

On Approximate Geodesic-Distance Queries amid Deforming Point Clouds^{*}

Pankaj K. Agarwal, Alon Efrat, R. Sharathkumar, and Hai Yu

Abstract. We propose data structures for answering a geodesic-distance query between two query points in a two-dimensional or three-dimensional dynamic environment, in which obstacles are deforming continuously. Each obstacle in the environment is modeled as the convex hull of a continuously deforming point cloud. The key to our approach is to avoid maintaining the convex hull of each point cloud explicitly but still able to retain sufficient geometric information to estimate geodesic distances in the free space.

1 Introduction

The geodesic-distance query problem in an obstacle-present environment arises in motion planning, assembly planning, gaming industry, computational geometry, and related fields. The goal is to construct a suitable data structure based on the geometry of the environment, so that the length of a shortest obstacle-avoiding path between two query points can be reported quickly. This was the focal of motion planning.

Pankaj K. Agarwal and R. Sharathkumar
Department of Computer Science, Duke University
e-mail: {pankaj, sharath}@cs.duke.edu

Alon Efrat
Department of Computer Science, The University of Arizona
e-mail: alon@cs.arizona.edu

Hai Yu
Google Inc. New York, NY
e-mail: fishhai@google.com

^{*} Work on this paper is supported by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and W911NF-07-1-0376, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation. Part of the work was done while the last author was at Duke University.

However, most of the existing work has focused on answering geodesic distance queries in a static environment or when each obstacle moves as a rigid body. There are several reasons to study geodesic-distance query problems when the queries are time dependent and obstacles are deforming continuously:

- With the availability of sensing and tracking technology, it is possible to monitor many geo-temporal phenomena in real time. Wild fires, areas contaminated by hazardous gas, regions under surveillance of enemy forces (in military applications), and regions with bad atmospheric conditions (in flights scheduling applications) can all be modeled as obstacles deforming in time, where one wishes to find a short path avoiding all the obstacles within some time frame.
- In an environment with a large numbers of moving obstacles one might wish to hierarchically cluster the obstacles and represent each cluster by its convex hull. Answering motion-planning queries can be done by either avoiding all obstacles within a cluster, if possible, or solving the problem with the convex hulls of the children sub-clusters. Obstacles might change their trajectory, start or stop being vertices of the convex hulls, and move from cluster to a sibling cluster, according to their location.
- The *asteroid avoidance problem* is to plan the path of a robot from source to destination while avoiding moving obstacles and not exceeding a given velocity. This important problem is known to be PSPACE-hard [17]. A natural heuristic to the problem is to dividing the free space and time domain into fine enough cells, so that within each region obstacles can be considered static with respect to the velocity of the robot, and the (portion of the) shortest path is constrained to the cell. One could use our algorithm as a tool to answer many geodesic distance queries in different time intervals or to recompute shortest paths after refining the time intervals.

Motivated by these applications, we focus on a model in which each obstacle is represented as the convex hull of a dynamic point cloud, e.g., it may correspond to a squad of enemy troops in motion, a scatter of spreading wild fire, or a cluster of asteroids. In the following, we introduce the model of motion to be followed throughout the paper and define our problem formally.

Model of motion. We use the *kinetic data structure* (KDS for short) framework proposed by Basch *et al.* [6] to handle dynamics of the environment. Let τ denote the time parameter. A moving point $p(\tau)$ in \mathbb{R}^d , for $d = 2, 3$, is a function $p : \mathbb{R} \rightarrow \mathbb{R}^d$. We call $p(\tau)$ *algebraic* if each individual coordinate of $p(\tau)$ (a real-valued function) is an algebraic function of τ . We use point clouds to model a deforming convex polytope $P(\tau)$ in \mathbb{R}^d . In this *point-cloud model*, let $S(\tau) = \{p_1(\tau), \dots, p_n(\tau)\}$ be a finite set of moving points in \mathbb{R}^d , and a deforming convex polytope $P(\tau)$ is defined to be $\text{conv}S(\tau)$ — the convex hull of $S(\tau)$. We emphasize that points in $S(\tau)$ are allowed to change their trajectories at any moment if needed.

In the KDS framework, we maintain a data structure on the fly, accompanied by a set of geometric predicates, called *certificates*, to serve as a proof of correctness for the maintained data structure. Since the current motion of objects are known, the KDS is able to predict the time (called *failure time*) at which each certificate

becomes invalid. All the failure times are scheduled in a global queue called *event queue*. The KDS does nothing until the time reaches the first failure time in the event queue. At that moment, an *event* occurs, and the KDS processes this event by updating the data structure to restore its correctness, as well as updating the certificates and the event queue accordingly. The KDS then moves on towards the next event.

As mentioned above, a change of trajectory or speed for an object is allowed, and it is assumed that the KDS is notified about the change when it occurs. A KDS is called *local* if each moving object is involved in a small number of certificates. A local KDS is able to quickly respond to the motion change of an object, by recomputing the failure times of all the certificates involving the object. All our KDS's are local and therefore accommodate motion changes efficiently.

Problem statement. Let \mathcal{F} be a path-connected closed subset in \mathbb{R}^d , for $d = 2, 3$. For two points $s, t \in \mathcal{F}$, a *geodesic path* between s and t within \mathcal{F} , denoted by $\Pi_{\mathcal{F}}(s, t)$, is a path from s to t completely lying inside \mathcal{F} and with minimum length. The *geodesic distance* between s and t , denoted by $d_{\mathcal{F}}(s, t)$, is the length of $\Pi_{\mathcal{F}}(s, t)$. In this paper, the subset \mathcal{F} in question is the *free space* of a set \mathbb{P} of k pairwise disjoint convex polytopes in \mathbb{R}^d , that is, $\mathcal{F} = \mathbb{R}^d \setminus \text{int} \bigcup_{P \in \mathbb{P}} P$. A path (line segment, point) is *free* (with respect to \mathbb{P}) if it lies in \mathcal{F} . So $\Pi_{\mathcal{F}}(s, t)$ is the shortest free path between s and t .

Let $\mathbb{P}(\tau)$ be a collection of pairwise disjoint deforming convex polytopes in \mathbb{R}^2 or \mathbb{R}^3 , each defined by the point-cloud model, and let $\mathcal{F}(\tau)$ be the free space of $\mathbb{P}(\tau)$. Our problem is to maintain a certain data structure as τ varies, so that at any τ , the geodesic distance between any two query points $s, t \in \mathcal{F}(\tau)$ can be reported.

Many existing data structures for geodesic-distance queries in static environments (to be reviewed shortly) are quite complicated and unlikely to render efficient kinetic data structures for dynamic environments. Given this situation and taking into account practical considerations, our goal in this paper is to design a kinetic data structure that meets the following criteria:

- (a) The number of events of the KDS is small, ideally nearly $O(n)$ in \mathbb{R}^2 and nearly $O(n^2)$ in \mathbb{R}^3 . In particular, one cannot maintain each polytope in $\mathbb{P}(\tau)$ explicitly as the point cloud deforms because the number of events for maintaining $\mathbb{P}(\tau)$ alone would be $\Omega(n^2)$ in \mathbb{R}^2 [2] and $\Omega(n^3)$ in \mathbb{R}^3 in the worst case.
- (b) The KDS provides a flexible tradeoff between the query time and the number of events.
- (c) The KDS handles motion changes and transient obstacles (i.e., obstacles in $\mathbb{P}(\tau)$ may be added or deleted) efficiently.

As a compromise, we allow that the data structure only reports a (reasonable) approximation of the geodesic distance, and that the query time can depend on the number of obstacles (but not their total complexity).

Related work. Geodesic-distance queries in a static environment have been extensively studied. Chiang and Mitchell [8] proposed a polynomial sized data structure that answers geodesic-distance query amid polygonal obstacles in \mathbb{R}^2 in $O(\log n)$ time; n is the total number of obstacle vertices. They also proposed tradeoffs

between space and query time. Chen [7] observed that an algorithm of Clarkson [9] can be turned into a data structure of size $O(n^2)$ to support $(1 + \varepsilon)$ -approximate geodesic-distance queries in $O(\varepsilon^{-1} \log n)$ time, for a fixed parameter ε . He also designed a data structure of size $O(n \log n)$ to answer $(6 + \varepsilon)$ -approximate geodesic-distance queries in $O(\log n)$ time (see also [5]). Very few results are known on geodesic-distance queries in \mathbb{R}^3 . Agarwal *et al.* [1] designed a data structure of size $O(n^6 m^{1+\delta})$, for $1 \leq m \leq n^2$ and for any $\delta > 0$, to store a convex polytope P with n vertices in \mathbb{R}^3 so that the geodesic-distance query for any two points on P can be answered in $O((\sqrt{n}/m^{1/4}) \log n)$ time. Har-Peled [11] considered the same problem but allowed $(1 + \varepsilon)$ -approximations for $\varepsilon > 0$, and showed that a $(1 + \varepsilon)$ -approximation of the geodesic distance between two query points can be reported in $O(\varepsilon^{-3/2} \log n + \varepsilon^{-3})$ time. Recently, Agarwal *et al.* [3] have developed data structure for answering geodesic-distance queries in \mathbb{R}^3 from a fixed source point.

We are not aware of any existing work on geodesic-distance queries in the dynamic environment. For an account of extensive work on collision detection and motion planning in dynamic environments, we refer the readers to [14, 18] and the references therein.

Our results. We present simple kinetic data structures for answering approximate geodesic-distance queries amidst a collection \mathbb{P} of k pairwise disjoint deforming convex polytopes in \mathbb{R}^2 and \mathbb{R}^3 . Each polytope $P \in \mathbb{P}$ is defined as the convex hull of a set of moving points, each moving along a bounded-degree algebraic trajectory. Let n be the total number of such points.

In \mathbb{R}^2 , for a prescribed parameter $\varepsilon > 0$, our kinetic data structure uses $O(n/\sqrt{\varepsilon})$ space and processes $O(\lambda_c(n)/\sqrt{\varepsilon})$ events in total, where $\lambda_c(n)$ is the maximum length of Davenport-Schinzel sequences of order c on n symbols and is nearly linear (throughout the paper, c denotes a constant integer related to the degree of the motion). Processing each event takes $O(\log^2 n)$ time. It can be used to report, in $O((k/\sqrt{\varepsilon}) \log(k/\varepsilon))$ time, a $(1 + \varepsilon)$ -approximation to the geodesic distance between two arbitrary query points s, t in the free space. In \mathbb{R}^3 , for a prescribed parameter $1 \leq m \leq n$, our data structure uses $O(n)$ space and processes $O(n\lambda_c(n/m))$ events in total, each of which can be handled in $O(\log^2 n)$ time. At any moment, given two query points s, t in the free space, the data structure is able to return, in $O(mk \log(n/m))$ time, an $O(k_{st})$ -approximation to the geodesic distance between s and t , where k_{st} is the number of polytopes intersected by the line segment st and is expected to be small in practice. Our data structures are simple enough to allow for points defining the polytopes to be inserted or deleted or change their motion in polylogarithmic time per event.

2 Geodesic Distance Queries in \mathbb{R}^2

For notational convenience, we will omit the time parameter τ when no confusion arises. Let $\mathbb{P} = \{P_1, \dots, P_k\}$ be a collection of k pairwise disjoint deforming convex polygons, where each P_i is defined as the convex hull of a set S_i of n_i moving points. Instead of explicitly maintaining each P_i , we maintain a certain “sketch” \tilde{P}_i of each

P_i , so that the geodesic distance between any two points s, t in the free space \mathcal{F} is approximately preserved. More formally, a set $\tilde{\mathbb{P}} = \{\tilde{P}_1, \dots, \tilde{P}_k\}$ of convex polygons is called an ε -*sketch* of \mathbb{P} if

- (i) $\tilde{P}_i \subseteq P_i$, for each $1 \leq i \leq k$;
- (ii) for any two points $s, t \in \mathcal{F}$, $d_{\mathcal{F}}(s, t) \leq (1 + \varepsilon)d_{\tilde{\mathcal{F}}}(s, t)$, where $\mathcal{F} \subseteq \tilde{\mathcal{F}}$ is the free space of $\tilde{\mathbb{P}}$.

Since $\mathcal{F} \subseteq \tilde{\mathcal{F}}$, $d_{\tilde{\mathcal{F}}}(s, t)$ is well-defined for any pair $s, t \in \mathcal{F}$ and $d_{\mathcal{F}}(s, t) \geq d_{\tilde{\mathcal{F}}}(s, t)$.

We construct an ε -sketch of \mathbb{P} as follows. Set $r = \lceil 2\pi/\sqrt{2\varepsilon} \rceil$. For $0 \leq j < r$, let $u_j = (\cos(j\sqrt{2\varepsilon}), \sin(j\sqrt{2\varepsilon}))$. The set $\mathcal{N} = \{u_j \mid 0 \leq j < r\}$ forms a uniform set of directions in \mathbb{R}^2 . For $1 \leq i \leq k$ and $0 \leq j < r$, let $p_i^j = \arg \max_{p \in S_i} \langle u_j, p \rangle$ denote the extremal point of S_i in the direction $u_j \in \mathcal{N}$. Let $\tilde{S}_i = \{p_i^j \mid 0 \leq j < r\}$, \tilde{P}_i is the convex hull of \tilde{S}_i and $\tilde{\mathbb{P}} = \{\tilde{P}_1, \dots, \tilde{P}_k\}$.

Maintaining $\tilde{\mathbb{P}}$ as \mathbb{P} deforms over time is straightforward. For each S_i and each $u_j \in \mathcal{N}$, we use a kinetic tournament¹ to keep track of the extreme point p_i^j . Note that $p_i^0, p_i^1, \dots, p_i^{r-1}$ are in convex position and naturally represent \tilde{P}_i in this order. (Although P_i deforms continuously, \tilde{P}_i may change discontinuously at certain time instances; a change of extremal point in some direction u_j may result in a discontinuous change in the shape of \tilde{P}_i .) When a point p is inserted or deleted or changes its trajectory, we update $O(1/\sqrt{\varepsilon})$ kinetic tournaments involving that point. A point cloud can also be added to or removed from the environment in a straightforward manner.

Lemma 2.1. $\tilde{\mathbb{P}}$ is an ε -sketch of \mathbb{P} .

Proof. Clearly, for $1 \leq i \leq k$, $\tilde{S}_i \subseteq S_i$ and as such $\tilde{P}_i \subseteq P_i$. Hence $\tilde{\mathbb{P}}$ satisfies (i). We next prove $\tilde{\mathbb{P}}$ also satisfies (ii).

Let s, t be two points in \mathcal{F} . Consider a geodesic path $\Pi = \Pi_{\tilde{\mathcal{F}}}(s, t)$. If $\Pi \subseteq \mathcal{F}$, then Π is also $\Pi_{\mathcal{F}}(s, t)$ and there is nothing more to prove. So from now on we assume that Π intersects $\tilde{\mathcal{F}} \setminus \mathcal{F}$.

By our construction, $\tilde{\mathcal{F}} \setminus \mathcal{F}$ consists of a set of convex polygons whose interiors are pairwise disjoint (see Figure 1). For each such polygon O , one of its edges called the *base* of O is an edge of \tilde{P}_i for some i , and the other edges called the *dome* of O come from a subsequence of edges of P_i . Let $p_i^j p_i^{j+1}$ be the base of O . Since p_i^j (resp., p_i^{j+1}) is an extreme point of P_i in direction u_j (resp., u_{j+1}), any point on the dome of O is extreme only in some direction between u_j and u_{j+1} on \mathbb{S}^1 . In other words, the outward unit normal of a point on the dome O lies in the interval $[u_j, u_{j+1}]$.

¹ A kinetic tournament [6] can be used to maintain the maximum (or minimum) of a set S of n moving points on the real line. The total number of events is $O(\lambda_c(n))$, each of which can be processed in $O(\log^2 n)$ time. Every point in S participates in $O(\log n)$ certificates, and hence a motion update can be performed in $O(\log^2 n)$ time. Similarly, a moving point can be inserted into or deleted from the kinetic tournament in $O(\log^2 n)$ time.

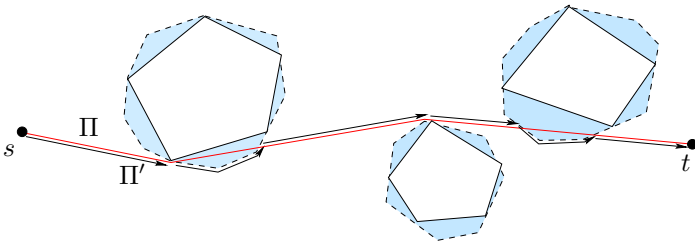


Fig. 1. Surgery on the path Π from s to t to make it free; shaded area represents $\tilde{\mathcal{F}} \setminus \mathcal{F}$, which consists of a set of convex polygons.

Let O be one of these polygons intersected by Π . Since O is convex and Π cannot cross the base of O , $\Pi \cap O$ is a line segment between two points p, q on the dome of O . We modify Π to bypass O , by replacing the segment pq on Π with the path Π_{pq} between p and q along the dome of O . Clearly, $\Pi_{pq} \subseteq \mathcal{F}$. This surgery increases the length of Π , but by not much. More precisely, Let z be the intersection of the two tangents of O at p and q respectively. By the discussion in the preceding paragraph, we have $\angle pzq \geq \pi - \sqrt{2\varepsilon}$. Hence,

$$\begin{aligned} |\Pi_{pq}| &\leq |pz| + |qz| \leq |pq| / \sin(\angle pzq/2) \leq |pq| / \sin(\pi/2 - \sqrt{\varepsilon}/2) \\ &\leq |pq| / (1 - \varepsilon/2) \leq (1 + \varepsilon)|pq|. \end{aligned}$$

Here, the first inequality follows from a standard convexity argument, the second inequality follows from elementary trigonometry, and the last inequality assumes $\varepsilon \leq 1$.

We perform the above surgery on Π for each of the polygons in $\tilde{\mathcal{F}} \setminus \mathcal{F}$ intersected by the original path Π . In the end, the new path Π' lies in \mathcal{F} . Therefore $d_{\mathcal{F}}(s, t) \leq |\Pi'| \leq (1 + \varepsilon)d_{\tilde{\mathcal{F}}}(s, t)$, and $\tilde{\mathbb{P}}$ is indeed an ε -sketch of \mathbb{P} . \square

The total number of vertices of $\tilde{\mathbb{P}}$ is $O(k/\sqrt{\varepsilon})$. For query points $s, t \in \mathcal{F}$, we use the algorithm of Hershberger and Suri [13] to compute $d_{\tilde{\mathcal{F}}}(s, t)$ in time $O(|\tilde{\mathbb{P}}| \log |\tilde{\mathbb{P}}|) = O((k/\sqrt{\varepsilon}) \log(k/\varepsilon))$ and return $(1 + \varepsilon)d_{\tilde{\mathcal{F}}}(s, t)$ which is a $(1 + \varepsilon)$ -approximation of $d_{\mathcal{F}}(s, t)$ by the preceding lemma.

Theorem 2.1. *Let $\mathbb{P} = \{P_1, \dots, P_k\}$ be a collection of pairwise disjoint deforming convex polygons in \mathbb{R}^2 , where each P_i is defined as the convex hull of a set of n_i moving points. Let $n = \sum_{i=1}^k n_i$. There is a kinetic data structure that reports, in $O((k/\sqrt{\varepsilon}) \log(k/\varepsilon))$ time, a $(1 + \varepsilon)$ -approximation to the geodesic distance between two arbitrary query points s, t in the free space. The data structure has $O(n/\sqrt{\varepsilon})$ size and processes $O(\lambda_c(n)/\sqrt{\varepsilon})$ events in total, each requiring $O(\log^2 n)$ time. A point (used for defining one of the convex hulls) can be inserted or deleted or change its motion in $O((1/\sqrt{\varepsilon}) \log^2 n)$ time.*

3 Geodesic-Distance Queries in \mathbb{R}^3

In this section we consider the geodesic-distance query problem amidst three-dimensional dynamic point clouds. We first describe the data structure for the case of one single point cloud; and then extend it to multiple point clouds.

3.1 Single Polytope

Consider a single polytope P defined as the convex hull of a set S of n moving points in \mathbb{R}^3 . We start by considering the special case in which the query points s, t both lie on the surface of P (*boundary query*) and then extend to the case in which s, t are two arbitrary points in the free space \mathcal{F} (*generic query*). In the former case, it is well known that $\Pi_{\mathcal{F}}(s, t)$ is also a path lying on the boundary of P , i.e., $d_{\mathcal{F}}(s, t) = d_{\partial P}(s, t)$.

Existing data structures for geodesic-distance query on the boundary of a convex polytope P [11] make use of the Dobkin-Kirkpatrick hierarchy [10] of P . However, as in the two-dimensional case, we do not want to explicitly maintain P explicitly, nor its Dobkin-Kirkpatrick hierarchy. The starting point of our algorithm is the work of Hershberger and Suri [12], in which a linear-time procedure is proposed for computing a constant-factor approximation for the geodesic-distance between two points $s, t \in \partial P$. Their procedure makes use of the two unit normals u_s and u_t at s and t respectively to guide the computation. Briefly, if the angle between u_s and u_t is small (say, less than $\pi/2$), then the Euclidean distance $\|st\|$ is a good approximation to their geodesic distance; otherwise, there is a point $p \in \partial P$ such that $\|sp\| + \|pt\|$ is a good approximation, and moreover u_p makes small angles with both u_s and u_t , where u_p is the normal to a plane containing p , avoiding the interior of P , and pointing away from P . In our case, since the polytope will not be explicitly maintained, we do not know the normals at s and t and therefore need a more careful design.

Data structure. For a unit vector $u \in \mathbb{S}^2$, we denote the plane $\langle p, u \rangle = 0$ ($p \in \mathbb{R}^3$) by h_u and the great circle $h_u \cap \mathbb{S}^2$ on \mathbb{S}^2 by g_u . For a set $X \subseteq \mathbb{R}^3$ and a unit vector $u \in \mathbb{S}^2$, we denote by $\downarrow_u(X)$ the projection of X onto h_u . On the positive hemisphere \mathbb{S}_+^2 (i.e., the closed hemisphere of \mathbb{S}^2 lying on or above the xy -plane), we choose a $(\pi/8)$ -net \mathcal{N} , that is, for any $u, v \in \mathcal{N}$, $\angle u, v \geq \pi/8$, and for any $u \in \mathbb{S}_+^2$, there is a $v \in \mathcal{N}$ so that $\angle u, v \leq \pi/8$ (here $\angle u, v$ denotes the angle between u and v). It can be shown that $|\mathcal{N}| = O(1)$. For each $u \in \mathcal{N}$, we fix arbitrarily a pair $\{u^x, u^y\}$ of orthogonal unit vectors in the plane h_u . We maintain the following information:

- (I1) for each $u \in \mathcal{N}$, the convex hull \mathcal{C}_u of $\downarrow_u(S)$ in the plane h_u ;
- (I2) for each \mathcal{C}_u , an auxiliary data structure \mathcal{D}_u so that given a point $p \in h_u$, it returns, in $O(\log n)$ time, the (at most) four edges of \mathcal{C}_u that the rays from p in directions $\pm u^x, \pm u^y$ first hit.

The data structure can be readily maintained under motion. Each \mathcal{C}_u can be maintained efficiently using a kinetic convex hull algorithm². The data structure \mathcal{D}_u is a balanced binary tree over the edges of \mathcal{C}_u . Since $|\mathcal{N}| = O(1)$, we obtain the following.

Lemma 3.1. *The data structure uses $O(n)$ space and can be maintained under motion in $O(\log^2 n)$ time per event, with a total of $O(n\lambda_c(n))$ events.*

Answering a boundary query. A face f of P is called *positive* with respect to a direction $u \in \mathbb{S}^2$ if $\langle u_f, u \rangle \geq 0$, where u_f is the outward unit normal of f , and *negative* if $\langle u_f, u \rangle \leq 0$. The positive faces form a connected component on ∂P called *positive component*, and similarly the negative faces form a *negative component*. The boundary between the positive and negative components consists of a sequence \mathcal{E}_u of edges of P called *horizon edges*. It can be shown that, for any $u \in \mathbb{S}^2$, $\downarrow_u(\mathcal{E}_u) = \mathcal{C}_u$, i.e., the projection of the horizon edges is the set of edges of the convex hull of $\downarrow_u(S)$.

For $u \in \mathbb{S}^2$, let $\mathbb{I}_u = \{p \in \mathbb{R}^3 \mid \downarrow_u(p) \text{ lies inside } \mathcal{C}_u\}$. The set \mathbb{I}_u forms an infinite prism containing P , and $\mathbb{I}_u \cap \mathcal{F}$ consists of two connected components separated by the horizon edges \mathcal{E}_u . Given two points $s, t \in \mathbb{I}_u \cap \mathcal{F}$, we say that u *separates s from t* if s, t lie in these two connected components respectively. As a convention, if either s or t lies on \mathcal{E}_u , then u also separates s and t . (When s, t are both on ∂P , this definition is equivalent to s, t lie in positive and negative components of ∂P respectively; the general definition presented here will be useful later for generic queries.)

The following procedure estimates the geodesic distance between two query points s and t lying on the boundary of P .

BOUNDARY_QUERY (s, t)

s, t : two points on the boundary of P

- (1) $\mathcal{N}_{st} \leftarrow \{u \in \mathcal{N} \mid u \text{ separates } s, t\}$;
- (2) **for** each direction $u \in \mathcal{N}_{st}$
 - $d[u] \leftarrow \|q \downarrow_u(s)\|_1$,
 - where q is the point closest to $\downarrow_u(s)$ on \mathcal{C}_u under the L_1 -norm;
- (3) $D \leftarrow \|st\| + \sqrt{2} \cdot \max_{u \in \mathcal{N}_{st}} d[u]$;
- (by convention, $\max_{u \in \mathcal{N}_{st}} d[u] = 0$ if $\mathcal{N}_{st} = \emptyset$)
- (4) **return** D ;

Since s, t are on the boundary of P , they lie inside $\mathbb{I}_u \cap \mathcal{F}$ for any $u \in \mathbb{S}^2$. So step (1) is well defined.

² A kinetic convex hull algorithm [4, 6] can be used to maintain the convex hull of a set S of n moving points in \mathbb{R}^2 . There are $O(n\lambda_c(n) \log n)$ events in total, each of which can be processed in $O(\log^2 n)$ time. In addition, each point participates in $O(\log n)$ certificates and hence a motion update can be performed in $O(\log^2 n)$ time. A moving point can also be inserted or deleted in $O(\log^2 n)$ time [4].

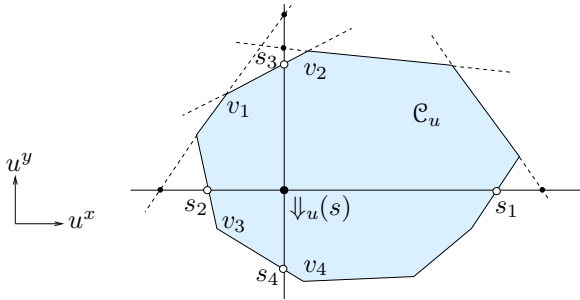


Fig. 2. Using (I2) to test whether u separates s, t , and to compute the closest point of $\Downarrow_u(s)$ on \mathcal{C}_u under L_1 -norm.

In step (1), we need to decide for a direction $u \in \mathcal{N}$ whether u separates s from t . This can be done as follows. Let v_1v_2 and v_3v_4 be the two edges of \mathcal{C}_u that $\Downarrow_u(s)$ projects onto in directions $\pm u^y$ (see Figure 2). Let $p_i \in S$ be such that $\Downarrow_u(p_i) = v_i$, for $i = 1, 2, 3, 4$. By examining whether s lies above or below the tetrahedron $p_1p_2p_3p_4$ along direction u , one can decide which component of $\mathbb{I}_u \cap \mathcal{F}$ contains s . (Recall that $s \in \mathcal{F}$, so it does not lie inside the tetrahedron.) A similar statement holds for t . Hence deciding whether u separates s, t can be done in $O(\log n)$ time using (I2).

Since $|\mathcal{N}| = O(1)$, step (1) takes $O(\log n)$ time and step (3) takes $O(1)$ time. The lemma below shows that step (2) also takes $O(\log n)$ time using (I2). Hence the total time for answering a query is $O(\log n)$.

Lemma 3.2. *Let p be any point lying inside \mathcal{C}_u . Let s_1, \dots, s_4 be the four projections of p onto \mathcal{C}_u in directions $\pm u^x, \pm u^y$. In the plane h_u equipped with the coordinate frame $\{u^x, u^y\}$, one of s_1, \dots, s_4 is a point on \mathcal{C}_u that is closest to p under L_1 -norm.*

Proof. For any line $\ell \subset h_u$, let p_1 and p_2 be the projections of p onto ℓ in direction u^x (or $-u^x$) and u^y (or $-u^y$) respectively. It is easy to verify that if ℓ makes an angle $\leq \pi/4$ with u^x or $-u^x$, then p_2 is the point on ℓ closest to p (under L_1 -norm), and otherwise p_1 is the closest. So either p_1 or p_2 realizes the closest point to p . Now, consider the projections of p in direction u^x to all the lines containing an edge of \mathcal{C}_u . The point s_1 is closest to p among all such projections (see Figure 2). Similarly, s_2 (resp., s_3, s_4) is the point closest to p among all projections of p onto these lines in direction $-u^x$ (resp., $u^y, -u^y$). Hence, the closest point on \mathcal{C}_u to p must be one of s_1, \dots, s_4 . \square

Next we show that BOUNDARY_QUERY indeed produces a constant-factor approximation to $d_{\partial P}(s, t)$. Observe that, if two points $s, t \in \partial P$ are separated along a direction $u \in \mathcal{N}$, then s lies on the positive component and t lies on the negative component, or vice versa. In particular, if there are outward unit normals u_s and u_t of P at s and t that lie on different sides of the great circle g_u on \mathbb{S}^2 , then u separates s from t .

Lemma 3.3. $D \leq 3 \cdot d_{\partial P}(s, t)$.

Proof. If $\mathcal{N}_{st} = \emptyset$, then trivially $D = \|st\| \leq d_{\partial P}(s, t)$. So assume $\mathcal{N}_{st} \neq \emptyset$. Let u be any direction in \mathcal{N}_{st} . Since u separates s, t , one of s, t lies on the positive component and the other lies on the negative component. Hence by continuity, $\Pi_{\partial P}(s, t)$ intersects \mathcal{E}_u at some point p . Note that $\downarrow_u(p) \in \mathcal{C}_u$. Let q be defined as in (2) of BOUNDARY_QUERY. Then,

$$\begin{aligned} d_{\partial P}(s, t) &\geq \|sp\| \geq \|\downarrow_u(s) \downarrow_u(p)\| \geq \|\downarrow_u(s) \downarrow_u(p)\|_1 / \sqrt{2} \\ &\geq (1/\sqrt{2}) \cdot \|q \downarrow_u(s)\|_1 = (1/\sqrt{2}) \cdot d[u]. \end{aligned}$$

Since trivially $d_{\partial P}(s, t) \geq \|st\|$, we have

$$\|st\| + \sqrt{2} \cdot d[u] \leq 3 \cdot d_{\partial P}(s, t).$$

As this is true for any $u \in \mathcal{N}_{st}$, we have $D \leq 3 \cdot d_{\partial P}(s, t)$ as claimed. □

For a plane $h \subseteq \mathbb{R}^3$ avoiding P , we use h^+ to denote the halfspace bounded by h and containing P . For two non-parallel planes h_1, h_2 both avoiding P , the boundary of $h_1^+ \cap h_2^+$ is called a *wedge*, and the dihedral angle of h_1, h_2 in the quadrant $h_1^+ \cap h_2^+$ is the *angle* of this wedge. It is shown in [12] that, for any two points p, q on a wedge W of angle θ , $d_W(p, q) \leq \|pq\| / \sin(\theta/2)$.

Lemma 3.4. $D \geq (\sin(\pi/16) / \sqrt{2}) \cdot d_{\partial P}(s, t)$.

Proof. Let u_s, u_t be the outward unit normals of P at s and t respectively (if the normal is not unique, choose one arbitrarily). We first consider the case in which the angle between u_s and u_t (denoted by $\angle u_s, u_t$) is at most $\pi/2$. Let W be the wedge formed by the two tangent planes of P at s and t with normals u_s and u_t respectively. The angle of W is $\pi - \angle u_s, u_t \geq \pi/2$. Therefore,

$$d_{\partial P}(s, t) \leq d_W(s, t) \leq \|st\| / \sin(\pi/4) = \sqrt{2} \cdot \|st\|.$$

Hence $D \geq \|st\| \geq (1/\sqrt{2}) \cdot d_{\partial P}(s, t)$.

Next consider the case $\angle u_s, u_t \geq \pi/2$. Let γ be the middle point on the geodesic arc between u_s, u_t on \mathbb{S}^2 , and $\eta \in \mathbb{S}^2_+$ be the direction orthogonal to the great circle of \mathbb{S}^2 passing through γ . Since \mathcal{N} is a $(\pi/8)$ -net of \mathbb{S}^2_+ , there exists a direction $u \in \mathcal{N}$ for which $\angle u, \eta \leq \pi/8$. Using elementary spherical geometry, it can be shown that u_s and u_t lie on different sides of g_u and thus u separates s, t ; furthermore, the angle between u_s and the plane h_u is at least $(1/2) \cdot \angle u_s, u_t - \pi/8 \geq \pi/8$, implying that $\angle u_s, v \leq \pi - \pi/8$ for all $v \in g_u$.

Let p be a point on \mathcal{E}_u such that $\downarrow_u(p) = q$ for q defined in (2) of BOUNDARY_QUERY. Since $p \in \mathcal{E}_u$, there is an outward unit normal u_p at p such that $u_p \in g_u$; in particular, $\angle u_s, u_p \leq \pi - \pi/8$ by the preceding discussion. Let W be the wedge formed by the two tangent planes of P at s and p with normals u_s and u_p respectively. The angle of W is $\pi - \angle u_s, u_p \geq \pi/8$. So,

$$d_{\partial P}(s, p) \leq d_W(s, p) \leq \|sp\| / \sin(\pi/16).$$

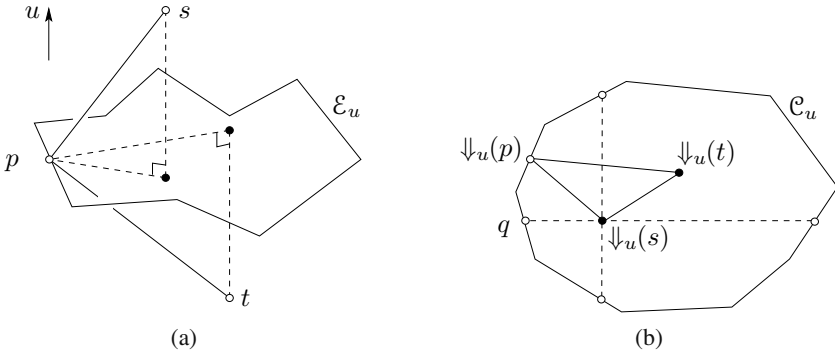


Fig. 3. Illustration for the proof of Lemma 3.4.

Similarly, $d_{\partial P}(p, t) \leq \|pt\| / \sin(\pi/16)$. Then we have

$$\begin{aligned}
 d_{\partial P}(s, t) &\leq d_{\partial P}(s, p) + d_{\partial P}(p, t) \\
 &\leq (\|sp\| + \|pt\|) / \sin(\pi/16) \\
 &\leq (\|\Downarrow_u(s) \Downarrow_u(p)\| + \|\Downarrow_u(t) \Downarrow_u(p)\| + \langle s - t, u \rangle) / \sin(\pi/16) \\
 &\leq (2 \cdot \|\Downarrow_u(s) \Downarrow_u(p)\| + \|\Downarrow_u(s) \Downarrow_u(t)\| + \langle s - t, u \rangle) / \sin(\pi/16) \\
 &\leq (2 \cdot \|\Downarrow_u(s) \Downarrow_u(p)\| + \sqrt{2} \cdot \|st\|) / \sin(\pi/16) \\
 &\leq (2 \cdot \|\Downarrow_u(s) \Downarrow_u(p)\|_1 + \sqrt{2} \cdot \|st\|) / \sin(\pi/16) \\
 &= (2 \cdot d[u] + \sqrt{2} \cdot \|st\|) / \sin(\pi/16).
 \end{aligned}$$

Hence, $D \geq \|st\| + \sqrt{2} \cdot d[u] \geq (\sin(\pi/16) / \sqrt{2}) \cdot d_{\partial P}(s, t)$. □

By Lemmas 3.3 and 3.4, it immediately follows that:

Lemma 3.5. *BOUNDARY_QUERY(s, t) returns a constant-factor approximation to $d_{\partial P}(s, t)$ in $O(\log n)$ time.*

Answering a generic query. Now we generalize the above boundary query to generic queries in which s, t are two arbitrary points in \mathcal{F} . The data structure remains the same as in (I1) and (I2). We only have to slightly change line (1) of the query procedure BOUNDARY_QUERY to the following.

$$(1') \mathcal{N}_{st} \leftarrow \{u \in \mathcal{N} \mid u \text{ separates } s, t, \text{ or } \Downarrow_u(s) \text{ or } \Downarrow_u(t) \text{ lies outside } \mathcal{C}_u\};$$

Note that (I2) can be used to determine whether a point $p \in h_u$ lies inside \mathcal{C}_u or not. So (1') takes $O(\log n)$ time. The resulting query procedure is GENERIC_QUERY.

Lemma 3.6. *GENERIC_QUERY(s, t) returns a constant-factor approximation to $d_{\mathcal{F}}(s, t)$ in $O(\log n)$ time.*

Proof. We first prove the following claim: $d_{\mathcal{F}}(s, t) = d_{\partial P'}(s, t)$, where $P' = \text{conv} S \cup \{s, t\}$. When either s or t does not lie on the boundary of P , there are two cases. If

s and t are visible to each other, then clearly $\Pi_{\mathcal{F}}(s, t)$ is the line segment between s and t , which also appears as an edge on the boundary of P' . If s and t are not visible, then $\Pi_{\mathcal{F}}(s, t)$ consists of three components: a line segment from s to a point $s' \in \partial P$, a geodesic path from s' to a point $t' \in \partial P$ (which lies on the boundary of P), and a line segment from t' to t . Note that no points in the interior of the path $\Pi_{\partial P}(s', t')$ are visible to either s or t , as otherwise $\Pi_{\mathcal{F}}(s, t)$ can be shortcutted. Hence $\Pi_{\mathcal{F}}(s, t)$ must lie on the boundary of P' , and the claim follows.

Therefore, to compute $d_{\mathcal{F}}(s, t)$, it suffices to compute $d_{\partial P'}(s, t)$. Imagine running BOUNDARY_QUERY (s, t) on P' . Lemma 3.5 then guarantees that it would return a constant-factor approximation. Let \mathcal{N}'_{st} be the set produced at line (1) of its execution, and \mathcal{N}_{st} be the set produced at line (1') of GENERIC_QUERY. Consider a direction $u \in \mathcal{N}$. If both $\downarrow_u(s)$ and $\downarrow_u(t)$ lie inside \mathcal{C}_u , then clearly $u \in \mathcal{N}_{st}$ if and only if $u \in \mathcal{N}'_{st}$. If either $\downarrow_u(s)$ or $\downarrow_u(t)$ lies outside \mathcal{C}_u , we have $u \in \mathcal{N}_{st}$; but at the same time, one of $\downarrow_u(s), \downarrow_u(t)$ must lie on the corresponding \mathcal{C}'_u for P' and therefore $u \in \mathcal{N}'_{st}$. Hence, \mathcal{N}'_{st} is the same as \mathcal{N}_{st} . The correctness of GENERIC_QUERY then follows. \square

Remark. It is possible to modify GENERIC_QUERY so that it also reports a free path between s, t whose length is within a constant factor of $d_{\mathcal{F}}(s, t)$, although P is not explicitly maintained. We omit the details.

3.2 Tradeoffs

So far we have described a kinetic data structure for the approximate geodesic-distance query problem on the boundary of a single polytope, which uses $O(n)$ space, processes $O(n\lambda_c(n))$ events, each requiring $O(\log^2 n)$ processing time, and answers queries in $O(\log n)$ time. These performance bounds are favorable when there are many queries. However, when the number of queries is expected to be small, then a scheme to trade query efficiency for kinetic maintenance efficiency is desirable. We present such a scheme by using generalized linear programming [15].

Let m be an adjustable integer parameter between 1 and n . We divide S into m groups S_1, \dots, S_m , each of size n/m (assume for the sake of simplicity that n is a multiple of m). For each $u \in \mathcal{N}$, instead of maintaining \mathcal{C}_u directly as in (II), we maintain the convex hull $\mathcal{C}_{u,i}$ of $\downarrow_u(S_i)$ in the plane h_u , for each $i = 1, \dots, m$, using the kinetic convex hull algorithm. Clearly, the total size of the data structure remains $O(n)$. The total number of events is $m \cdot O((n/m)\lambda_c(n/m)) = O(n\lambda_c(n/m))$, and the time to process each event remains $O(\log^2 n)$.

We next explain how to compute the projection of an arbitrary point $p \in h_u$ onto \mathcal{C}_u in direction u^y , in $O(m \log(n/m))$ time using the $\mathcal{C}_{u,i}$'s. By handling each of the other directions $-u^y, \pm u^x$ similarly, we thereby fulfill (I2). The query times of both BOUNDARY_QUERY and GENERIC_QUERY are dominated by the query time provided here.

For simplicity, assume that h_u is xy -plane, p is the origin, and u^y is $+y$ -axis. We first describe a simple but slower procedure that runs in $O(m^2 \log(n/m))$ time, which will become useful later. We only consider the *upper chain* $\mathcal{C}_{u,i}^+$ of each $\mathcal{C}_{u,i}$

in direction u^y , that is, the upper part of $\mathcal{C}_{u,i}$ lying between its two extreme vertices along $\pm x$ -axis. We further divide each $\mathcal{C}_{u,i}^+$ into two sub-chains, the left chain $\mathcal{L}_{u,i}^+$ which lies to the left of $+y$ -axis, and the right chain $\mathcal{R}_{u,i}^+$ which lies to the right of $+y$ -axis. If there is an edge of $\mathcal{C}_{u,i}^+$ intersecting $+y$ -axis, it does not belong to either subchains. For each pair of left chain $\mathcal{L}_{u,i}^+$ and right chain $\mathcal{R}_{u,j}^+$, since they are disjoint, we can use an algorithm of Overmars and van Leeuwen [16] to compute their common outer tangent line $\ell_{i,j}$ in $O(\log |\mathcal{L}_{u,i}^+| + \log |\mathcal{R}_{u,j}^+|) = O(\log(n/m))$ time. Let $p_{i,j}$ be the intersection of $\ell_{i,j}$ with $+y$ -axis. We can compute all such $p_{i,j}$'s in time $O(m^2 \log(n/m))$.

Let $v_1 v_2$ be the edge of \mathcal{C}_u that contains the sought projection of p , and ℓ be the line containing $v_1 v_2$. Assume without loss of generality that v_1 lies to the left of $+y$ -axis and v_2 to the right. Consider the left chain $\mathcal{L}_{u,a}^+$ that v_1 belongs to as a vertex, and the right chain $\mathcal{R}_{u,b}^+$ that v_2 belongs to as a vertex. Observe that the line ℓ is tangent to both $\mathcal{L}_{u,a}^+$ and $\mathcal{R}_{u,b}^+$ and thus is their common tangent. As such, the intersection $p_{a,b}$ of $\ell_{a,b}$ (i.e., ℓ) with $+y$ -axis is exactly the sought projection of p . For a common tangent $\ell_{i,j}$ other than $\ell_{a,b}$, observe that $p_{i,j}$ belongs to $\text{conv} \mathcal{L}_{u,i}^+ \cup \mathcal{R}_{u,i}^+ \subseteq \text{conv} \downarrow_u(S)$ and hence lies below $p_{a,b}$ on $+y$ -axis. Thus, the highest point on $+y$ -axis among all $p_{i,j}$'s is the projection of p onto \mathcal{C}_u in direction $+y$.

Next we improve the time for finding the projection to $O(m \log(n/m))$, by formulating it as an LP-type problem. Consider the following optimization problem specified by pairs (\mathcal{H}, w) , where $\mathcal{H} = \{\downarrow_u(S_i) \mid i = 1, \dots, m\}$ and $w : 2^{\mathcal{H}} \rightarrow \mathbb{R}$ is a function that maps each subset $\mathcal{G} \subseteq \mathcal{H}$ to the y -coordinate of the projection of p (the origin) onto $\text{conv} \bigcup_{X \in \mathcal{G}} X$ in direction $+y$ ($w(\mathcal{G}) = -\infty$ if the projection does not exist). For a subset $\mathcal{G} \subseteq \mathcal{H}$ with $w(\mathcal{G}) > -\infty$, a *basis* of \mathcal{G} is a minimal subset \mathcal{B} of \mathcal{G} with $w(\mathcal{B}) = w(\mathcal{G})$. The goal is to compute a basis \mathcal{B} of \mathcal{H} , from which the value of $w(\mathcal{H})$ then can be computed from $w(\mathcal{B})$. We verify the following properties of (\mathcal{H}, w) :

Finite basis: every subset of \mathcal{H} has a basis of size at most two. For a subset $\mathcal{G} \subseteq \mathcal{H}$ with $w(\mathcal{G}) > -\infty$, Let $v_1 v_2$ be the edge on $\text{conv} \bigcup_{X \in \mathcal{G}} X$ that p projects onto in direction $+y$. Let $X_1, X_2 \in \mathcal{G}$ be elements of \mathcal{G} that contain v_1, v_2 respectively. Then clearly $\{X_1, X_2\}$ is a basis of \mathcal{G} . See Figure 4.

Monotonicity: for any $\mathcal{F} \subseteq \mathcal{G} \subseteq \mathcal{H}$, $w(\mathcal{F}) \leq w(\mathcal{G})$. This is because $\mathcal{F} \subseteq \mathcal{G}$ implies $\text{conv} \bigcup_{X \in \mathcal{F}} X \subseteq \text{conv} \bigcup_{X \in \mathcal{G}} X$.

Locality: for any $\mathcal{F} \subseteq \mathcal{G} \subseteq \mathcal{H}$ with $w(\mathcal{G}) = w(\mathcal{F}) > -\infty$, and any $X \in \mathcal{H}$, $w(\mathcal{G}) < w(\mathcal{G} \cup \{X\})$ implies $w(\mathcal{F}) < w(\mathcal{F} \cup \{X\})$. Let ℓ be the line containing the edge of $\text{conv} \bigcup_{X \in \mathcal{G}} X$ that p projects onto. The condition $w(\mathcal{G}) < w(\mathcal{G} \cup \{X\})$ implies that X contains a point lying above ℓ . By $w(\mathcal{F}) = w(\mathcal{G})$ and $\mathcal{F} \subseteq \mathcal{G}$, it can be shown that ℓ is also the line containing the edge of $\text{conv} \bigcup_{X \in \mathcal{F}} X$ that p projects onto. It follows that $w(\mathcal{F}) < w(\mathcal{F} \cup \{X\})$.

Matoušek *et al.* [15] showed that such an optimization problem (\mathcal{H}, w) can be solved in $O(|\mathcal{H}| \cdot T + E \cdot \log |\mathcal{H}|)$ expected time, where T is the time to test whether $w(\mathcal{B}) = w(\mathcal{B} \cup \{X\})$ for some basis \mathcal{B} and element $X \in \mathcal{H}$, and E is the time to compute a basis of $\mathcal{B} \cup \{X\}$ for some basis \mathcal{B} and element $X \in \mathcal{H}$. In our context,

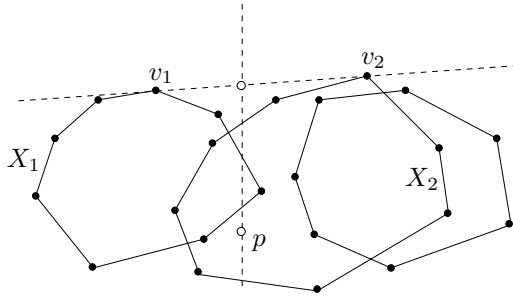


Fig. 4. Reducing computing the projection to an LP-type problem.

$|\mathcal{H}| = m$. Furthermore, any basis consists of at most two elements in \mathcal{H} , both T and E are in $O(\log(n/m))$, by applying the aforementioned slower procedure. Hence, the projection of p on \mathcal{C}_u can be computed in $O(m \log(n/m))$ expected time.

Theorem 3.1. *Let P be the convex hull of a set of n points in \mathbb{R}^3 under algebraic motion. For any parameter $1 \leq m \leq n$, there is a kinetic data structure that can be used to report, in $O(m \log(n/m))$ expected time, a constant-factor approximation to the geodesic distance between two arbitrary query points s, t in the free space. The data structure has $O(n)$ size and processes $O(n \lambda_c(n/m))$ events in total, each requiring $O(\log^2 n)$ time. A point (used for defining one of the convex hulls) can be inserted or deleted or change its motion in $O(\log^2 n)$ time.*

3.3 Multiple Polytopes

Let $\mathbb{P} = \{P_1, \dots, P_k\}$ be a collection of k pairwise disjoint deforming convex polytopes in \mathbb{R}^3 , where each P_i is the convex hull of a set of n_i moving points. Set $n = \sum_{i=1}^k n_i$. We maintain a separate data structure of Theorem 3.1 for each P_i . The total space of these data structures is $\sum_i O(n_i) = O(n)$, and the total number of events is $\sum_i O(n_i \lambda_c(n_i/m)) = O(n \lambda_c(n/m))$.

Let s, t be two query points in the free space \mathcal{F} . As observed in [12], one can obtain an $O(k)$ -approximation to the geodesic distance between s, t by summing up the query results for each of the k separate data structures. We can improve the approximation factor to $O(k_{st})$ by a simple trick, where k_{st} is the number of polytopes in \mathbb{P} intersected by the line segment st , as follows: among the k returned distances, we simply add up those whose values are $\Omega(\|st\|)$. We omit the details.

Theorem 3.2. *Let \mathbb{P} be a collection of k deforming obstacles each of which is the convex hull of a dynamic point cloud under algebraic motion. Let n be the total number of points in all the point clouds. For any parameter $1 \leq m \leq n$, there is a kinetic data structure that can be used to report, in $O(mk \log(n/m))$ expected time, a $O(k_{st})$ -approximation to the geodesic distance between two arbitrary query points s, t in the free space. The data structure has $O(n)$ size and processes $O(n \lambda_c(n/m))$*

events in total, each requiring $O(\log^2 n)$ time. A point (used for defining one of the convex hulls) can be inserted or deleted or change its motion in $O(\log^2 n)$ time.

References

1. Agarwal, P.K., Aronov, B., O'Rourke, J., Schevon, C.: Star unfolding of a polytope with applications. *SIAM J. Comput.* 26, 1689–1713 (1997)
2. Agarwal, P.K., Guibas, L., Hershberger, J., Veach, E.: Maintaining the extent of a moving point set. *Discrete Comput. Geom.* 26, 353–374 (2001)
3. Agarwal, P.K., Sharathkumar, R., Yu, H.: Approximate Euclidean shortest paths amid convex obstacles. In: Proc. 20th ACM-SIAM Sympos. Discrete Algorithms (to appear)
4. Alexandron, G., Kaplan, H., Sharir, M.: Kinetic and dynamic data structures for convex hulls and upper envelopes. *Comput. Geom. Theory Appl.* 36, 144–158 (2007)
5. Arikati, S., Chen, D., Chew, L., Das, G., Smid, M., Zaroliagis, C.: Planar spanners and approximate shortest path queries among obstacles in the plane. In: Díaz, J. (ed.) *ESA 1996*. LNCS, vol. 1136, pp. 514–528. Springer, Heidelberg (1996)
6. Basch, J., Guibas, L.J., Hershberger, J.: Data structures for mobile data. *J. Algorithms* 31, 1–28 (1999)
7. Chen, D.: On the all-pairs Euclidean short path problem. In: Proc. 6th Annu. ACM-SIAM Sympos. Discrete Algorithms, pp. 292–301 (1995)
8. Chiang, Y.-J., Mitchell, J.S.B.: Two-point Euclidean shortest path queries in the plane. In: Proc. 10th Annu. ACM-SIAM Sympos. Discrete Algorithms, pp. 215–224 (1999)
9. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: Proc. 19th Annu. ACM Sympos. Theory Comput., pp. 56–65 (1987)
10. Dobkin, D.P., Kirkpatrick, D.G.: Determining the separation of preprocessed polyhedra — a unified approach. In: Paterson, M. (ed.) *ICALP 1990*. LNCS, vol. 443, pp. 400–413. Springer, Heidelberg (1990)
11. Har-Peled, S.: Approximate shortest-path and geodesic diameter on convex polytopes in three dimensions. *Discrete Comput. Geom.* 21, 217–231 (1999)
12. Hershberger, J., Suri, S.: Practical methods for approximating shortest paths on a convex polytope in \mathbb{R}^3 . *Comput. Geom. Theory Appl.* 10, 31–46 (1998)
13. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.* 28, 2215–2256 (1999)
14. Ling, M., Manocha, D.: Collision and proximity queries. In: Goodman, J., O'Rourke, J. (eds.) *Handbook of Discrete and Computational Geometry*, 2nd edn., pp. 787–808. CRC Press, Boca Raton (2004)
15. Matoušek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. *Algorithmica* 16, 498–516 (1996)
16. Overmars, M., van Leeuwen, J.: Maintenance of configurations in the plane. *J. Comput. Syst. Sci.* 23, 166–204 (1981)
17. Reif, J., Sharir, M.: Motion planning in the presence of moving obstacles. *J. Assoc. Comput. Mach.* 41, 764–790 (1994)
18. van den Berg, J.: Path Planning in Dynamic Environments. PhD thesis, Utrecht University (2007)