Shooting Permanent Rays among Disjoint Polygons in the Plane

Mashhood Ishaque* Bettina Speckmann† Csaba D. Tóth‡

Abstract

We present a data structure for ray shooting-and-insertion in the free space among disjoint polygonal obstacles with a total of n vertices in the plane, where each ray starts at the boundary of some obstacle. The portion of each query ray between the starting point and the first obstacle hit is inserted permanently as a new obstacle. Our data structure uses $O(n \log n)$ space and preprocessing time, and it supports m successive ray shooting-and-insertion queries in $O(n \log^2 n + m \log m)$ total time. We present two applications for our data structure:

(1) Our data structure supports efficient implementation of *auto-partitions* in the plane i.e. binary space partitions where each partition is done along the supporting line of an input segment. If n input line segments are fragmented into m pieces by an auto-partition, then it can now be implemented in $O(n \log^2 n + m \log m)$ time. This improves the expected runtime of Patersen and Yao's classical randomized auto-partition algorithm for n disjoint line segments to $O(n \log^2 n)$.

(2) If we are given disjoint polygonal obstacles with a total of n vertices in the plane, a permutation of the reflex vertices, and a half-line at each reflex vertex that partitions the reflex angle into two convex angles, then the folklore convex partitioning algorithm draws a ray emanating from each reflex vertex in the prescribed order in the given direction until it hits another obstacle, a previous ray, or infinity. The previously best implementation (with a semi-dynamic ray shooting data structure) requires $O(n^{3/2-\varepsilon/2})$ time using $O(n^{1+\varepsilon})$ space. Our data structure improves the runtime to $O(n \log^2 n)$.

1 Introduction

Ray shooting data structures are a classical core component of computational geometry. They preprocess a set of objects in space such that one can efficiently find the first object hit by a query ray. A simple polygon with n vertices can be preprocessed in $O(n)$ time to answer ray shooting queries in $O(\log n)$ time using either a linear-time triangulation [16], a balanced geodesic triangulation [9], or a Steiner triangulation [18]. However, the free space among disjoint polygonal obstacles with a total of n vertices (e.g., $n/2$ disjoint line segments) cannot be handled so easily: The best ray shooting data structures can answer a query in $O(n/\sqrt{m})$ time using $O(m)$ space and preprocessing, based on range searching data structures via parametric search. That is, each query takes up to $O(\sqrt{n})$ time using $O(n)$ space. Refer to [1, 2, 26] for higher dimensional variants and special cases.

Geometric algorithms often rely on a ray shooting data structure, where the result of each query may affect the course of the algorithm and modify the data. The current best dynamic data structure, due to Goodrich and R. Tamassia [15], uses $O(n \log n)$ space and preprocessing time and supports ray shooting queries, segment insertions and deletions in $O(\log^2 n)$ time, if the free space around the obstacles consists of *simply connected* faces. However, for the free space around disjoint obstacles (equivalently, for a polygon with holes), the current best data structure uses dynamized range searching data structures via parametric search, which can answer a query in $O(n/\sqrt{m})$ time using $O(m)$ space and preprocessing.

Results. We present a data structure for ray shooting-and-insertion queries among disjoint polygonal obstacles in the plane. Each query is a point p on the boundary of an obstacle and a direction d_p; we report

*Department of Computer Science, Tufts University, Medford, MA, USA, mishaq01@cs.tufts.edu. Partially supported by NSF grants CCF-0431027 and CCF-0830734.

†Department of Maths. and Computer Science, TU Eindhoven, the Netherlands, speckman@win.tue.nl. Supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.022.707.

‡Department of Mathematics and Statistics, University of Calgary, AB, Canada, cdtoth@ucalgary.ca. Research by Cs. D. Tóth was done while visiting Tufts University. Supported in part by NSERC grant RGPIN 35586.

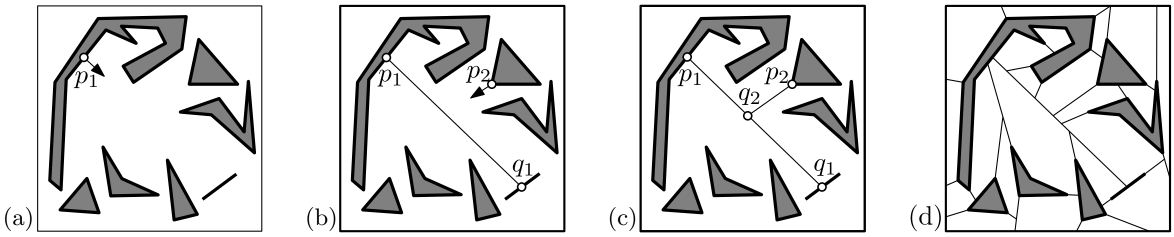

Figure 1: (a) Disjoint polygonal objects in the plane; (b) first ray shooting-and-insertion query at p_1; (c) second ray shooting-and-insertion query at p_2; (d) a convex partition of the free space.

the point q where the the ray emanating from p in direction d_p hits the first obstacle (ray shooting) *and* insert the segment pq as a new obstacle edge (Fig. 1). If the input polygons have a total of n vertices, our data structure uses $O(n \log n)$ preprocessing time, and it supports m ray shooting-and-insertion queries in $O(n \log^2 n + m \log m)$ total time in the real RAM model of computation. The worst case time bound for a single ray shooting-and-insertion query, however, is $O(n^{\frac{1}{2}+\delta} + \log m)$ for any fixed $\delta > 0$.

Applications. Successive ray shooting queries are responsible for a bottleneck in the runtime of geometric algorithms which recursively partition the plane along rays (along the portion of rays between their starting points and the first obstacles hit, to be precise). Two specific applications are:

(1) Computing binary space partitions for disjoint line segments. An *auto-partition* for a set L of disjoint line segments in the plane is a recursive decomposition of the plane into convex cells. Each step partitions the plane along the supporting line of an input segment and recurses on the segments clipped in each open halfplane [11]. We can implement auto-partition by inserting the partition lines as new barriers. To partition along the supporting line of segment $\ell \in L$, shoot rays from the endpoints of ℓ, and whenever a ray hits another segment ℓ', shoot a new ray from the opposite side of ℓ' in the same direction. An auto-partition that partitions the input segments into m fragments requires $O(m)$ ray shooting-and-insertion queries. In particular, with our structure the runtime for the deterministic auto-partition that fragments n disjoint line segments into $O(n \log n / \log \log n)$ pieces [29] can be implemented in $O(n \log^2 n)$ time; and the runtime of an earlier auto-partition that fragments n disjoint line segments with $k \leq n$ distinct directions into $O(nk)$ pieces [30] improves from $O(n^2 k)$ to $O(nk \log^2 n)$. The classical randomized auto-partition algorithm by Patersen and Yao [25], which partitions along the supporting lines of the input in a random order and produces an expected $O(n \log n)$ fragments, can be implemented in $O(n \log^2 n)$ time. An implementation with a "somewhat disappointing" runtime of $O(n^2 \log n)$ was presented in [11], however, Mark de Berg noted [12] that one can also compute an auto-partition of expected size $O(n \log n)$ in $O(n \log^2)$ time by incrementally inserting the line segments, using Cheng and Janardan's dynamic point location data structure.

(2) Computing convex partition of the free space around disjoint polygons. Given a set of disjoint polygons in the plane with a total of n vertices, a permutation of the reflex vertices of the free space, and a half-line at each reflex vertex that partitions the reflex angle into two convex angles. Process the reflex vertices in the specified order. From each reflex vertex, draw a segment emanating from the vertex along the given ray until it hits another obstacle, a previously drawn segment, or infinity.

This folklore convex partition algorithm splits every reflex angle into convex angles, hence it partitions the free space into convex faces. If the algorithm can choose the permutation of the reflex vertices, then it is easy to compute *a* convex partition in $O(n \log n)$ time: First process simultaneously all rays pointing to the right in a left-to-right line sweep, if two segments along the rays meet, give priority to one over the other arbitrarily; then process similarly all the rays pointing to the right in a right-to-left sweep. However, in many applications [5, 6, 19], the order of the reflex vertices and the rays is given online. If the permutation of the reflex vertices is given, then the best previously known data structure requires $O(n^{3/2-\varepsilon/2})$ time using $O(n^{1+\varepsilon})$ space. Our data structure improves the runtime to $O(n \log^2 n)$ and uses $O(n \log n)$ space.

Techniques and Organization. The biggest challenge for the design of our data structure was bridging the gap between the $O(\log n)$ query time in simple polygons and the $O(\sqrt{n})$ query time among disjoint obstacles. Our structure is based on two tools which we describe in detail in the beginning of Section 2: Geometric partition trees in two dimensions and geodesic hulls, which are a generalization of convex hulls in the presence of obstacles. In the remainder of Section 2 we introduce the types of polygons we use in our tiling of the free space between the obstacles. In a nutshell, our data structure, which we discuss in Section 3, works as follows. In each convex cell of a geometric partition tree, we maintain the geodesic hull of all reflex corners. The boundary of this geodesic hull separates the obstacles lying in the interior of the cell from all other obstacles. The geodesic hulls form a nested structure of weakly simple polygons, which creates a tiling of the free space. Each tile is a simple polygon that can easily be processed for fast ray shooting queries. A ray shooting query can be answered by tracing the query ray along these polygons.

The use of geodesic hulls allows us to control the total complexity of m ray insertion queries. Basically, a query ray intersects a geodesic hull only if it partitions the set of incident reflex vertices into two nonempty subsets. Since a set of k points can recursively be partitioned into nonempty subsets at most $k-1$ times, we can charge the total number intersections between rays and geodesic hulls to the number of such partition steps. We detail this analysis in Section 4.

Related work. For a set of points in the plane, Overmars and van Leeuwen's [24] fully dynamic *convex hull* data structure as well as the semi-dynamic data structures of Chazelle [8], and Hershberger and Suri [17] rely on a binary hierarchy of nested convex hulls. This idea was recently extended to a semi-dynamic data structure for *geodesic hulls* that supports point deletion and obstacle insertion [20], but only in the special case that every face in the free space is *simply connected*, and using additional polylogarithmic factors. A tiling of the free space between disjoint polygons in the plane can serve as a proof that the polygons do not intersect. If the tiling is easily maintainable as the polygons move, then it can be the basis for kinetic algorithms for collision detection. Kirkpatrick and Speckmann [28, 21] and independently Agarwal *et al.* [3, 7] developed kinetic data structures that are based on tilings with *pseudo-triangles*–simple polygons with exactly three convex angles. The theoretical study of geodesics in the interior of a simple polygon was pioneered by Toussaint [31, 32]. He showed that the geodesic hull $\mathrm{gh}_P(S)$ of a set S of n points in a simple n-gon P can be computed in $O(n \log n)$ time, and any line segment in the interior of P crosses at most two edges of $\mathrm{gh}_P(S)$. Mitchell [23] and Ghosh [14] survey results on geometric shortest paths.

Dynamic ray-shooting in simple polygons. Chazelle *et al.* [9] showed that a balanced geodesic triangulation of a polygon with n vertices can be used to answer ray shooting queries in the polygon in $O(\log n)$ time. Goodrich and Tamassia [15] generalized this data structure to dynamic subdivisions defined by noncrossing line segments where each face is a *simple polygon*. They maintain a balanced geodesic triangulation of each face. For m segments, the data structure has $O(m)$ size. Each segment insertion and deletion, point location, and ray shooting query takes $O(\log^2 m)$ time.

2 Preliminaries

Geometric partition trees are at the core of many hierarchical data structures. A *geometric partition tree* for n points in a bounding box $B \subset \mathbb{R}^d$ in d-space is a rooted tree T of bounded degree where (1) every node u corresponds to a convex cell C_u in $\mathbb{R}^d$; (2) the root, at level 0, corresponds to B; (3) the children of every node u correspond to a subdivision of C_u into convex cells; and (4) every cell C_u, $u \in T$, at level k of T contains at most $n/2^k$ points. The convex cells C_u, for all leaf nodes $u \in T$ form a subdivision of the bounding box B. In particular, in a *binary* geometric partition tree, for every non-leaf node $u \in T$, the cell C_u is partitioned into two convex cells along a hyperplane.

The simplest geometric partition tree for a set S of n points in the plane is a binary partition of the point set by vertical lines. All cells C_u are vertical slabs; if $|S \cap C_u| \geq 2$, then C_u is subdivided into two slabs by a vertical line through the x-median of $S \cap C_u$. This *vertical slab* partition tree can be constructed in

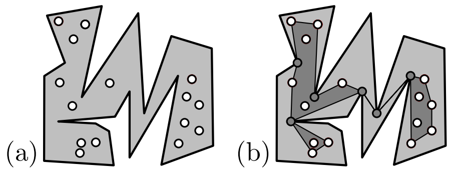

Figure 2: (a) A point set S in a simply connected polygonal domain D; (b) the geodesic hull $\mathrm{gh}_D(S)$.

$O(n \log n)$ time, and it was used for dynamic convex hull computation [8] and many other early dynamic data structures [24].

Chazelle *et al.* [10, 4] constructed geometric partition trees with *low stabbing number* in $O(n \log n)$ time. In the plane, there is a geometric partition tree of size $O(n)$ such that any line stabs at most $O(n^{\frac{1}{2}+\delta})$ cells, for any fixed $\delta > 0$ [22, 33]. It can be represented as a *binary* geometric partition tree if we allow that every cell C_u at level k of T contains at most $n/2^{\Theta(k)}$ points (rather than $n/2^k$ or fewer points), and every cell is a convex polygon with at most $O(1)$ vertices. Agarwal and Sharir [4] noted that this structure can support point insertions and deletions in $O(\log^2 n)$ amortized time using standard dynamization techniques.

Geodesic hulls. The *geodesic hull*, which is also known as *relative convex hull*, was introduced in the digital imaging community [27] and later rediscovered in computational geometry (c.f. [31]). It is a key concept for the best available data structures for motion planning, collision detection [3, 7], and robotics [13]. The geodesic hull is a generalization of the convex hull, restricted to some simply connected domain. For a set S of points in the plane, the *convex hull* is the minimum set that contains S and is convex (i.e., contains the line segment between any two of its points). For a set S of points and a simply connected domain D, the *geodesic hull* is the minimum set that contains $S \cap D$, lies in D, and contains the shortest path with respect to D between any two of its points.

The boundary of the convex hull, denoted by $\mathrm{ch}(S)$, is a simple polygon. The boundary of the geodesic hull, denoted by $\mathrm{gh}_D(S)$, is a *weakly simple polygon* [32] (albeit, not necessarily a simple polygon, Fig 2(b)). A weakly simple polygon on k vertices is a closed polygonal chain $(p_1, p_2, \ldots, p_k)$ such that for any $\varepsilon > 0$ the points p_i can be moved to a location p'_i in the ε-neighborhood of p_i so that $(p'_i, p'_2, \ldots, p'_k)$ is a simple polygon. Toussaint [32] showed that any line segments lying in D intersects the boundary $\mathrm{gh}_D(S)$ in at most two points.

Crescent polygons. Data structures for ray-shooting and collision detection in the plane typically use tessellations of the free space among obstacles by *pseudo-triangles*–simple polygons with exactly three convex angles. We construct a tessellation by *crescent polygons*, which are a superset of *spiral polygons* (polygons with at most one reflex chain). Crescent polygons are bounded by one convex polygonal chain and at most one reflex polygonal chain (Fig. 3). A polygonal chain on the boundary of a polygon P is *convex* (*reflex*) if the interior angle of P is less (more) than π at every internal vertex of the chain. As opposed to spiral polygons, crescent polygons may contain a hole (at most one)–the two polygonal chains can be disjoint. That is, the family of crescent polygons does not only include all spiral polygons (which, in particular, include convex polygons), but also convex polygons with a convex hole (the boundary of the hole is a reflex chain of the polygon).

We build a *monotone decomposition* to store a crescent polygon P. A reflex vertex v of P is y-extremal if both incident vertices are in a closed halfplane bounded by a horizontal line passing through v. For each y-extremal reflex vertex v, shoot a vertical ray in the interior of P, the ray necessarily hits the convex arc

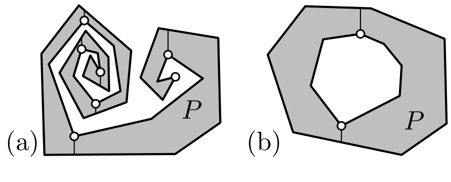

Figure 3: (a) A simple crescent polygon; (b) a non-simple crescent polygon.

of P. The rays partition P into y-monotone subpolygons (Fig. 3). Each subpolygon is bounded by a reflex chain and up to two vertical segments from one side (left or right); and a convex chain from the other side (right or left). We store the edges of the convex and the reflex chain of each subpolygon in a self-balancing search tree; we also store the adjacency relations between the subpolygons. This monotone decomposition has $O(n)$ size for a crescent polygon with n edges and can easily be built in $O(n \log n)$ time.

Proposition 1 *A line segment inside a crescent polygon P intersects at most two subpolygons of P.*

Proof. If the line segment s is vertical, then it is disjoint from the separators between subpolygons and can hence intersect only one subpolygon. Assume that s is not vertical and crosses a vertical separator between two subpolygons P' and P''. Assume further that the separator between P' and P'' is incident to a local *minimum* of the reflex chain of P. Then in both P' and P'', the portion of the reflex chain lies *above* the line through s. Any other separator on the boundary of P' or P'' is incident to a local maximum of the reflex chain, which must also be above the line through s. Hence s cannot cross any other separator on the boundary of P' and P''. □

Proposition 2 *Let P be a crescent polygon with a reflex chain of n_1 edges, a convex chain of n_2 edges, and with a monotone decomposition. A ray shooting query is a triple $(e, p, \overrightarrow{d})$, where e is an edge of P, p is a point on e, and $\overrightarrow{d}$ is a direction. A ray shooting query in the interior of P can be answered in $O(\log n_1)$ time if the ray hits the reflex chain, and in $O(\log n_1 + \log n_2)$ time if it hits the convex chain.*

Proof. By Proposition 1, the ray intersects at most two subpolygons of P. The edge e is adjacent to a unique subpolygon, hence we know which subpolygon the ray starts from. We trace the ray through the subpolygons of P. In each subpolygon P', first detect any intersection with the reflex chain of P'. This takes $O(\log n_1)$ time, since the reflex chain of P' is y-monotone and its edges are sorted by slope. If the ray intersects the reflex chain, then it ends there and we can report the first point hit. Otherwise detect any intersection with the vertical separating segments on the boundary of P' in $O(1)$ time. If the ray crosses a separator, then it leaves to an adjacent subpolygon. Otherwise the ray must hit the convex chain of P'. We can compute the first edge hit in $O(\log n_2)$ time. □

Cavity, exterior, and bridge polygons. Given a finite set $\mathcal{P}$ of disjoint polygonal obstacles in the plane, we define *cavity polygons* to be crescent polygons lying in the free space whose convex chains are the maximal convex chains of the free space (that is, maximal reflex chains along obstacles). See Fig. 4(ab). For a maximal convex chain α of the free space with at least one internal vertex, let a cavity polygon $P(\alpha)$ be the maximal crescent polygon whose convex chain is α. In particular, if α is a closed convex chain that does not contain any obstacle, then $P(\alpha)$ is the convex polygon bounded by α. If α is a closed convex chain that contains a set $\mathcal{Q}_\alpha \subset \mathcal{P}$ of obstacles, then $P(\alpha)$ is a polygon bounded by α and $\mathrm{ch}(\mathcal{Q}_\alpha)$. If α is an open convex chain, then $P(\alpha)$ is bounded by α and the shortest path between the endpoints of α in the

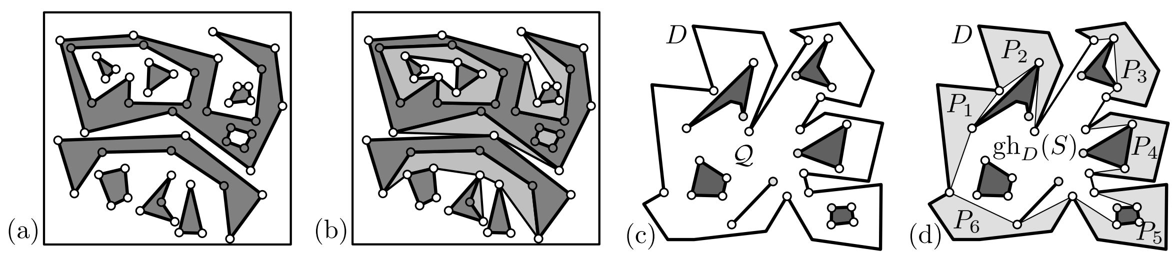

Figure 4: (a) A set of disjoint polygonal obstacles in the plane. Full (empty) circles mark the convex (reflex) vertices of the free space. (b) The cavity polygons. (c) A polygon D containing a set $\mathcal{Q}$ of polygonal obstacles in the interior, Empty circles mark the set S. (d) The exterior polygons of $\mathrm{gh}_D(S)$ are $P_1, \ldots, P_6$.

free space. Cavity polygons are interior disjoint and, intuitively, form a "buffer" around a convex chain of the free space.

Let D be a simple polygon, and let $\mathcal{Q}$ be a set of polygonal obstacles in the interior of D. Denote by S the set of reflex vertices of the free space $\mathrm{int}(D) \setminus (\bigcup \mathcal{Q})$ in the interior of D around the obstacles. D contains the geodesic hull bounded by $\mathrm{gh}_D(S)$. The space in the interior of D and the exterior of $\mathrm{gh}_D(S)$ decomposes naturally into polygons which we call the *exterior polygons* of $\mathrm{gh}_D(S)$. See Fig. 4(cd).

Proposition 3 *Every exterior polygon of $\mathrm{gh}_D(S)$ is a crescent polygon, where the convex chain is a maximal convex chain of D and the reflex chain is a subchain of $\mathrm{gh}_D(S)$.*

Proof. If D is convex, then $\mathrm{gh}_D(S) = \mathrm{ch}(S)$ lies in the interior of D. The boundary of the free space between D and $\mathrm{gh}_D(S)$ has two connected components: An outer boundary D, and a hole $\mathrm{gh}_D(S) = \mathrm{ch}(S)$. Assume that D has $k \geq 1$ reflex vertices. All reflex vertices are in S. The boundary of the geodesic hull $\mathrm{gh}_D(S)$ passes through all k reflex vertices of D, but none of the convex vertices. The two endpoints of any maximal convex chain of D are connected by two polygonal chains: The maximal convex chain of D and a polygonal chain along $\mathrm{gh}_D(S)$. The chain along $\mathrm{gh}_D(S)$ is reflex, since if it had a convex angle at an internal vertex $p \in S$, then the geodesic hull of S would not contain the shortest path between two neighbors of p. If the length of a maximal convex chain of D is at least two, then the two polygonal chains are different, and they bound a simple polygon. Otherwise both chains are reduced to the same line segment, and they do not enclose a simple polygon. $\square$

In our data structure, we have to deal with pairs of adjacent exterior (crescent) polygons, separated by a line. Let a *double-crescent* Q be a simple polygon bounded by at most two convex chains and at most two reflex chains such that the two reflex chains are separated by a line ℓ and ℓ partitions Q into two crescent polygons Q_1 and Q_2 (Fig. 5). Let α_1 and α_2 be the convex chains of Q. The cavity polygons $P(\alpha_1)$ and $P(\alpha_2)$ are interior disjoint and are adjacent to α_1 and α_2. If the cavity polygons $P(\alpha_1)$ and $P(\alpha_2)$ do not fill Q entirely, the space in between them is called the *bridge polygon* of Q.

Proposition 4 *A double-crescent Q has at most one bridge polygon, which is a simple polygon bounded by two reflex chains on opposite sides of line ℓ and two common tangents of the reflex chains of Q.*

Proof. Every internal vertex of the reflex chains of $P(\alpha_1)$ and $P(\alpha_2)$ is a reflex vertex of Q. Since the reflex chains of Q are separated by a line, the overlap of the reflex chain of $P(\alpha_i)$, $i = 1, 2$, and a reflex chain of Q is a continuous polygonal chain. The reflex chain of $P(\alpha_i)$, $i = 1, 2$, consists of subchains of the two reflex chains of Q connected by a their common external tangents, see Fig. 5 (c). If these common tangents are not identical, then $P(\alpha_1)$ and $P(\alpha_1)$ do not fill Q entirely, and the remaining space is bounded by a subchain of each reflex chain of Q (a subchain may also be a single edge or a vertex), and the two common external tangents of the reflex chains. $\square$

Proposition 5 *The common external tangents of the two reflex chains of a double-crescent Q (and hence the partition of Q into $P(\alpha_1)$, $P(\alpha_2)$, and a possible bridge polygon) can be computed in $O(\log n)$ time, where n is the total number of edges in the two reflex chains.*

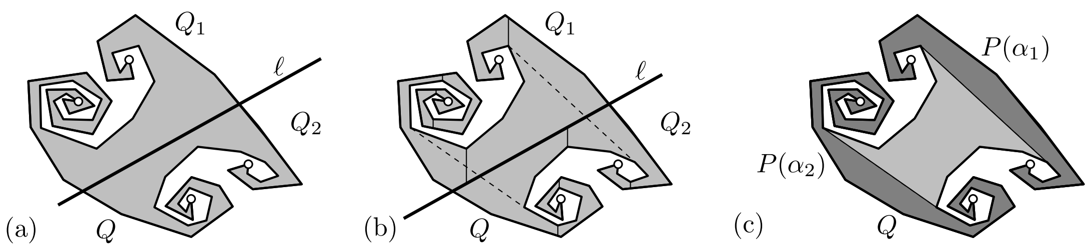

Figure 5: (a) Double-crescent Q partitioned into two crescent polygons; (b) the common external tangents of the two reflex chains in Q; (c) the cavity polygons of the two convex chains of Q and a bridge polygon.

Proof. By Proposition 1, line ℓ, which forms the common boundary of Q_1 and Q_2, intersects at most two subpolygons in each of Q_1 and Q_2. All common tangents of the reflex chains of Q_1 and Q_2 cross ℓ, and so they may intersect the subpolygons along ℓ and adjacent subpolygons in the decomposition of Q_1 and Q_2. The common external tangents of the reflex chains of Q are common external tangents of two reflex chains in some pair of these subpolygons. We have $O(1)$ pairs of subpolygons of Q_1 and Q_2 to consider. In each pair, the common tangents can be computed in $O(\log n)$ time [24], since the reflex chain in each subpolygon is y-monotone. Out of $O(1)$ common tangents of reflex subchains, we can select the common external tangents of the two reflex subchains of Q in $O(1)$ time. $\square$

3 Data structure

In this section, we present our data structure and establish bounds on its space complexity and preprocessing time. Our structure directly gives a tessellation of the free space between the obstacles into cavity polygons and bridge polygons. We can *answer* a ray shooting query by simply tracing the ray through the tessellation. The update of our data structure after *inserting* the query ray as a new obstacle is discussed in Section 4.

Description. We are given a set $\mathcal{P}$ of pairwise disjoint polygonal obstacles with a total of n vertices. Denote by S the set of reflex vertices of the free space (that is, convex vertices of the obstacles). The backbone of our data structure is a binary geometric partition tree T for the point set S, where the root corresponds to a bounding box B containing all obstacles in its interior. Let $F = \text{int}(B) \setminus (\bigcup \mathcal{P})$ denote the free space around the obstacles. The choice of the partition tree has no effect on our results for the total processing time of m queries; it affects only the time required for a single query. Each query is processed in $O((n+m)^{\frac{1}{2}+\delta})$ time for any fixed $\delta > 0$ if we use a geometric partition tree with *low stabbing number*, proposed by Chazelle *et al.* [10]. We augment the geometric partition tree T, and store several structures related to the portion of the obstacles clipped in C_u at each node $u \in T$. At node u, we store the following:

- C_u is the convex cell associated with node $u \in T$; and
- ℓ_u is the line splitting cell C_u into two subcells at a non-leaf node $u \in T$.
- Let $S_u = S \cap C_u$ be the set of reflex vertices of the free space in cell C_u;
- let $\mathcal{P}_u$ denote the set of obstacles that intersect the boundary of C_u;
- let $\mathcal{Q}_u$ denote the obstacles lying in the interior of C_u (Fig. 6 (a));
- let $\mathcal{Q}_u^* \subseteq \mathcal{Q}_u$ denote the obstacles $\mathcal{Q}_u$ crossed by ℓ_u (in particular, obstacles in $\mathcal{Q}_u^*$ are in the interior of cell C_u but not in the interior of any subcell of C_u).

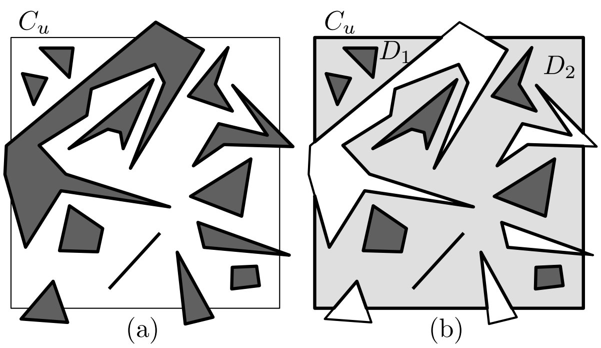

Figure 6: (a) A set $\mathcal{P}$ of disjoint polygonal obstacles clipped in a cell C_u; (b) $C_v \setminus (\bigcup \mathcal{P}_u)$ consists of two simply connected polygonal domains D_1 and D_2; (c) the geodesic hulls $\text{gh}_{D_i}(S)$ for $i = 1, 2$; (d) $\text{gh}_{D_2}(S)$ is subdivided into the geodesic hulls $\text{gh}_{D_2^-}(S)$ and $\text{gh}_{D_2^+}(S)$, the obstacle P lying in the interior of C_u but intersecting line ℓ_u, and some bridge polygons.

At the root u_0, we have $C_{u_0} = B$, $\mathcal{P}_{u_0} = \emptyset$, and $\mathcal{Q}_{u_0} = \mathcal{P}$. In general, the set $C_u \setminus (\bigcup \mathcal{P}_u)$ is the union of the free space clipped in C_u and all obstacles in $\mathcal{Q}_u$. Each connected component of $C_u \setminus (\bigcup \mathcal{P}_u)$ is a *simply connected* polygonal domain (Fig. 6(b)). At node $u \in T$, we store all components of $C_u \setminus (\bigcup \mathcal{P}_u)$ that are incident to some vertex in S using a doubly linked edge list (components that are not incident to any reflex vertex are *not* stored).

- Let $\mathcal{D}_u$ denote the set of the polygonal domains stored at u.

For each $D \in \mathcal{D}_u$, we store $\mathrm{gh}_D(S)$ (Fig. 6 (c)). We also store every exterior polygon of $\mathrm{gh}_D(S)$.

Furthermore, for every non-leaf node, we store the following pointers between some adjacent domains and exterior polygons. If $u \in T$ is a non-leaf node, then cell C_u is split along a line ℓ_u into two subcells, say $C_v = C_u \cap \ell_u^-$ and $C_w = C_u \cap \ell^+$, where $v, w \in T$ are the two children of u. Consider a domain $D \in \mathcal{D}_u$ that intersects ℓ_u and contains vertices in S on both sides of ℓ_u. Note that the sets $D^- = D \cap \ell_u^- \in \mathcal{D}_v$ and $D^+ = D \cap \ell_u^+ \in \mathcal{D}_w$ are not necessarily connected (Fig. 6(d)), it may consists of several components. If a component $\hat{D}$ of D^- or D^+ has a reflex vertex, then it contains a unique domain stored at $\mathcal{D}_v$ or $\mathcal{D}_w$, respectively, which is obtained as $E_v = \hat{D} \setminus (\bigcup \mathcal{P}_v)$ or $E_w = \hat{D} \setminus (\bigcup \mathcal{P}_w)$, after removing the obstacles of $\mathcal{Q}_u$ that intersect ℓ_u. For each domain $D \in \mathcal{D}_v$ (resp., $\mathcal{D}_w$), we maintain a pointer to the domain of $\mathcal{D}_u$ that contains it, and to any adjacent domain stored in $\mathcal{D}_w$ (resp., $\mathcal{D}_v$). For each exterior polygon of each $\mathrm{gh}_{E_v}(S)$, $E_v \in \mathcal{D}_v$, along line ℓ_u, we maintain a pointer to the adjacent exterior polygon on the opposite side of ℓ_u, which is an exterior polygon of $\mathrm{gh}_{E_w}(S)$ for some $E_w \in \mathcal{D}_w$.

Finally, we store a tessellation of the convex hull $\mathrm{ch}(S)$ of all obstacles. The cavity polygons are interior disjoint from the geodesic hulls of any set of reflex vertices; the geodesic hulls computed in nested layers of the hierarchical subdivision are also nested. Therefore, the union of the edges of all cavity polygons and the geodesic hulls $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$, $u \in T$, forms a planar straight line graph, which we denote by G. Since $\mathrm{gh}_{D_0}(S) = \mathrm{ch}(S)$ is the convex hull of all the obstacles in $\mathcal{P}$ at the root $u_0 \in T$, the graph G is a tessellation of the interior of $\mathrm{ch}(S)$. We store G, and we store each cavity polygon with a monotone decomposition as described in Section 2. This completes the description of our data structure.

Structural properties. We next present lemmas on the structure of the tessellation in the exterior and the interior of a geodesic hull $\mathrm{gh}_D(S)$ in a domain $D \in \mathcal{D}_u$, $D \cap S \neq \emptyset$, for a node $u \in T$. Let $u \in T$ be a non-leaf node in our data structure, with children v and w. Let $D \in \mathcal{D}_u$ be a domain incident to at least one reflex vertex in S for some node $u \in T$. Assume that $D \in \mathcal{D}_u$ such that ℓ_u stabs D and $D \cap S$ contains reflex vertices on both sides of ℓ_u. Some of the edges in $\mathrm{gh}_D(S)$ may already be edges of $\mathrm{gh}_E(S)$ for some subdomain $E \in \mathcal{D}_v \cup \mathcal{D}_w$.

Lemma 6 *The following properties hold for a segment pq which is an edge of $\mathrm{gh}_D(S)$ or a cavity polygon along an obstacle in $\mathcal{Q}_u^*$ but which is not an edge of $\mathrm{gh}_E(S)$ for any $E \in \mathcal{D}_v \cup \mathcal{D}_w$.*

1. *pq lies in the union of a domain $E_v \in \mathcal{D}_v$ and a domain $E_w \in \mathcal{D}_w$;*
2. *pq lies in the union of an exterior polygon P_v of $\mathrm{gh}_{E_v}(S)$ and an exterior polygon P_w of $\mathrm{gh}_{E_w}(S)$;*
3. *both P_v and P_w are adjacent to the splitting line ℓ_u and to the boundary ∂D of D;*
4. *pq is the common tangent of the reflex chain of the double-crescent polygon $P_v \cup P_w$.*

Proof. 1. The splitting line ℓ_u partitions domain D into subdomains. Since pq crosses ℓ_u at most once, it can intersect at most two subdomains, at most one on each side of ℓ_u. Denote these domains by $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$.

2. Since the geodesic hulls $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ are nested in $\mathrm{gh}_D(S)$, the edge pq cannot cross any edge of $\mathrm{gh}_{E_v}(S)$ or $\mathrm{gh}_{E_w}(S)$. It must lie in the exterior of both $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Since E_v and E_w are separated by the line ℓ_u, the edge pq can intersect at most one exterior polygon of each of $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Denote these exterior polygons by $P_v \subset E_v$ and $P_w \subset E_w$.

3. If P_v is not adjacent to ℓ_u, then no shortest path between $S \cap D$ can enter the interior of P_v. If P_v is adjacent to ℓ_u but not adjacent to the boundary of D, then it is adjacent to some obstacles in $\mathcal{Q}_u$, which

lie in the interior of D. Since every obstacle in $\mathcal{Q}_u$ must be inside $\mathrm{gh}_D(S)$, the polygon P_v is also inside $\mathrm{gh}_D(S)$. Therefore, if pq lies in P_v, then P_v is adjacent to ℓ_u and to ∂D. The same argument holds for P_w.

4. On the boundaries of P_v and P_w, all vertices of S are in two reflex chains $\beta_v \subset \mathrm{gh}_{E_v}(S)$ and $\beta_w \subset \mathrm{gh}_{E_w}(S)$, respectively. The two chains are separated by the line ℓ_u. Since the geodesic hull of $S \cap D$ with respect to D contains the shortest path between any two points of S, it contains all line segments between the reflex chains on the boundaries of P_v and P_w. The edge pq is on the convex hull $\mathrm{ch}(\beta_v \cup \beta_w)$, and so pq is a common tangent of the arcs β_v and β_w. □

Next, we study the tiling of the interior of a single geodesic hull, bounded by $\mathrm{gh}_D(S)$.

Lemma 7 *The interior of* $\mathrm{gh}_D(S)$ *for a domain* $D \in \mathcal{D}_u$, $u \in T$ *is tiled by the following interior disjoint polygons.*

(i) *geodesic hulls* $\mathrm{gh}_E(S)$, *for some* $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$;
(ii) *obstacles in* $\mathcal{Q}_u^*$ *that lie in* D;
(iii) *cavity polygons along the boundary of obstacles in* $\mathcal{Q}_u^*$ *that line in* D;
(iv) *bridge polygons in the union of any two adjacent exterior polygons of* $\mathrm{gh}_{E_v}(S)$ *and* $\mathrm{gh}_{E_w}(S)$, *with* $E_v \cup E_w \subseteq D$, $E_v \in \mathcal{D}_v$ *and* $E_w \in \mathcal{D}_w$ *on opposite sides of* ℓ_u.

Proof. It is clear that $\mathrm{gh}_D(S)$ contains all geodesic hulls $\mathrm{gh}_E(S)$, for all $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$. Every obstacle in the interior of C_v or C_w is contained in the geodesic hull $\mathrm{gh}_E(S)$ for some $E \in \mathcal{D}_v \cup \mathcal{D}_w$. All obstacles in $\mathcal{Q}_u$ that lie in D are contained in $\mathrm{gh}_D(S)$, but the obstacles in $\mathcal{Q}_u^*$ are not contained in smaller nested geodesic hulls. If an obstacle is contained in $\mathrm{gh}_D(S)$, then all adjacent cavity polygons are also contained in $\mathrm{gh}_D(S)$, since a geodesic hull of reflex vertices cannot separate an obstacle from an adjacent cavity polygon. This shows that the polygons of type (i), (ii), and (iii) are all contained in $\mathrm{gh}_D(S)$. It remains to show that any remaining polygon is a bridge.

Delete all polygons of type (i), (ii), and (iii) from the interior of $\mathrm{gh}_D(S)$, and let P be a connected component of the remaining space. Since P is outside of the geodesic hulls $\mathrm{gh}_E(S)$ for any $E \in \mathcal{D}_v \cup \mathcal{D}_w$, it must be contained in exterior polygons. An exterior polygon of $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$, is a crescent polygon, bounded by a convex chain β and a subchain of $\mathrm{gh}_E(S)$. If β is disjoint from the boundary of C_v, then β is a maximal convex chain along an obstacle in $\mathcal{Q}_u^*$, hence the exterior polygon is a cavity polygon adjacent to an obstacle in $\mathcal{Q}_u^*$. If β touches the boundary of C_u but is disjoint from ℓ_u, then the exterior polygon is exterior to $\mathrm{gh}_D(S)$, too, and it is disjoint from P. So P is contained in exterior polygons of some $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$, that lie in the interior of C_u and are adjacent to ℓ_u.

Consider two adjacent exterior polygons Q_1 and Q_2 on opposite sides of ℓ_u such that P intersects their union $Q_1 \cup Q_2$ (Fig. 5). Let $s \subset \ell_u$ be a common edge of Q_1 and Q_2, and let α_1 and α_2 be the two maximal convex chains along obstacles that contain the endpoints of s. By Proposition 4, the union $Q_1 \cup Q_2$ of the two exterior polygons is covered by the cavity polygons of α_1 and α_2, and a possible bridge polygon R. Since P is disjoint from cavity polygons, we have $P \subseteq R$. The bridge polygon R lies in the free space and it is bounded by two geodesics of $\mathrm{gh}_{E_v}(S)$, $E_v \in \mathcal{D}_v$, and $\mathrm{gh}_{E_w}(S)$, $E_w \in \mathcal{D}_w$; and by two of their common external tangents. R is, therefore, interior disjoint from any polygon of type (i), (ii), and (iii); and so $P = R$. □

Corollary 8 *The free space between two nested layers of geodesic hulls in our data structure is tiled with cavity polygons and bridge polygons.*

Lemma 9 *Let pq be a line segment lying in the free space among the obstacles.*

1. *For every $u \in T$, segment pq intersects at most one domain $D \in \mathcal{D}_u$.*
2. *Segment pq intersects at most two cavity polygons in the tiling G of $\mathrm{ch}(S)$.*
3. *For every $u \in T$, segment pq intersects at most one bridge polygon within $\mathrm{gh}_D(S)$, $D \in \mathcal{D}_u$.*

Proof. 1. If pq contains points $a \in D_a$ and $b \in D_b$, with $D_a, D_b \in \mathcal{D}_u$, then the line segment $ab \subseteq pq$ lies in the convex cell C_u. Since $D_a, D_b \subset C_u$ are separated by obstacles in $\mathcal{P}$, segment ab must traverse an obstacle, contradicting the assumption that pq lies in the free space.

2. Each cavity polygon is bounded by a convex chain along an obstacle P and a reflex chain. If a ray $\overrightarrow{pq}$ enters a cavity polygon P, it enters through its reflex chain and hits the convex chain along P. Since the rays $\overrightarrow{pq}$ and $\overrightarrow{qp}$ can hit at most two convex chains along obstacles, pq intersects at most two cavity polygons.

3. If pq intersects two distinct bridge polygons in the tiling of $\mathrm{gh}_D(S)$ then there is a segment $ab \subset pq$ that connects the points a, b on the boundaries of two distinct bridges, R_1 and R_2, and does not pass through any other bridge in $\mathrm{gh}_D(S)$. The segment ab cannot cross ℓ_u, since all points of ℓ_u in the interior of $\mathrm{gh}_D(S)$ are covered by obstacles, adjacent cavity polygons, and bridges. Hence, ab lies on one side of the line ℓ_u, and so ab lies in a domain $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$. That is, ab connects two points along $\mathrm{gh}_E(S)$, while the relative interior of ab is in the exterior of $\mathrm{gh}_E(S)$. This contradicts the fact that the shortest path between two points of $\mathrm{gh}_E(S)$ passes in the geodesic hull of $S \cap E$ with respect to E. □

Space. A binary geometric partition tree T for n points in the plane has $O(n)$ nodes and $O(\log n)$ height. A node $u \in T$ stores a convex cell C_u, of constant complexity, and a splitting line ℓ_u. Every vertex of a domain $D \in \mathcal{D}_u$ is either a vertex of an obstacle or the intersection of an edge of an obstacle and the boundary of the cell C_u. Since C_u has $O(1)$ edges, a domain $D \in \mathcal{D}_u$ has at most $O(1)$ consecutive vertices that are not vertices of any obstacle. Recall that a domain $D \in \mathcal{D}_u$ is stored only if it contains at least one reflex vertex in S. Every vertex in S lies in at most $O(\log n)$ cells C_u (at most one on each level of T), hence the domains $D \in \mathcal{D}_u$ for all $u \in T$ have a total of $O(n \log n)$ vertices. All vertices of $\mathrm{gh}_D(S)$ are in S, and each vertex $p \in S$ occurs in $O(\log n)$ geodesic hulls (one at each level of T). The total complexity of $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$ and $u \in T$ is $O(n \log n)$.

Preprocessing. A binary geometric partition tree T for n points can be computed in $O(n \log n)$ time, for both the vertical slab tree [24] or the one with low stabbing number [10]. In a top-down traversal of the tree T, we can compute all domains $D \in \mathcal{D}_u$, the sets S_u of reflex vertices lying in C_u, and the sets $\mathcal{Q}_u$ of obstacles in the interior of C_u. At the root u_0, we have $\mathcal{D}_{u_0} = \{B\}$. If a domain $D \in \mathcal{D}_u$ intersects the splitting line ℓ_u, it is split into the domains $\mathcal{D}_v$ and $\mathcal{D}_w$ as follows: Test every edge along the boundary of D and every edge of obstacles lying in the interior of D, whether they intersect ℓ_u. The intersection points of ℓ_u with the edges of ∂D can be sorted along ℓ_u by a simple traversal of the simple polygon D. We can insert into this sorted list each intersection with obstacles in $\mathcal{Q}_u$ in $O(\log n)$ time. By traversing the edges and the portions of ℓ_u between consecutive intersection points, we trace out the subdomains of D. We can discard any subdomain of D that is not incident to any reflex vertex in S. We spend $O(\log n)$ time for sorting edges of the obstacles in $\mathcal{Q}_u$ stabbed by ℓ_u, which were in the interior of a domain but will be on the boundary of subdomains; and we spend $O(1)$ time for each edge each domain $D \in \mathcal{D}_u$. Over $\log n$ levels of T, all domains $D \in \mathcal{D}_u$, sets S_u, and sets $\mathcal{Q}_u$, for all $u \in T$, are computed in $O(n \log n)$ total time.

We compute the cavity polygons and the geodesic hulls $\mathrm{gh}_D(S)$ for all $D \in \mathcal{D}_u$, $u \in T$, in a bottom-up traversal of T. Note that for any polygonal domain $D \in \mathcal{D}_u$, $u \in T$, all reflex vertices of D are in S_u. If $u \in T$ is a leaf, then cell C_u contains exactly one vertex of S, and the geodesic hull $\mathrm{gh}_D(S)$ is a single point. Assume now that $u \in T$ is a non-leaf node with children v and w, the geodesic hull $\mathrm{gh}_E(S)$ has been computed for every subdomain $E \in \mathcal{D}_v \cup \mathcal{D}_w$, $E \subset D$, and we want to compute $\mathrm{gh}_D(S)$. By Lemma 6, we obtain all new edges of $\mathrm{gh}_D(S)$ and any cavity polygon contained in $\mathrm{gh}_D(S)$ by computing the common external tangents of two reflex polygonal arcs along adjacent exterior polygons of $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$. By Proposition 5, each common external tangent can be computed in $O(\log n)$ time. Since G is a planar graph with n vertices and $O(n)$ edges, a total of $O(n)$ common tangents are computed. Hence we can compute all geodesic hulls $\mathrm{gh}_D(S)$ and cavity polygons in $O(n \log n)$ total time.

Finally, the planar straight line graph G has n vertices and $O(n)$ edges. So the total size of all cavity and bridge polygons of G is $O(n)$. We compute a monotone decomposition for each crescent polygon in the tessellation in $O(n \log n)$ total time.

4 Ray Tracing and Update Operations

A single ray shooting-and-insertion query. Given a point p on the boundary of an obstacle and a direction d_p, we answer the associated ray shooting query by tracing the ray through the bridge and cavity polygons of the tessellation G until it hits the boundary of an obstacle or leaves the convex hull $\mathrm{ch}(S)$. By Proposition 2, we can trace a ray through these polygons in logarithmic time in terms of the number of vertices. Recall that after inserting $O(m)$ rays, the number of convex vertices is $O(n+m)$ but the number of reflex vertices remains $O(n)$. Therefore, ray shooting in a cavity or bridge polygon takes $O(\log n)$ time if the ray hits a reflex chain or line ℓ_u; and it takes $O(\log(m+n))$ time if it hits a convex chain. Since convex chains lie on the boundary of obstacles, each ray hits at most one convex chain. So if a ray $p_i q_i$ traverses k faces of the tessellation G, then the ray shooting query takes $O(k \log n + \log(m+n)) = O(k \log n + \log m)$ time.

After answering the ray *shooting* query, we also *insert* the portion of the ray between the starting point and the first obstacle hit as a new obstacle into our data structure. The tree T (and the corresponding hierarchical cell decomposition) remains the same.

First, we update the set of domains $\mathcal{D}_u$, $u \in T$. As before, let v and w denote the children of a non-leaf node $u \in T$. In a top-down traversal of the tree T, detect all domains $D \in \mathcal{D}_u$ that intersect pq. By Lemma 9, pq intersects at most one domain $D \in \mathcal{D}_u$ for each $u \in T$. We distinguish four cases depending on the position of pq within $D \in \mathcal{D}_u$ (refer to Fig. 7). (1) If pq is disjoint from D, then no update is necessary, and there is no need to descend to the polygons of $\mathcal{D}_v$ and $\mathcal{D}_w$ contained in D. (2) If pq lies in the interior of D (that is, its endpoints lie on obstacles in the interior of D), then D is not updated, but we descend to the domains of $\mathcal{D}_v$ and $\mathcal{D}_w$ contained in D. (3) If pq has exactly one endpoint in the interior of D, say p, which is incident to an obstacle P in the interior of D, then we update D by appending the edges $pq \cap D$ and all edges of P to the boundary of D; and descend to its subdomains in $\mathcal{D}_v$ and $\mathcal{D}_w$. (4) Finally, if pq intersects the boundary of D twice, then we split D into two domains; and continue with its subdomains in $\mathcal{D}_v$ and $\mathcal{D}_w$.

In each domain $D \in \mathcal{D}_u$ along pq, we also update the geodesic hulls $\mathrm{gh}_D(S)$. This is done in a bottom up traversal of the tree T. When processing the geodesic hull with respect to $D \in \mathcal{D}_u$, $u \in T$, assume that the geodesic hulls $\mathrm{gh}_E(S)$ have been updated for all $E \in \mathcal{D}_v \cup \mathcal{D}_w$ at the children $v, w \in T$ of u. Note that $\mathrm{gh}_D(S)$ changes only if the domain D changes, that is, in cases (3) and (4) above. The ray pq crosses at most two edges of $\mathrm{gh}_D(S)$, these edges have to be removed. By Lemma 6, every edge of a cavity polygon or $\mathrm{gh}_D(S)$ that is not an edge of any $\mathrm{gh}_E(S)$, $E \in \mathcal{D}_v \cup \mathcal{D}_w$, is a common external tangent of two reflex arcs in two exterior polygons of some $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ for some $E_v \in \mathcal{D}_v$ and $E_w \in \mathcal{D}_w$

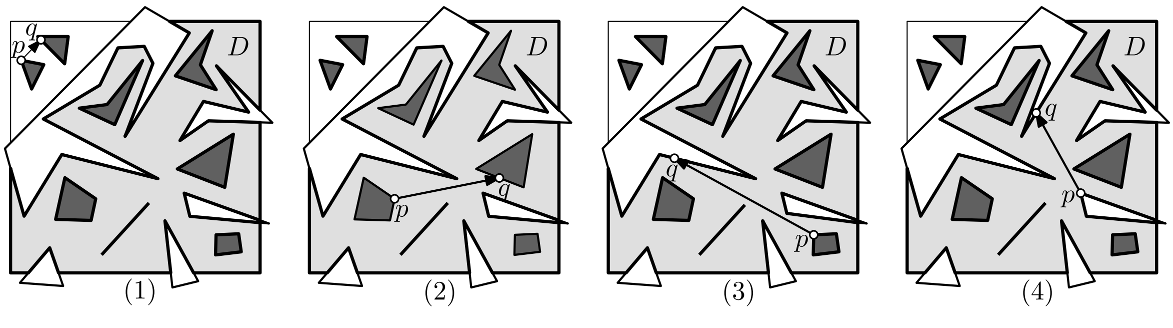

Figure 7: Cases for updating a domain $D \in \mathcal{D}_u$ depending on the position of the segment pq w.r.t. D.

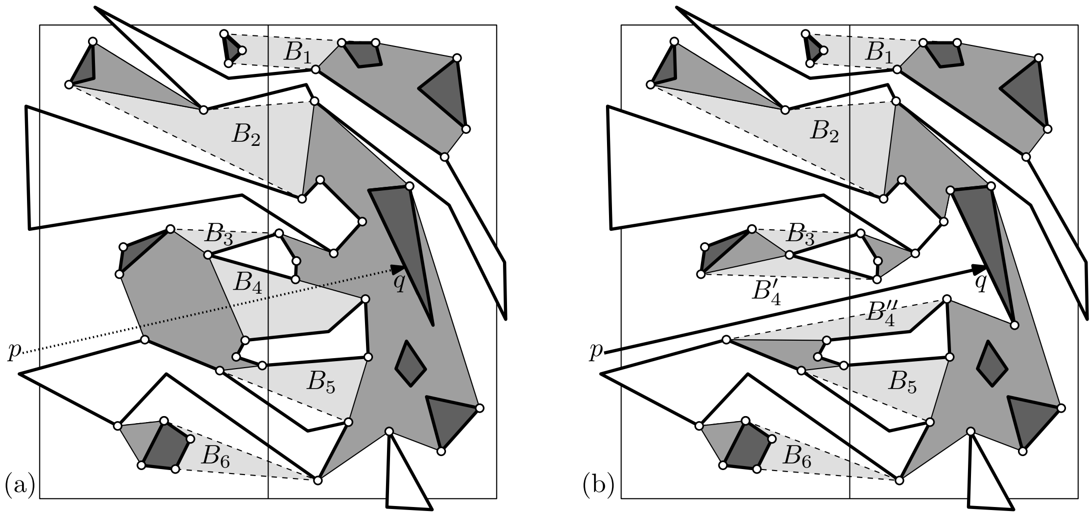

Figure 8: (a) The subdivision of $\mathrm{gh}_D(S)$ before inserting segment pq; (b) after inserting segment pq.

that are adjacent to line ℓ_u. It follows that an update is necessary only if the two exterior polygons are adjacent to line ℓ_u and the segment pq. There are two pairs of such extremal polygons, one on each side of pq. It is enough to compute the common external tangents between $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$ in these two pairs of exterior polygons. Hence, we can update the geodesic $\mathrm{gh}_D(S)$ and the cavity polygons of $\mathcal{Q}_u^*$ by deleting two edges crossed by pq and inserting two new common external tangents between two reflex arcs of $\mathrm{gh}_{E_v}(S)$ and $\mathrm{gh}_{E_w}(S)$. Since any two reflex arcs have a total of $O(n)$ vertices, we can update the geodesic hull in $\mathrm{gh}_D(S)$ in $O(\log n)$ time by Proposition 5.

Time complexity of m successive queries. For a line segment pq, denote by $\mathrm{tr}(pq)$ the number of edges of the tessellation G that intersect pq. We need to update all $\mathrm{tr}(pq)+1$ cavity polygons and geodesic hulls along pq (some of which may split into two parts). We saw that the ith ray shooting query takes $O(\mathrm{tr}(pq)\log n+\log i)$ time. The *insertion* of segment pq as a new obstacle takes $O(\mathrm{tr}(pq)\log n))$ time. The total runtime of processing m successive rays $p_1q_1, p_2q_2, \ldots, p_mq_m$, is

$$O\left(\left(\sum_{i=1}^{m}\mathrm{tr}(p_iq_i)\right)\log n + m\log m\right).$$

Proposition 10 *For m successive ray segment insertion queries, we have $\sum_{i=1}^{m}\mathrm{tr}(p_iq_i)=O((n+m)\log n)$.*

Proof. The tree T has $\log n$ levels. The geodesic hulls on level j, $0\le j<\log n$, of T are pairwise disjoint. Distinguish two types of intersections between a segment p_iq_i (between two points on the boundary of obstacles) and a geodesic hull $\mathrm{gh}_D(S)$: (1) segment p_iq_i partitions D into two domains $D=D^-\cup D^+$ and the point set $D\cap S$ is partitioned into two nonempty subsets $S\cap D=(S\cap D^-)\cup(S\cap D^+)$; (2) one of p_i and q_i lies in the interior of $\mathrm{gh}_D(S)$. Since a set of cardinality k can recursively be partitioned into nonempty subsets at most $k-1$ times, the geodesic hulls on each level are partitioned $O(n)$ times. The two endpoints of each segment p_iq_i may lie in the interior of two geodesic hulls on each level. So on each level of of T, there are $O(n)$ intersections of type (1) and $O(m)$ intersections of type (2). Summing the two types of intersections over $\log n$ levels, we obtain $\sum_{i=1}^{m}\mathrm{tr}(p_iq_i)=O((n+m)\log n)$. $\square$

The total runtime of m successive ray shooting-and-insertion queries is $O(n\log^2 n+m\log m)$. If the geometric partition tree T is arbitrary, then $\mathrm{tr}(pq)=\Theta(n)$ is possible and the ith ray shooting-and-insertion query takes $O(n\log^2 n+\log i)$ time. However, if we use a geometric partition tree T of low stabbing number, then every line intersect at most $O(n^{1/2+\delta})$ cells C_u, $u\in T$, for any fixed δ. By Lemma 9(1), a line segment in the free space intersects at most one domain $D\in\mathcal{D}_u$ and at most one geodesic hull $\mathrm{gh}_D(S)$ in cell C_u, and so $\mathrm{tr}(pq)=O(n^{1/2+\delta})$. Hence the ith ray shooting-and-insertion query takes $O(n^{1/2+\delta}\log n+\log i)\le O(n^{1/2+2\delta}+\log i)$ for any fixed $\delta>0$.

5 Conclusion

We proposed a data structure for disjoint polygonal obstacles in the plane with n vertices that supports m ray shooting-and-insertion queries in $O(n \log^2 n + m \log m)$ total time and $O((n+m)\log n)$ space. Our data structure applies, with minimal adjustments, to polygons with holes having a total of n vertices. With our data structure, we improve the expected runtime of Patersen and Yao's classical randomized auto-partition algorithm for n disjoint line segments in the plane to $O(n \lg^2 n)$. A deterministic auto-partition algorithm that fragment n disjoint line segments into a worst-case optimal number of pieces [29] can also be implemented in $O(n \log^2 n)$ time and $O(n \log n)$ space. Also the convex partition of the free space around n disjoint line segments in the plane (with $m = n$), where the segments are extended beyond their endpoints in a specified order, can now be computed in $O(n \log^2 n)$ time and $O(n \log n)$ space. It remains a challenge for future research to find an $O(n \log n)$ time algorithm.

It was essential for our techniques that the obstacles are *polygonal*. We believe that our techniques can be extended to handle curvilinear polygons, bounded by arcs of algebraic curves of bounded degree. Typically, n such arcs can be decomposed into a finite number of locally convex or concave arcs, and be approximated well enough by polygonal arcs with a total of $O(n)$ vertices. The extension of our data structure to disjoint obstacles of bounded description complexity is left for future work.

It is unlikely that our results generalizes to three or higher dimensions using linear "rays," since rays in $\mathbb{R}^2$ either partition the free space into two disjoint components or decrease the first Betty number (that is, the number of holes) in a single component. In $\mathbb{R}^3$, however, a linear ray never partitions the free space, it may decrease the 2nd Betty number of a component but increase the 1st Betty number. A natural generalization in $\mathbb{R}^3$ would ask for efficiently computing the convex partition of the free space around polyhedral obstacles with a total of n edges, where the partition steps are governed by queries. A convex partition of the free space may have $\Theta(n^2)$ vertices. The best we can hope for, therefore, is a data structure that supports computing the convex partition in output-sensitive near-linear total time.

References

[1] P. Agarwal and J. Erickson, Geometric range searching and its relatives, in: *Adv. in Discrete and Comput. Geometry* vol. 223 of *Contemp. Maths*, AMS, Providence, RI, 1999, pp. 1–56.

[2] P. K. Agarwal, Range searching, in: *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds., 2 ed. CRC Press LLC, Boca Raton, FL, 2004, ch. 36, pp. 809–838.

[3] P. K. Agarwal, J. Basch, L. J. Guibas, J. E. Hershberger, and L. Zhang, Deformable free space tilings for kinetic collision detection, *Int. J. Robotics Research* **21** (3) (2002), 179–197.

[4] P. K. Agarwal and M. Sharir, Applications of a new space-partitioning technique, *Discrete Comput. Geom.* **9** (1993), 11–38.

[5] O. Aichholzer, S. Bereg, A. Dumitrescu, A. García, C. Huemer, F. Hurtado, M. Kano, A. Márquez, D. Rappaport, S. Smorodinsky, D. Souvaine, J. Urrutia, and D. R. Wood, Compatible geometric matchings, *Comput. Geom.* (2009), in press.

[6] M. Al-Jubeh, M. Hoffmann, M. Ishaque, D. L. Souvaine, and C. D. Tóth, Convex partitions with 2-edge connected dual graphs, in: *Abstracts of the 18th Fall Workshop on Comput. Geom.*, 2008, pp. 62–63.

[7] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang, Kinetic collision detection between two simple polygons, *Comput. Geom. Theory Appl.* **27** (3) (2004), 211–235.

[8] B. Chazelle, On the convex layers of a planar set, *IEEE Trans. Inform. Theory* **IT-31** (4) (1985), 509–517.

[9] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, Ray shooting in polygons using geodesic triangulations, *Algorithmica* **12** (1994), 54–68.

[10] B. Chazelle, M. Sharir, and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, *Algorithmica* **8** (1992), 407–429.

[11] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.

[12] M. de Berg, personal communication, March, 2009.

[13] A. Ganguli, J. Cortes, and F. Bullo, Multirobot rendezvous with visibility sensors in nonconvex environments, *IEEE Transactions on Robotics* **25** (2) (2009), to appear.

[14] S. Ghosh, *Visibility Algorithms in the Plane.* Cambridge University Press, New York, 2007.

[15] M. T. Goodrich and R. Tamassia, Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations, *J. Algorithms* **23** (1997), 51–73.

[16] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica* **2** (1987), 209–233.

[17] J. Hershberger and S. Suri, Applications of a semi- dynamic convex hull algorithm, *BIT* **32** (1992), 249–267.

[18] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *J. Algorithms* **18** (3) (1995), 403–431.

[19] M. Hoffmann and C. D. Tóth, Segment endpoint visibility graphs are Hamiltonian, *Comput. Geom. Theory Appl.* **26** (1) (2003), 47–68.

[20] M. Ishaque and C. D. Tóth, Relative convex hulls in semi-dynamic subdivisions, in: *Proc. 16th ESA*, vol. 5193 of *LNCS*, 2008, Springer, pp. 780–792.

[21] D. G. Kirkpatrick and B. Speckmann, Kinetic main- tenance of context-sensitive hierarchical representati- ons for disjoint simple polygons, in: *Proc. 18th Symp. on Comput. Geom.*, 2002, ACM Press, pp. 179–188.

[22] J. Matousek, Spanning trees with low crossing number, *ITA* **25** (1991), 103–124.

[23] J. S. B. Mitchell, Geometric shortest paths and network optimization, in: *Handbook of Computational Geometry*, Elsevier, Amsterdam, 2000.

[24] M. H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Comput. Syst. Sci.* **23** (1981), 166–204.

[25] M. S. Paterson and F. F. Yao, Efficient binary space partitions for hidden-surface removal and solid modeling, *Discrete Comput. Geom.* **5** (1990), 485–503.

[26] M. Pellegrini, *Ray shooting and lines in space*, 2nd ed., CRC Press, Boca Raton, 2004, ch. 37, pp. 239–256.

[27] J. Sklansky, R. L. Chazin, and B. J. Hansen, Minimum perimeter polygons of digitized silhouettes, *IEEE Trans. Comput.* **C-21** (1972), 260–268.

[28] B. Speckmann, *Kinetic Data Structures for Collision Detection*, PhD thesis, University of British Columbia, Vancouver, 2001.

[29] C. D. Tóth, Binary plane partitions for disjoint line segments, in *Proc. 25th Sympos. on Comput. Geom.*, ACM Press, 2009, these proceedings.

[30] C. D. Tóth, Binary space partition for line segments with a limited number of directions, *SIAM J. Comput.* **32** (2) (2003), 307–325.

[31] G. T. Toussaint, Shortest path solves translation separability of polygons, Report SOCS-85.27, School Comput. Sci., McGill Univ., Montreal, QC, 1985.

[32] G. T. Toussaint, An optimal algorithm for computing the relative convex hull of a set of points in a polygon, in: *Signal Processing III: Theories and Applications*. 1986, pp. 853–856.

[33] E. Welzl, On spanning trees with low crossing numbers, in: *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, vol. 594 of *LNCS*, 1992, Springer, pp. 233–249.