer box of some shrinking decomposition, or (3) b is contained within some leaf box. Case (3) can be reduced to case (2) by thinking of each leaf box as an outer box with no inner box. We show how to reduce case (2) to case (1). Recall that the doughnut region of each shrinking operation is partitioned into at most 4 rectangular cells to resolve the nonconvex angles of the doughnut. Because c is incident to at most two obstacle edges, it follows that the intersection of c with the doughnut consists of only a constant number of connected components. The processes of trimming and merging can only reduce this number further. Thus, the S' cannot have more than $O(1)$ cells that overlap the intersection of c with the doughnut. To complete the proof it suffices to consider the cells of S' that overlap the inner box, which is a box in both subdivisions. If the inner box does not intersect c then we are done. Otherwise, let us restrict b to be this inner box, and restrict c to its intersection with b.

At this point we may assume that b is a box in both the *k-d* tree subdivision and in the box-decomposition, and the cell c of the reduced *k-d* subdivision is a connected component of b incident to at most two obstacle edges. Consider the box-decomposition tree prior to merging, and consider any cell of this subdivision that intersects c.

We consider two cases. First, if the cell of the trimmed subdivision is partially exterior to b, then it follows that this cell intersects the boundary of b along a splitting segment that contains no vertices of the box-decomposition. (For example, c'' in Figure 12(b).) This is true because the existence of a single vertex of the box decomposition along the intersection region would result in a bullet firing along this edge. Since there are at most two obstacle edges intersecting c, there can be at most two untrimmed segments along the intersection of c with the boundary of b, and hence there can only be at most two such partially exterior cells.

In the second case, the cell of the trimmed subdivision, denoted c', is entirely interior to b (see Figure 12(b)). We claim that after merging, all of these cells will be merged, leaving at most one such cell (implying that the situation shown in Figure 12(b) cannot occur.) Among all the cells that are entirely interior to b, select c' to be any one that is separated from a neighbor by the lowest level edge in the box-decomposition tree. Because this edge is at the lowest level, it follows that the properties of bounded neighborhood size and partial convexity cannot be violated by the merger. (The bounded neighborhood size can only be violated when neighbors are significantly small, implying the existence of a lower level neighboring edge. Partial convexity is a simple consequence of the facts that merging is performed in a bottom-up manner in the decomposition tree.) We argue that bounded boundary complexity cannot be an impediment to merging either. To see this, first note that if both cells are entirely interior to b, then because c is incident to only two edges, the merger contains only two obstacle edges, and hence cannot violate the complexity bound. On the other hand, suppose the neighbor of c', denoted c'', is partially exterior to b (as in Figure 12(b)). As noted above, c'' can be partially exterior to b only if it used both of the obstacle edges of c to trim part of b's boundary. Since c' is entirely interior to b, it can intersect only a subset of the edges that c'' intersects, and hence their merger does not violate the complexity bound. $\square$

Combining this result with Theorem 2, it follows immediately that if a segment s intersects no obstacles, then it intersects $O(cscc(s))$ cells in the decomposition. From the comments made throughout the presentation, it follows that the total size of the subdivision is $O(n)$ and the total preprocessing time is $O(n \log n)$.

Query processing involves two steps. Given a directed segment s, we first apply a point location algorithm to determine the cell containing the origin of the segment in $O(\log n)$ time. Once this cell is known, the algorithm simply walks from one cell to the next until either reaching the destination of the segment, or until encountering an obstacle edge. This can be done in time proportional to the number of cells traversed, since each cell is bounded by a constant number of edges and has a constant number of neighboring cells. Since the traced portion of the query segment intersects no obstacles, the time needed to trace the segment is $O(cscc(s))$. Therefore, we have established Theorem 3.

4. Conclusions

We have presented a query-sensitive approach to segment shooting queries, where the running time of the algorithm is described in terms of an intrinsic geometric property of the query segment, called simple cover complexity, which intuitively captures the inherent complexity of processing the query. We feel that a query-sensitive analysis can provide a more accurate insight into the running time of an algorithm than may be available from a worst-case analysis.

One of the major challenges of this type of analysis is the methodological problem of selecting parameters that do a good job of identifying the natural sources of complexity in a query. Obviously, there is the danger of first deriving a poor algorithm and then attempting to justify its performance in terms of some unnatural parameter. For geometric problems, parameters should be invariant under geometric transformations under which the problem itself is invariant. Parameters should intuitively reflect the complexity of the query, and should be small for "typical" cases in practice. For example, it should be clear to a potential user of the algorithm whether a particular class of inputs of interest will likely produce unacceptably large parameter values. Also, parameter values should be reasonably robust to small perturbations in input coordinates.

Although we feel that the complexity measure chosen here does a good job of capturing the intuitive complexity of ray shooting, there are two respects in which our complexity measure could be improved. First, the "strongness" in the cover complexity (parameterized by ϵ) is an obvious source of unwanted complexity in the definitions. Although we showed that the actual value of $\epsilon > 0$ is irrelevant to the asymptotic analysis, it would be desirable to dispense with it altogether. The second shortcoming is that, although the measure is invariant under rigid geometric transformation, it is not invariant under general nonsingular affine transformations. In general it can be shown that, the degradation in the parameter value is proportional to the "thinness" of the image of the unit sphere under such a transformation. However, the ray shooting problem is invariant under all nonsingular affine transformations. It would be interesting to develop an approach that is sensitive to a natural parameter that is invariant under all such transformations.

Another interesting generalization is that of directed curve shooting. This problem has application in the area of motion simulation. Obviously the method presented here is easy to modify for tracing curves, given access to a primitive that determines the earliest intersection of the curve and a line segment (or generally a $d-1$ dimensional polyhedron of bounded complexity). One property of lines that we use implicitly in our analysis is that the obstacle-free portion of the query segment can intersect each convex cell of the subdivision in a single connected component. If the curves are sufficiently simple that a curve intersects a convex polyhedral cell of bounded complexity in a constant number of components, then all the results of this paper apply to these curves as well.

Acknowledgements

We thank Robin Flatland for spotting a bug in the decomposition algorithm, and the referees for their detailed comments and suggestions, which significantly improved the paper.

References

1. P. K. Agarwal. A deterministic algorithm for partitioning arrangements of lines and its applications. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 11–22, 1989.
2. P. K. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in three dimensions. In *Proc. 11th ACM Sympos. Comput. Geom.*, pages 267–276, 1995.
3. P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.
4. P. K. Agarwal and J. Matoušek. On range searching with semi-algebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.
5. P. K. Agarwal and M. Sharir. Ray shooting amidst convex polytopes in three dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 260–270, 1993.
6. P. K. Agarwal and M. Sharir. Applications of a new space-partitioning technique. *Discrete Comput. Geom.*, 9:11–38, 1993.
7. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.
8. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.
9. M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1993.
10. M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 21–30, 1991.
11. M. Bern. Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45:95–99, 1993.
12. M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 231–241, 1990.
13. P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 546–556, 1992.

14. P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest-pair and n-body potential fields. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 263–272, 1995.

15. B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. 18th Internat. Colloq. Automata Lang. Program.*, volume 510 of *Lecture Notes in Computer Science*, pages 661–673. Springer-Verlag, 1991.

16. B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 135–146, 1985.

17. S. W. Cheng and R. Janardan. Space-efficient ray shooting and intersection searching: Algorithms, dynamization and applications. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 7–16, 1991.

18. S. W. Cheng and R. Janardan. Algorithms for ray-shooting and intersection searching. *J. Algorithms*, 13:670–692, 1992.

19. K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983

20. C. R. Dyer. The space efficiency of quadtrees, *Computer Graphics and Image Processing*, 19:335–348, 1982.

21. A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press, San Deigo, CA, 1989.

22. M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths via balanced geodesic triangulations. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 318–327, 1993.

23. M. Held, J. T. Klosowski, and J. S. B. Mitchell. Evaluation of collision detection methods from virtual reality fly-throughs. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 205–210, 1995.

24. J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

25. B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics*, 21(4):103–110, July 1987.

26. G. M. Hunter and K. Steiglitz. Operations on images using quad trees, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1:145–153, 1979.

27. S. A. Mitchell and S. A. Vavasis. Quality mesh generation in three dimensions. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 212–221, 1992.

28. D. W. Moore. *Simplicial mesh generation with applications*. Ph.D. thesis, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, 1992. Report 92-1322.

29. M. Overmars, H. Schipper, and M. Sharir. Storing line segments in partition trees. *BIT*, 30:385–403, 1990.

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

31. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

32. O. Schwarzkopf. Ray shooting in convex polytopes. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 286–295, 1992.

33. F. X. Sillion and C. Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, 1994.

34. P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.