Dynamic Geodesic Convex Hulls in Dynamic Simple Polygons*

Eunjin Oh[1] and Hee-Kap Ahn[2]

1 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
`jin9082@postech.ac.kr`
2 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
`heekap@postech.ac.kr`

Abstract

We consider the geodesic convex hulls of points in a simple polygonal region in the presence of non-crossing line segments (barriers) that subdivide the region into simply connected faces. We present an algorithm together with data structures for maintaining the geodesic convex hull of points in each face in a sublinear update time under the fully-dynamic setting where both input points and barriers change by insertions and deletions. The algorithm processes a mixed update sequence of insertions and deletions of points and barriers. Each update takes $O(n^{2/3}\log^2 n)$ time with high probability, where n is the total number of the points and barriers at the moment. Our data structures support basic queries on the geodesic convex hull, each of which takes $O(\text{polylog}\, n)$ time. In addition, we present an algorithm together with data structures for geodesic triangle counting queries under the fully-dynamic setting. With high probability, each update takes $O(n^{2/3}\log n)$ time, and each query takes $O(n^{2/3}\log n)$ time.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Dynamic geodesic convex hull, dynamic simple polygons

Digital Object Identifier 10.4230/LIPIcs.SoCG.2017.51

1 Introduction

A set X of points in a (weakly) simple polygon P is *geodesically convex* if the shortest path between any two points of X with respect to P is contained in X. The *geodesic convex hull* of a set of points contained in a simple polygon, which was introduced by Sklansky et al. [18], is defined as the intersection of all geodesic convex sets containing the set.

Geodesic convex hulls have been widely used for a variety of applications including collision detection [1], robotics [10], and motion planning [19]. These algorithms assume that the input points and the region where the points lie are static, that is, all input elements remain the same over the time. However, in many real-world geometric applications, particularly those that run in real-time, input data may change over time–input data elements may be inserted or deleted. Therefore it is required to handle these changes to maintain the geodesic convex hull for dynamically changing input data efficiently.

In this paper, we consider the problem of maintaining the geodesic convex hulls of dynamic points in a dynamic simple polygonal region. In the problem, points are inserted and deleted in a simple polygonal region as well as the region changes by insertions and deletions of

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

non-crossing line segments (barriers) under the restriction that the dynamic line segments always subdivide the region into simply connected faces.

We also study data structures that support a *geodesic triangle counting query* under the dynamic environment. The geodesic triangle defined by three points contained in a simple polygon P is the geodesic convex hull of them. Given a set S of input points in P, a geodesic triangle counting query asks for the number of input points contained in the interior of the geodesic triangle defined by three query points. No algorithm or data structure is known for the dynamic geodesic triangle counting problem in a dynamic region while an algorithm for the Euclidean version of the problem is known [14].

Related work for dynamic geodesic convex hulls. The *convex hull* of a set S of n points in the plane is defined to be the intersection of all convex sets containing S. The first nontrivial algorithm for maintaining the convex hull under point insertions and deletions in the plane was given by Overmars and Leeuwen [16]. Their data structure supports each update in $O(\log^2 n)$ worst-case time, where n is the number of points they have at the moment. The data structure can answer several basic queries including finding the extreme point in a direction and finding tangent points. Each query takes $O(\log n)$ worst-case time. Later, the update time was improved to $O(\log n)$ amortized time [3].

Surprisingly, little has been known for maintaining geodesic convex hulls of dynamic points in a dynamic region, except the one by Ishaque and Tóth [12]. They considered the problem in a *semi-dynamic* setting where point insertions and barrier deletions are not allowed (a barrier is an edge of a simple polygon in their work). With this restriction, they observe that the geodesic convex hull of points contained in a connected face of the barrier arrangement gradually decreases. Moreover, the total number of changes to their convex hull representations is $O(n+m)$, where n is the number of points in the initial state and m is the number of barriers in the final state. With this observations, they presented an algorithm to maintain the geodesic convex hulls in $O((n+m)\log^2(n+m)\log n)$ total time. Their data structure supports several basic queries in $O(\text{polylog}\{n,m\})$ time.

A natural question one may ask is whether the geodesic convex hulls can be maintained with a sublinear update time in a more general setting where both insertions and deletions of points and barriers are allowed.

Related work for geodesic triangle counting. The simplex counting problem is a fundamental query problem which has been studied extensively in the literature [4, 6, 14]. For a set of n static points in $\mathbb{R}^d$, Chan [4] gave a near-optimal algorithm which achieves $O(n^{1-1/d})$ query time with high probability after $O(n\log n)$-time preprocessing using linear space. The dynamic version of this problem where insertions and deletions of points are allowed is *decomposable* in the sense that a query over $D \cup D'$ can be answered in constant time from the answers from D and D' for any pair of disjoint data sets D and D' [14]. Thus, we can obtain a dynamic data structure from a static data structure of this problem using the framework of Bentley and Saxe [2], or Overmars and Leeuwen [17].

The geodesic triangle counting problem in a simple polygon is a generalization of the simplex counting problem. In the problem, we are given three query points in a simple polygon and want to count all input points contained in the interior of the geodesic triangle defined by the query points. We do not know any previous work achieving nontrivial results on this problem. We consider this problem in the fully-dynamic setting where both input points and barriers change by insertions and deletions. The problem is not decomposable because the simple polygon changes in the course of updates. Therefore, it is unclear how to

apply the framework of Bentley and Saxe [2], or Overmars and Leeuwen [17] even if we have a data structure for the static version of the problem.

Our results. We present an algorithm and data structures to maintain the geodesic convex hulls in the fully-dynamic setting where both input points and barriers change by insertions and deletions. Each update can be processed in $O(n^{2/3}\log^2 n)$ amortized time with high probability, where n is the total number of points and barriers at the moment. In addition, we show that any data structure for maintaining all edges of the geodesic convex hull requires $\Omega(n^{1/3})$ update time. By maintaining the geodesic convex hulls, we can answer various basic queries in $O(\text{polylog}\, n)$ time in the worst case: line stabbing, inclusion, and tangent queries.

In this algorithm, we use a subprocedure to answer a geodesic triangle counting query under insertions and deletions of points and barriers. Each update can be processed in $O(n^{2/3}\log n)$ amortized time with high probability, and each query can be answered in $O(n^{2/3}\log n)$ time with high probability, where n is the total number of points and barriers at the moment. We believe that this algorithm is of independent interest.

All these algorithms are randomized in terms of their running times because we use the partition tree given by Chan [4]. We can obtain algorithms with deterministic running times by using the partition tree of Chazelle et al. [8] instead of the one of Chan [4]. In this case, the update times increase slightly while the query times remain the same.

1.1 Outline

We first present a data structure for a geodesic triangle counting query for static points contained in a static simple polygon P. This data structure consists of three levels. The first level is the geodesic triangulation of P obtained from the algorithm by Chazelle et al. [7]. In the second level, each geodesic triangle of the geodesic triangulation is associated with a balanced binary search tree with respect to the x-coordinates of the input points contained in the geodesic triangle. In the third level, each node of the balanced binary search tree is associated with a data structure for an Euclidean triangle counting query.

Given three query points which define the geodesic triangle ⅄ contained in P, we first find the nodes of the balanced binary search trees whose associated point sets contain input points contained in ⅄. For each such node, we find an Euclidean triangle $\triangle$ such that $S' \cap \triangle$ coincides with $S' \cap \text{⅄}$ for a set S' of the input points associated with the node. Then we use a query algorithm of an Euclidean triangle counting query with $\triangle$. This procedure takes $O(n^{1/2}\log n)$ time with high probability, where n is the total number of points and barriers.

We use this data structure to answer a geodesic triangle counting query for dynamic points contained in a dynamic simple polygon P. The key idea is that we reconstruct the data structure periodically, instead of updating it for each change. To be specific, we reconstruct the data structure after $n^{1/3}$ updates are made, where n is the total number of the points and barriers at the moment.

Given the geodesic triangle ⅄ defined by three query points, ⅄ may not be the geodesic convex hull of its three corners with respect to the barriers we had when the data structure was constructed, because the barrier set has changed after the (re)construction of the data structure. To overcome this difficulty, we decompose ⅄ into smaller geodesic triangles with respect to the barriers we had at the last time the data structure was constructed such that for each smaller geodesic triangle $\widetilde{\text{⅄}}$, there are $O(n^{1/3}\log n)$ nodes of the balanced binary search trees whose associated point sets contain some input points contained in $\widetilde{\text{⅄}}$. Using these properties, we apply the query algorithm for the static data structure with each smaller

geodesic triangle in the decomposition of $\triangle$. By careful analysis, we show that the query time is $O(n^{2/3} \log n)$ with high probability.

Then we use this dynamic structure for geodesic triangle counting queries and show how to maintain the geodesic convex hulls in the fully-dynamic setting. We observed that the total number of distinct edges that appear on the geodesic convex hulls is less than the total changes to their convex hull representations. Based on this observation, we compute some edges that may appear on the geodesic convex hull in advance when we construct the data structure of a geodesic triangle counting query.

To handle each update, we replace part of the convex hull with a chain of edges precomputed and maintained in the data structure. To find such a chain, we apply a geodesic triangle counting query. Then we can achieve $O(n^{2/3} \log^2 n)$ update time with high probability.

Due to lack of space, some of the proofs and details are omitted.

2 Preliminaries

Arrangements of Barriers. Let R be a sufficiently large rectangle. We consider the arrangement of a set B of non-crossing line segments, called barriers, contained in R. The set B is initially empty and changes by insertions and deletions of barriers.

The *intersection graph* of B is defined as follows. The graph has $|B| + 1$ nodes, one for each barrier in B and an additional node for R, where $|X|$ denotes the number of elements in a set X. Two nodes of the graph are connected by an edge if and only if their corresponding barriers (or the boundary of R) are adjacent: one endpoint of a barrier lies on the other barrier (or the boundary of R). We restrict the intersection graph to be connected at any time, which gives a certain constraint on barriers that we consider. This restriction was also assumed in the papers [5, 9, 11, 12]. Moreover, we require a barrier to have a constant number of adjacent barriers, that is, each node in the intersection graph has a constant degree, which was also assumed in the papers [5, 9, 11].

Each connected region of $R \setminus \cup_{b \in B} b$ together with its boundary forms a weakly simple polygon under the restriction. A single point may appear on the boundary of a weakly simple polygon more than once, but we treat them as distinct points. We call each connected region of $R \setminus \cup_{b \in B} b$ together with its boundary a *face* of the arrangement of B.

In the following, we assume that the intersection graph of B is a tree. Thus the arrangement of B consists of exactly one face at all times. We use $R(B)$ to denote the unique face (weakly simple polygon) of the arrangement. A more general case that the intersection graph of B has cycles can also be handled analogously in the same time and space since there are a constant amount of changes to the arrangement for each update. See Figure 1(a). We can get rid of this assumption by modifying our algorithm slightly.

Update Sequences. Initially, both S and B are empty. We are given updates for points and barriers one by one. An update is either an insertion or a deletion of a point or a barrier. With these updates, S and B change accordingly. We are to process each update before we receive the next update. Let $U = \langle u_1, u_2, \ldots \rangle$ be a mixed sequence of updates. Let S_i and B_i denote S and B, respectively, after u_i is processed for an index i. Let CH_i denote the geodesic convex hull of S_i with respect to B_i. Let n_i be the total complexity of S_i and B_i.

For two sets X and Y, we use $X \oplus Y$ to denote the symmetric difference between X and Y, that is, $(X \setminus Y) \cup (Y \setminus X)$. By definition, it holds that $|S_i \oplus S_j| + |B_i \oplus B_j| \leq |i - j|$. We sometimes use an index i of U to denote the interval between the time when we receive u_i and the time when we receive u_{i+1}. Time i indicates an arbitrary time in this interval.

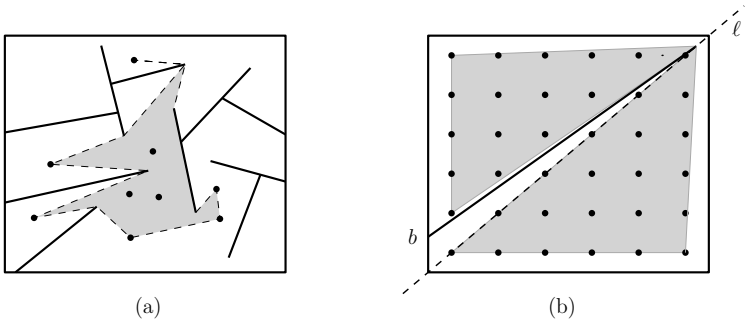


Figure 1 (a) The geodesic convex hull of points in a weakly simple polygon. (b) Every point on ℓ appears on the geodesic convex hull when barrier b is inserted.

Geodesic triangles and geodesic triangulations. For any two points x and y in $R(B)$, the *geodesic path* between x and y, denoted by $\pi_B(x, y)$ (or simply by $\pi(x, y)$), is the shortest path between x and y in $R(B)$. The boundary of the *geodesic triangle* defined by three points p_1, p_2 and p_3 in $R(B)$ consists of three geodesic paths $\pi_B(p_1, p_2)$, $\pi_B(p_2, p_3)$ and $\pi_B(p_3, p_1)$. The three vertices p_1, p_2 and p_3 are called the *corners* of the geodesic triangle. The interior of the geodesic triangle is bounded by three concave chains $\pi_B(p'_1, p'_2)$, $\pi_B(p'_2, p'_3)$, $\pi_B(p'_3, p'_1)$, where p'_i is the point such that $\pi_B(p_i, p'_i)$ is the maximal common path of $\pi_B(p_i, p_j)$ and $\pi_B(p_i, p_k)$ for three distinct indices i, j and k in $\{1, 2, 3\}$. We call the interior of a geodesic triangle the *deltoid* of it. We slightly abuse the term "deltoid" to denote a geodesic triangle excluding its boundary whose deltoid coincides with itself.

We sometimes mention a geodesic triangle (or deltoid) without specifying its corners. In this case, we mention it together with a barrier set B such that the boundary of the geodesic triangle consists of three geodesic paths with respect to B.

A *geodesic triangulation* of $R(B)$ is the decomposition of $R(B)$ into a number of interior-disjoint geodesic triangles with respect to B such that the union of the geodesic triangles coincides with $R(B)$. We say that a geodesic triangulation of $R(B)$ is *balanced* if any line segment that avoids the barriers in B intersects $O(\log |B|)$ geodesic triangles of the triangulation.

A lower bound for maintaining the geodesic convex hull. To maintain the geodesic convex hull at all times, we have to store the edges appearing on the boundary of the geodesic convex hull in the algorithm, which takes $\Omega(n^{4/3})$ time in total by the following lemma.

▶ **Lemma 1.** *For a mixed sequence of n updates of points and barriers, the number of distinct edges appearing on the boundary of the geodesic convex hull is $\Omega(n^{4/3})$.*

Proof. We prove this lemma using a lower bound example of a point-line incidence [15]. This lower bound example consists of a set $\mathcal{P}$ of N points and a set $\mathcal{L}$ of N lines such that there are $\Omega(N^{4/3})$ distinct point-line pairs (p, ℓ) with $p \in \ell$ for $p \in \mathcal{P}$ and $\ell \in \mathcal{L}$.

We construct a mixed sequence of n updates as follows. Let $N = n/3$. We first insert the points of $\mathcal{P}$ one by one. For a line $\ell \in \mathcal{L}$, there is a barrier with one endpoint on the boundary of R such that the insertion of the barrier makes all points of $\mathcal{P}$ lying on ℓ appear on the boundary of the geodesic convex hull. See Figure 1(b). We insert such a barrier, and then we remove it. We repeat this for every line in $\mathcal{L}$. Then we have $n/3$ point insertions, $n/3$ barrier insertions, and $n/3$ barrier deletions.

For the insertion of a barrier, the number of the new edges appearing on the geodesic convex hull is at least the number of the points of $\mathcal{P}$ lying on the line $\ell \in \mathcal{L}$ corresponding

to the barrier. Moreover, the new edges are contained in ℓ. Therefore, all such edges are distinct for every line in $\mathcal{L}$. This implies that the number of distinct edges of the geodesic convex hull is $\Omega(n^{4/3})$. ◀

Note that this lower bound example contains collinear points. We can perturb the points slightly in a certain way such that no three distinct points are collinear and the bound still holds. We omit details due to lack of space.

3 Dynamic Geodesic Triangle Range Queries

We are given three points c_1, c_2 and c_3 in $R(B_i)$ as a query for a geodesic triangle counting problem. We show how to compute the number of points of S_i lying in the deltoid of the geodesic triangle of c_1, c_2 and c_3 at time i.

3.1 Two Data Structures

We maintain two data structures: a geodesic-path data structure and an α-geodesic-triangulated subdivision. The first one is given by Goodrich and Tamassia [11]. With their structure, we can compute the geodesic path between any two query points in $R(B_i)$ represented as a balanced binary search tree in $O(\log^2 n_i)$ time.

The second one is our main data structure. At time i, an *α-geodesic-triangulated subdivision* is a balanced geodesic triangulation of $R(B_{i-\alpha})$ for some $0 \leq \alpha \leq i$ such that every geodesic triangle C in the triangulation is associated with a hierarchy of the partition trees given by Chan [4] on the point set $C \cap S_{i-\alpha}$, which will be described later.

The first level: balanced geodesic triangulation. We use the balanced geodesic triangulation $\mathcal{T}_i$ of $R(B_{i-\alpha})$ given by Chazelle at al. [7]. We call α the *inconsistency* of $\mathcal{T}_i$. The choice of α will be described in Section 3.2. Let $\widetilde{B}_i$ and $\widetilde{S}_i$ be $B_{i-\alpha}$ and $S_{i-\alpha}$, respectively. We call a deltoid in the geodesic triangulation a *cell* of $\mathcal{T}_i$. To make the description easier, we assume that no point of $\widetilde{S}_i$ lies on the boundary of any cell of $\mathcal{T}_i$. If it is not the case, we assume that a point of $\mathcal{T}_i$ lying on the common boundary of two cells of $\mathcal{T}_i$ belongs to exactly one of them to avoid overcounting.

The geodesic triangulation is for a *static* simple polygonal region. We do not make any change to this structure for updates until the inconsistency of the structure exceeds a certain level. Then we reconstruct it from scratch so that the structure has no inconsistency for the current set of barriers. Thus, $\widetilde{B}_i$ and $\widetilde{S}_i$ are B and S we had at the last time the data structures were constructed, respectively. Details will be described in Section 3.2.

▶ **Lemma 2.** *A deltoid with respect to B_i intersects $O((\alpha+1)\log|\widetilde{B}_i|)$ cells in the α-geodesic-triangulated subdivision. Moreover, they can be found in $O((\alpha+1)(\log^2|\widetilde{B}_i| + \log\alpha))$ time.*

The second and third levels: a hierarchy of partition trees. Imagine that we construct Chan's partition tree on $\widetilde{S}_i \cap C$ for every cell C of $\mathcal{T}_i$ that answers a Euclidean triangle counting query [4]. The partition tree requires $O(N)$ preprocessing time, $O(N)$ space, and $O(\sqrt{N})$ query time with high probability, where N is the number of input points.

▶ **Lemma 3.** *Let C be a cell of $\mathcal{T}_i$ and $\widetilde{\Delta}$ be a deltoid with respect to $\widetilde{B}_i$. We can compute the number of points in $(C \cap \widetilde{S}_i) \cap \widetilde{\Delta}$ in $O(n_C^{1/2})$ time with high probability, where $n_C = |C \cap \widetilde{S}_i|$.*

In our problem, a query is given as a deltoid with respect to B_i, not $\widetilde{B}_i$. Thus, we cannot apply the algorithm in Lemma 3 to the query deltoid directly. Instead, for a query deltoid $\triangle$, we construct a set $\widetilde{\mathcal{Q}}(C, \triangle)$ of pairwise disjoint deltoids with respect to $\widetilde{B}_i$ for each cell C of $\mathcal{T}_i$ such that the closures of the deltoids in $\widetilde{\mathcal{Q}}(C, \triangle)$ contain $C \cap \triangle$ in their union. We apply the algorithm in Lemma 3 to each deltoid in $\widetilde{\mathcal{Q}}(C, \triangle)$ and get the answer. The running time of this approach is $O(m_C n_C^{1/2})$, where $m_C = |\widetilde{\mathcal{Q}}(C, \triangle)|$ and $n_C = |C \cap \widetilde{S}_i|$.

To reduce the running time, instead of constructing Chan's partition tree on the set $C \cap \widetilde{S}_i$ for each cell C of $\mathcal{T}_i$, we construct a hierarchy of Chan's partition trees on $C \cap \widetilde{S}_i$ using a range tree. We construct a one-dimensional range tree (balanced binary search tree) on $C \cap \widetilde{S}_i$ with respect to the x-coordinates of the points. Each node v of the tree corresponds to a vertical slab $H(v)$. The root corresponds to the (degenerate) vertical slab $\mathbb{R}^2$. The left child and the right child of a node v correspond to the left vertical slab and the right vertical slab obtained from the partition of $H(v)$ with respect to the median x-coordinate of $\widetilde{S}_i$ contained in $C \cap H(v)$, respectively. For each node v of the range tree, we construct Chan's partition tree on the set $(C \cap H(v)) \cap \widetilde{S}_i$ (excluding the points of $\widetilde{S}_i$ lying on the right vertical side of $H(v)$) as the associated data structure of v.

In the query algorithm, we construct a set $\widetilde{\mathcal{Q}}(C, \triangle)$ of pairwise disjoint deltoids with respect to $\widetilde{B}_i$ satisfying Property ($\star$) such that the closures of the deltoids contains $C \cap \triangle$ in their union. This procedure is described in Section 3.3.1. Then we apply Lemma 4 as a subprocedure.

▶ **Property ($\star$).** *Any vertical line intersects $O(1)$ deltoids in $\widetilde{\mathcal{Q}}(C, \triangle)$ for every cell C in $\mathcal{T}_i$.*

▶ **Lemma 4.** *Given a deltoid $\triangle$ with respect to B_i, we can compute the number of points of $\widetilde{S}_i$ contained in $C \cap \triangle$ for a cell C of $\mathcal{T}_i$ in $O((m_C n_C)^{1/2} \log n_C)$ time with high probability, where $m_C = |\widetilde{\mathcal{Q}}(C, \triangle)|$ and $n_C = |C \cap \widetilde{S}_i|$.*

▶ **Lemma 5.** *We can construct the hierarchy of Chan's partition trees for every cell C of $\mathcal{T}_i$ in $O(|\widetilde{S}_i| \log(|\widetilde{B}_i| + |\widetilde{S}_i|))$ time in total with high probability.*

Maintaining the point-location data structure. We store one more piece of information in the second level of the α-geodesic-triangulated subdivision. For each cell C of $\mathcal{T}_i$, we maintain the dynamic point-location data structure of Chan and Nekrich [5] on the boundary of C and the barriers of $B_i \setminus \widetilde{B}_i$ intersecting C. Their data structure supports two types of queries: a point-location query and a vertical ray-shooting query. Each query takes $O(\log N (\log \log N)^2)$ time and each update takes $O(\log N \log \log N)$ time, where N is the complexity of the boundary of C and the barriers of $B_i \setminus \widetilde{B}_i$ intersecting C.

3.2 A Procedure for Updates

We do not make any change to the geodesic-triangulated subdivision for updates until the inconsistency α becomes larger than $n_i^{1/3}$. Then we reconstruct the subdivision from scratch so that the structure has no inconsistency, that is, we set $\widetilde{S}_i = S_i$ and $\widetilde{B}_i = B_i$ and construct $\mathcal{T}_i$ accordingly when the inconsistency becomes larger than $n_i^{1/3}$. We can reconstruct the geodesic-triangulated subdivision in $O(n_i^{2/3} \log n_i)$ amortized time.

For the other two data structures, we update them for each insertion and deletion of barriers: the point-location data structure for every cell C of $\mathcal{T}_i$ intersecting the barrier in $O(n_i^{1/3} \log^3 n_i (\log \log n_i))$ time, and the geodesic-path data structure in $O(\log^2 n_i)$ time.

▶ **Lemma 6.** *At any time i, the amortized time complexity for the reconstruction of the geodesic-triangulated subdivision is $O(n_i^{2/3} \log n_i)$ with high probability.*

3.3 A Procedure for Geodesic Triangle Counting Queries

Assume that we have an update u_i to process and we have the α-geodesic-triangulated subdivision $\mathcal{T}_i$ with $\alpha \leq n_i^{1/3}$. Note that $|\widetilde{B}_i| = O(n_i)$ and $|\widetilde{S}_i| = O(n_i)$. Given any three query points c_1, c_2 and c_3 in $R(B_i)$, we present an algorithm that returns the number of points of S_i contained in the deltoid $\triangle$ of the geodesic triangle defined by c_1, c_2, c_3 with respect to B_i in $O(n_i^{2/3} \log n_i)$ time. Recall that the geodesic triangulated subdivision is constructed on $R(\widetilde{B}_i)$ while $\triangle$ is a deltoid with respect to B_i. Thus we cannot apply the algorithm in Lemma 3 directly to $\triangle$. Instead, we decompose $\triangle$ into a number of deltoids with respect to $\widetilde{B}_i$ and apply the algorithm in Lemma 3 to each such deltoid.

The query algorithm consists of three steps. In the first step, we find all cells of $\mathcal{T}_i$ intersecting $\triangle$ in $O((\alpha+1)\log^2 n_i)$ time using Lemma 2. Let $\mathcal{C}(\triangle)$ be the set of such cells. In the second step described in Section 3.3.1, for each cell $C \in \mathcal{C}(\triangle)$, we construct a set $\widetilde{\mathcal{Q}}(C, \triangle)$ of pairwise disjoint deltoids satisfying Property $(\star)$ with respect to $\widetilde{B}_i$ whose closures contain $C \cap \triangle$ in their union. The total complexity of this set for every cell in $\mathcal{C}(\triangle)$ is $O(1+\alpha)$. Then, in the last step described in Section 3.3.2, we compute the number X_1 of the points of $\widetilde{S}_i$ contained in $\triangle$ using the set $\widetilde{\mathcal{Q}}(C, \triangle)$ for every cell $C \in \mathcal{C}(\triangle)$. In addition, we compute the numbers of points in $S_i \setminus \widetilde{S}_i$ and $\widetilde{S}_i \setminus S_i$ contained in $\triangle$, and denote them by X_2 and X_3, respectively. Then $X_1 + X_2 - X_3$ is the number of points of S_i contained in $\triangle$.

3.3.1 Constructing a Set of Deltoids with respect to $\widetilde{B}_i$

Let C be a cell in $\mathcal{C}(\triangle)$. In brief, we construct a set $\widetilde{\mathcal{Q}}(C, \triangle)$ of pairwise disjoint deltoids with respect to $\widetilde{B}_i$ as follows. We compute the vertical decomposition of C with respect to the barriers in $B_i \oplus \widetilde{B}_i$. Then we find all cells of the vertical decomposition intersecting $\triangle$. For each such cell, we compute the intersection of the cell with $\triangle$, which is the geodesic convex hull of at most six points with respect to $\widetilde{B}_i$. Then for each intersection, we obtain at most four deltoids from a geodesic triangulation of it. All such deltoids form $\widetilde{\mathcal{Q}}(C, \triangle)$.

Computing the vertical decomposition. Let $\mathcal{V}(C)$ be the set of the endpoints of the barriers of $B_i \oplus \widetilde{B}_i$ contained in the closure of C. We consider two vertical extensions from every endpoint in $\mathcal{V}(C)$ going to opposite directions until they hit a barrier in $B_i \oplus \widetilde{B}_i$ or the boundary of C. Note that no barrier in $\widetilde{B}_i$ intersects the interior of C. Thus, we can compute all extensions in $O(|\mathcal{V}(C)| \log n_i (\log\log n_i)^2)$ time in total using the point-location data structure associated with C that supports a vertical ray-shooting query.

The extensions together with the barriers in $B_i \setminus \widetilde{B}_i$ decompose C into $O(1+\alpha)$ cells. We call this decomposition the *vertical decomposition of* C. We dynamically maintain the arrangement of $B_i \oplus \widetilde{B}_i$ using the point-location data structure associated with C. Thus by computing all extensions, we can obtain the vertical decomposition of C.

Note that each cell (connected region) of the vertical decomposition contains at most four convex vertices on its boundary. Moreover, the closure of each cell is the geodesic convex hull of its four convex vertices with respect to both $\widetilde{B}_i$ and B_i.

Traversing the vertical decomposition. Then we find all cells of the vertical decomposition of C intersecting $\triangle$ by traversing the vertical decomposition of C. In the case that C is contained in $\triangle$, every cell in the vertical decomposition intersects $\triangle$. Thus we consider the case that the boundary of $\triangle$ intersects C.

We observe that we can compute the intersection of the boundary of C with the boundary of $\triangle$ while we compute the set $\mathcal{C}(\triangle)$. Then, starting from the intersection points, we traverse

the vertical decomposition of C along the boundary of $\triangle$. In this way, we can visit every cell of the vertical decomposition intersecting $\triangle$. Moreover, we visit only the cells intersecting $\triangle$.

Every cell of the decomposition intersecting $\triangle$ contains an endpoint of at most one barrier of $B_i \setminus \widetilde{B}_i$ on its boundary. Thus, during the traversal, we can find a neighboring cell of the vertical decomposition intersecting the boundary of $\triangle$ in $O(\log n_i)$ time. The running time of this procedure is bounded by the number of cells of the decomposition intersecting $\triangle$ times $O(\log n_i)$.

Computing the set of deltoids. For each cell of the vertical decomposition of C intersecting $\triangle$, consider the intersection of $\triangle$ with the cell. It is the geodesic convex hull of at most six points with respect to both $\widetilde{B}_i$ and B_i. By computing a geodesic triangulation of it with respect to $\widetilde{B}_i$, we can obtain at most four deltoids with respect to $\widetilde{B}_i$ from the intersection in $O(\log^2 n_i)$ time. Here, we do not explicitly compute the deltoids. Instead, we compute the corners of each deltoid using the geodesic-path data structure we maintain. Then we can compute the convex vertices of the deltoid of the geodesic triangle in $O(\log^2 n_i)$ time.

Let $\widetilde{\mathcal{Q}}(C, \triangle)$ be the set of all such deltoids obtained from all cells in the vertical decomposition of C intersecting $\triangle$. Note that the number of all cells of the decomposition of C intersecting $\triangle$ is asymptotically the same as the size of $\widetilde{\mathcal{Q}}(C, \triangle)$. The running time of this procedure is bounded by the size of $\widetilde{\mathcal{Q}}(C, \triangle)$ times $O(\log^2 n_i)$.

Analysis. For analysis, we prove the followings.

- The total number of deltoids in $\widetilde{\mathcal{Q}}(C, \triangle)$ for all cells $C \in \mathcal{C}(\triangle)$ is $O(\alpha + 1)$.
- The set $\widetilde{\mathcal{Q}}(C, \triangle)$ satisfies Property $(\star)$.

The first claim implies that the running time for computing $\widetilde{\mathcal{Q}}(C, \triangle)$ for all cells $C \in \mathcal{C}(\triangle)$ is $O((\alpha + 1) \log n_i (\log \log n_i)^2)$, which is dominated by the time for constructing the vertical decomposition for every cell in $\mathcal{C}(\triangle)$. The first and second claims give properties for $\widetilde{\mathcal{Q}}(C, \triangle)$ we want to achieve.

3.3.2 Computing the Number of Points in a Deltoid

We compute the number of points in $(C \cap \widetilde{\triangle}) \cap \widetilde{S}_i$ for every cell C in $\mathcal{C}(\triangle)$ and every deltoid $\widetilde{\triangle}$ of $\widetilde{\mathcal{Q}}(C, \triangle)$. Due to properties of $\widetilde{\mathcal{Q}}(C, \triangle)$, we can compute it by applying Lemma 4 in $O(n_i^{2/3} \log n_i)$ time. Then we handle the points in $S_i \oplus \widetilde{S}_i$ by deciding whether each of them is contained in $\triangle$ in $O(\log^2 n_i)$ time. We omit details due to lack of space.

▶ **Theorem 7.** *A geodesic triangle counting query can be answered in $O(n^{2/3} \log n)$ time with high probability under insertions and deletions of points and barriers, where n is the total number of the points and barriers at the moment. We process each update in $O(n^{2/3} \log n)$ amortized time with high probability using $O(n \log n)$ space.*

4 Maintaining the Geodesic Convex Hull

In this section, we present an algorithm together with three data structures including the data structures described in Section 3 to maintain the geodesic convex hull under insertions and deletions of points and barriers.

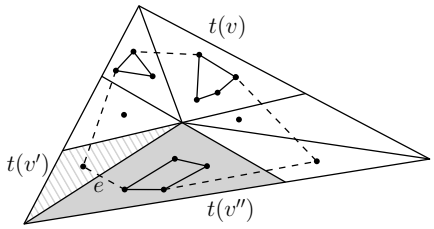


Figure 2 The nodes v' and v'' are children of the node v. For v, we compute the bridges (dashed segments). The edge e is the bridge of the convex hull of $P(v')$ and the convex hull of $P(v'')$.

4.1 A Triangle-Range Hull Tree

In addition to the data structures for a geodesic triangle query, we maintain a data structure, which we call a *triangle-range hull tree*. The triangle-range hull tree is constructed on the set $P_i = (S_i \cap \widetilde{S}_i) \cup V(B_i \cap \widetilde{B}_i)$, where $V(B)$ denotes the vertices of $R(B)$ for a set B of barriers. Note that $|P_i| = O(n_i)$. Given a set of points, this data structure allows us to compute the Euclidean convex hull of the points of P_i lying inside a query Euclidean triangle. In this section, we present an algorithm to construct the triangle-range hull tree and an algorithm to compute the Euclidean convex hull of the points contained in a query Euclidean triangle.

A partition tree of Chan. The triangle-range hull tree is a partition tree constructed on P_i containing additional information. There are several variants of a partition tree with different partitioning schemes. Among them, we use the partition tree given by Chan [4]. This is because in their scheme, the triangle $t(v)$ corresponding to a node v is subdivided into a constant number of *interior-disjoint* triangles each of which corresponds to a child of v. Each node v is associated with a point set $P(v) = t(v) \cap P_i$. Moreover, for any Euclidean triangle $\triangle$, the number of nodes v in T such that $t(v)$ intersects the boundary of $\triangle$ is $O(\sqrt{|P_i|})$.

Construction of the triangle-range hull tree. In our problem, we use the partition tree constructed on P_i to compute the Euclidean convex hull of points contained in a query Euclidean triangle. Recall that P_i consists of points from $S_i \cap \widetilde{S}_i$ and points from $V(B_i \cap \widetilde{B}_i)$. For a vertex p of the Euclidean convex hull of $P' \subseteq P_i$, we call the vertex of the Euclidean convex hull that comes first from p in clockwise order along the convex hull among all vertices from $S_i \cap \widetilde{S}_i$ (or, $V(B_i) \cap V(\widetilde{B}_i)$) the *$S$-neighbor* (or, *$B$-neighbor*) of p.

For every node v of the partition tree, we compute a part of the convex hull of $P(v)$ such that the partition tree supports the following operations for the convex hull of $P(v)$:

- **(O1)** Given two edges e_1 and e_2 of the convex hull, we can compute the number of edges of the convex hull lying from e_1 to e_2 in clockwise order in $O(\log n_i)$ time.
- **(O2)** For an integer j and an edge e of the convex hull, we can access the jth edge of the convex hull from e in clockwise order in $O(\log n_i)$ time.
- **(O3)** Given a vertex p of the convex hull, we can find the S-neighbor and B-neighbor of p in clockwise order along the convex hull in $O(\log n_i)$ time.

Let v be a node of T. Assume that for every descendant v' of v, we have already computed a part of the convex hull of $P(v')$ such that the partition tree supports all three operations. We show how to compute a part of the convex hull of $P(v)$ as follows. We compute a constant number of edges of the convex hull of $P(v)$ that do not appear on the convex hull of $P(v')$ for any child v' of v, which we call *bridges* for v. For illustration, see Figure 2. By property of the partition tree, $P(v)$ is the union of $P(v')$ for all children v' of v, and $t(v)$ contains

the union of the triangles $t(v')$ for all children v' of v. A bridge for v is an outer common tangent of two convex hulls of $P(v_1)$ and of $P(v_2)$ for two children v_1 and v_2 of v.

Kirkpatrick and Snoeyink [13] presented an algorithm to compute the outer common tangents of two given convex polygons with k vertices in $O(\log k)$ time once the vertices of each convex polygon are stored in an array. In our case, we need $O(\log k)$ time for accessing the jth edge of the convex hull of $P(v_1)$ (or $P(v_2)$) from a given edge for an integer j. Since a node of T has a constant number of children, we can compute all bridges for v in $O(\log^2 n_i)$ time. We maintain the bridges for v in clockwise order along the convex hull of $P(v)$. In addition, we compute information for the bridges to support operations O1, O2 and O3 for v. (We omit details due to lack of space.)

▶ **Lemma 8.** *The triangle-range hull tree supports operations O1, O2 and O3 for every node.*

To construct the triangle-range hull tree, we spend $O(\log^2 n_i)$ time for each node v, thus the running time for the construction is $O(n_i \log^2 n_i)$. Moreover, the number of bridges we compute additionally is asymptotically bounded by the number of edges of the partition tree. Thus, the size of the partition tree remains the same and we have the following lemma.

▶ **Lemma 9.** *The triangle-range hull tree can be constructed on P_i in $O(n_i \log^2 n_i)$ time with high probability. The size of the triangle-range hull tree is $O(n_i)$.*

Computation of the convex hull of points in a query Euclidean triangle. Let $\triangle$ be a Euclidean triangle. We compute the Euclidean convex hull of $P_i \cap \triangle$. To be specific, we compute a tree of size $O(\sqrt{n_i})$ supporting all three operations for the convex hull of $P_i \cap \triangle$. Each node of the tree is associated with a sequence of edges of the convex hull of $P_i \cap \triangle$.

The algorithm is similar to the one for triangle range searching. We start from the root of the triangle-range hull tree T. Let v be a node we just reached. Then we compute the convex hull of $P(u) \cap \triangle$ recursively for every child u of v. There are three possible cases: (1) $t(u)$ intersects the boundary of $\triangle$, (2) $t(u)$ is contained in the interior of $\triangle$, or (3) $t(u)$ does not intersect $\triangle$. For the first case, we search further the subtrees rooted at u recursively. For the second case, we already have the convex hull of $P(u) \cap \triangle$. This is because the subtree of T rooted at u supports the three operations for $P(u) = P(u) \cap \triangle$. For the third case, we do not search further the subtree rooted at u.

After computing the convex hull of $P(u) \cap \triangle$ for every child u of v, we compute the edges of the convex hull of $P(v) \cap \triangle$ which do not appear on the boundary of the convex hull of $P(u) \cap \triangle$ for any child u of v, which are bridges for v on the convex hull of $P(v) \cap \triangle$. Note that such a bridge is an outer common tangent of the convex hull of $P(u_1) \cap \triangle$ and the convex hull of $P(u_2) \cap \triangle$ for two children u_1 and u_2 of v. Since v has a constant number of children, there are a constant number of bridges for v. We compute them in $O(\log^2 n)$ time using the algorithm by Kirkpatrick and Snoeyink [13], and sort them in clockwise order along the convex hull of $P(v) \cap \triangle$. We also compute additional information to support operations O1, O2 and O3 for v. Finally, we reach leaf nodes v of T. Since $P(v)$ has a constant size, we compute the convex hull of all points of $P(v)$ lying inside $\triangle$ explicitly in constant time.

As a result of handling the query, we return the nodes of T we visited and the sequence of bridges for each such node. That is, the output of the query algorithm is a tree consisting of the nodes of T we visited each of which stores the sequence of bridges. Since we visited $O(\sqrt{n_i})$ nodes, the size of the output is $O(\sqrt{n_i})$. One difference of the partition tree and the output tree is that a leaf node of the output tree does not necessarily correspond to a constant number of points. It happens when $t(u)$ is contained in $\triangle$ for some node u of T. In

this case, the leaf node of the output tree points to its corresponding node of T. Thus we can apply all three operations on the leaf node of the output tree.

With a simple analysis, we can show that the running time of this query algorithm is $O(N \log^2 n_i)$, where N is the number of nodes v in T such that $t(v)$ intersects the boundary of $\triangle$. Therefore, the running time for our query algorithm is $O(\sqrt{n_i} \log^2 n_i)$.

▶ **Lemma 10.** *We can compute a tree of size $O(\sqrt{n_i})$ in $O(\sqrt{n_i} \log^2 n_i)$ time with high probability that supports the three operations for the convex hull of points of P contained in a query Euclidean triangle.*

Whenever we reconstruct the data structure for a geodesic triangle counting query, we also reconstruct the triangle-range hull tree. Additionally, we handle the deletion of a point p from $S \cup V(B)$ by removing it from the triangle-range hull tree in $O(\log^3 n_i)$ time.

4.2 Representation of the Geodesic Convex Hull

Ishaque and Tóth [12] showed that n updates may induce $\Omega(n^2)$ combinatorial changes in the geodesic convex hull. However, we observe that the total number of distinct edges of the geodesic convex hull under n updates is less than n^2. Based on this observation, we compute a number of chains in advance, which are geodesic paths stored in the geodesic-path data structure and the boundaries of convex hulls stored in the triangle-range hull tree. Then at any time i, we represent the geodesic convex hull CH_i of S_i with respect to B_i as a sequence consisting of subchains along the boundary of CH_i.

We maintain this sequence using a concatenable queue implemented by AVL-trees, and call it the *representation tree*. A concatenable queue is used also in [16] to represent the Euclidean convex hull. This data structure allows us to split a sequence, merge two sequences, insert or delete an element in $O(\log n_i)$ time. Each element in the representation tree corresponds to a subchain of the boundary of CH_i of one of the three types:

- **(T1)** A single edge of CH_i connecting two points in S_i
- **(T2)** A geodesic path with respect to B_i connecting two points in S_i
- **(T3)** A subchain of the convex hull of $P \cap \triangle$ for some Euclidean triangle $\triangle$

For a T1 element e, we simply store its corresponding edge to e. For an element of other types, instead of storing the subchain directly, we store information that supports the three operations for the element. Moreover, we store additional information to each node v (element) to support operations O1, O2 and O3 for CH_i.

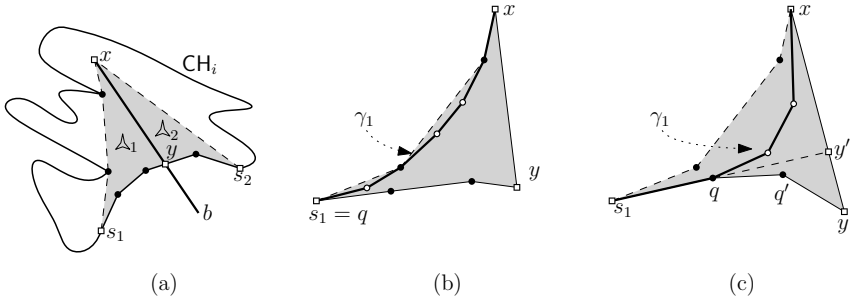
▶ **Lemma 11.** *The representation tree supports operations O1, O2 and O3 for CH_i.*

4.3 Procedures for Various Types of Queries

Lemma 11 allows us to answer various basic queries. Here, we show how to answer queries of three types, which were considered by Ishaque and Tóth [12]: a *line stabbing query*, an *inclusion query* and a *tangent query*.

For a line stabbing query with a line segment ℓ contained in $R(B_i)$, we want to find the intersection of ℓ with CH_i. For an inclusion query with a point p in $R(B_i)$, we want to determine whether or not p is contained in CH_i. For a tangent query with a point p lying outside of CH_i, we want to find the vertices v of CH_i where p is tangent to CH_i. Each query can be answered in polylogarithmic time.

▶ **Lemma 12.** *We can answer a line stabbing and inclusion query in $O(\log^2 n_i)$ time in the worst case. We can answer a tangent query in $O(\log^3 n_i)$ time in the worst case.*



(a) (b) (c)

Figure 3 (a) When b is inserted, we replace $\pi_{B_{i-1}}(s_1, s_2)$ with two convex chains in $\triangle_1$ and $\triangle_2$. (b), (c) We compute q such that the maximal common path of $\pi(s_1, y)$ and γ_1 is $\pi(s_1, q)$.

4.4 Procedures for Updates

We update the triangle-range hull tree when a point or a barrier is deleted. But we do not update it for insertions of points or barriers. Instead, we reconstruct it whenever we reconstruct the data structure for a geodesic triangle counting query. We also compute the geodesic convex hull of S with respect to B when we reconstruct them. After the reconstruction, the representation tree consists of T1 elements only.

In this paper, we present algorithms for processing an insertion of a barrier b. We omit details of the procedures for the other cases. At time i, we have the geodesic convex hull CH_{i-1}. We first check whether b intersects the interior of CH_{i-1} by applying the stabbing query with the line segment b. If b does not intersect the interior of CH_{i-1}, it holds that $\mathsf{CH}_i = \mathsf{CH}_{i-1}$. Otherwise, the intersection of b with the interior of CH_{i-1} consists of at most two connected components. (This happens when ℓ contains an edge of CH_{i-1}.) If it consists of two connected components, we consider them as two distinct barriers. So, in the following, we assume that the intersection of b with the interior of CH_{i-1} is connected.

Now we consider the intersection of b with the boundary of CH_{i-1}. The intersection consists of at most two points. We consider the case that the intersection is a single point y. The other case can be handled analogously.

Let s_1 and s_2 be the S-neighbors of y along the boundary of CH_{i-1} in clockwise and counterclockwise orders, and let x be the endpoint of b lying inside CH_{i-1}. See Figure 3(a). We can compute s_1 and s_2 in $O(\log n_i)$ time using operation O3. As the barrier b is inserted, some points of S_i lying in the interior of CH_{i-1} appear on the boundary of CH_i. Such points lie in the deltoid $\triangle$ of the geodesic triangle with three corners s_1, s_2 and x with respect to B_i. Moreover, we have the following observation.

▶ **Observation 13.** *CH_i is the geodesic convex hull of $(\mathsf{CH}_{i-1} \setminus \triangle) \cup (S_i \cap \triangle)$ w.r.t. B_i.*

Let $\triangle_1$ and $\triangle_2$ be the two deltoids such that their union including their common boundary xy is $\triangle$. Without loss of generality, we assume that $s_1 \in \triangle_1$ and $s_2 \in \triangle_2$. We use γ_t to denote the part of the boundary of the geodesic convex hull of $(S_i \cap \triangle_t) \cup \{s_t, x\}$ with respect to B_i that connects s_t and x (and is not $\pi(s_t, x)$) for $t = 1, 2$. See Figure 3(b). To obtain CH_i, we replace $\pi(s_1, s_2)$ with γ_1 and γ_2 in the representation tree of CH_{i-1}. We show how to compute γ_1 only. The polygonal chain γ_2 can be computed analogously. Then we show how to replace $\pi(s_1, s_2)$ with them in the representation tree.

The procedure for computing γ_1 consists of three steps. First, we find a smaller deltoid $\triangle' \subseteq \triangle_1$ with properties similar to ones of $\triangle_1$. Second, we find an Euclidean triangle $\triangle$ such that the part of the boundary of the Euclidean convex hull of $(S_i \cup V(B_i)) \cap \triangle$ from x

to q in clockwise order is exactly $\gamma_1 \setminus \pi(s_1, q)$, where q is the corner of $\triangle$ that does not lie on xy. Third, we compute the convex hull of $(S_i \cup V(B_i)) \cap \triangle$ using the triangle-range hull tree and update CH_i using this information.

Finding a smaller deltoid. Here, instead of considering $\mathbin{\text{⏢}}_1$, we choose a deltoid $\mathbin{\text{⏢}}' \subseteq \mathbin{\text{⏢}}_1$ satisfying the following properties:

- The boundary of $\mathbin{\text{⏢}}'$ consists of one (maximal) concave chain and two line segments.
- The boundary of the geodesic convex hull of $S_i \cap \mathbin{\text{⏢}}'$ excluding $\pi_{B_i}(x, q)$ is exactly $\gamma_1 \setminus \pi_{B_i}(s_1, q)$, where q is the corner of $\mathbin{\text{⏢}}'$ that does not lie on xy.

We observe that $\pi(s_1, y) \cap \gamma_1$ is connected. We find the point $q \in \pi(s_1, y)$ closest to y such that $\pi(s_1, q) \subseteq \gamma_1$. It is possible that $q = s_1$. See Figure 3(b) and (c). We can compute q in $O(n_i^{2/3} \log^2 n_i)$ time by applying binary search on $\pi(s_1, y)$ with a geodesic triangle counting query described in Section 3. We extend the edge of $\pi(s_1, q)$ incident to q until it hits xy. Let y' be the intersection of xy with the extension. We let $\mathbin{\text{⏢}}'$ be the geodesic triangle with corners q, y' and x. Then this geodesic triangle satisfies the properties we want to achieve.

Finding an Euclidean triangle $\triangle$. We choose the Euclidean triangle $\triangle$ whose three corners are the three corners of $\mathbin{\text{⏢}}'$. Since $\mathbin{\text{⏢}}'$ is a deltoid with respect to B_i, the line segment xq connecting x and q appears on the boundary of the Euclidean convex hull CH of $(S_i \cup V(B_i)) \cap \triangle$. Moreover, any point of $S_i \cup V(B_i)$ contained in the interior of the region bounded by $\pi(x, q)$ and xq does not appear on the boundary of CH. Thus the following holds.

▶ **Lemma 14.** *The polygonal chain $\gamma_1 \setminus \pi(s_1, q)$ coincides with the part of the boundary of the Euclidean convex hull of $(S_i \cup V(B_i)) \cap \triangle$ excluding xq.*

Computing the convex hull of $(S_i \cup V(B_i)) \cap \triangle$. Let $P = (\widetilde{S}_i \cap S_i) \cup V(\widetilde{B}_i \cap B_i)$. We first compute the convex hull of $P \cap \triangle$ using the triangle-range hull tree. Let CH be the convex hull. Then we consider each point p in $S_i \setminus \widetilde{S}_i$ one by one, and update CH to be the convex hull of CH and p. Each update takes $O(\log^2 n_i)$ time because we have to spend $O(\log n_i)$ time to access the jth vertex of CH for some index j. Finally, we compute the Euclidean convex hull of $(S_i \cup V(B_i)) \cap \triangle$.

▶ **Lemma 15.** *We can compute the Euclidean convex hull of $(S_i \cup V(B_i)) \cap \triangle$ in $O({n_i}^{1/2} \log^2 n_i)$ time with high probability.*

Now we have γ_1 and γ_2 consisting of $O({n_i}^{1/3})$ subchains belonging to T1, T2, or T3. We insert them to the representation tree in $O({n_i}^{1/3} \log n_i)$ time. Then we remove $\pi(s_1, s_2)$ from the representation tree. This takes $O(\log n_i)$ time since the representation tree is a concatenable queue supporting split operation.

Therefore, the running time for handling the insertion of a barrier is dominated by the running time for the second step, which takes $O(n_i^{2/3} \log^2 n_i)$ time.

▶ **Lemma 16.** *The geodesic convex hull of S_i with respect to $B_{i-1} \cup \{b\}$ for some barrier b can be computed in $O(n_i^{2/3} \log^2 n_i)$ time with high probability once we have CH_{i-1}.*

▶ **Theorem 17.** *We can update the geodesic convex hull in $O(n^{2/3} \log^2 n)$ amortized time with high probability under insertions and deletions of points and barriers, where n is the total number of the points and barriers at the moment. A line stabbing query, inclusion query, and tangent query can be answered in polylogarithmic time in the worst case.*

References

1. Julien Basch, Jeff Erickson, Leonidas J. Guibas, John Hershberger, and Li Zhang. Kinetic collision detection between two simple polygons. *Computational Geometry*, 27(3):211–235, 2004.
2. Jon Louis Bentley and James B. Saxe. Decomposable searching problems 1: Static-to-dynamic transformations. *Journal of Algorithms*, 1(4):297–396, 1980.
3. Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pages 617–626, 2002.
4. Timothy M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.
5. Timothy M. Chan and Yakov Nekrich. Towards an optimal method for dynamic planar point location. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 390–409, 2015.
6. Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, 2(4):637–666, 1989.
7. Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.
8. Bernard Chazelle, Micha Sharir, and Emo Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8(1):407–429, 1992.
9. Yi-Jen Chiang, Franco P. Preparata, and Roberto Tamassia. A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps. *SIAM Journal on Computing*, 25(1):207–233, 1996.
10. Anurag Ganguli, Jorge Cortés, and Francesco Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *IEEE Transactions on Robotics*, 25(2):340–352, 2009.
11. Michael T. Goodrich and Roberto Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *Journal of Algorithms*, 23(1):51–73, 1997.
12. Mashhood Ishaque and Csaba D. Tóth. Relative convex hulls in semi-dynamic arrangements. *Algorithmica*, 68(2):448–482, 2014.
13. David Kirkpatrick and Jack Snoeyink. Computing common tangents without a separating line. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS 1995)*, pages 183–193, 1995.
14. Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992.
15. Jiří Matoušek. *Lectures on discrete geometry*. Springer Science & Business Media, 2013.
16. Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Science*, 23(2):166–204, 1981.
17. Mark H. Overmars and Jan van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problem. *Information Processing Letters*, 12(4):168–173, 1981.
18. Jack Sklansky, Robert L. Chazin, and Bruce J. Hansen. Minimum-perimeter polygons of digitized silhouettes. *IEEE Transactions on Computers*, C-21(3):260–268, 1972.
19. Ileana Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete and Computational Geometry*, 34:587–635, 2005.