Approximating Nearest Neighbor Among Triangles in Convex Position

Yusu Wang*

Abstract

Given a query point, locating its nearest neighbor among a set of objects is a fundamental problem arising in many application fields. While nearest neighbor queries among a set of points has been widely studied and relatively well understood, much less is known about such queries in a collection of other types of objects, such as triangles. In this paper, we consider the problem of approximating the nearest neighbor of a query point among a set of triangles X that are in convex position, namely, X is a subset of faces of some convex polyhedron in three space. The problem is useful in, say, convex decomposition of surfaces. We present an efficient algorithm to ε-approximate the closest distance from any point to such a set of n triangles in $O(\log^2 n/\varepsilon^2)$ time, using a data structure of size $O(n/\varepsilon^2)$ that can be constructed in $O(n \log n/\varepsilon^2)$ time.

1 Introduction

Given a query point, locating its nearest neighbor among a set of objects is a fundamental problem arising in many applications fields, including computer graphics, pattern recognition, robot motion planning, information retrieval, machine learning, and data mining. For example, in computer animation, collision detection tests usually boil down to checking whether a query point intersects any object in the scene or not.

The most widely studied version of this problem is to find the nearest neighbor of a query point among a set of points under the Euclidean distance. For n points in the plane, there are several methods to answer such a nearest neighbor query in $O(\log n)$ time after preprocessing the input points into a data structure of size $O(n)$ in $O(n \log n)$ time [14, 18]. For a set of points in $\mathbb{R}^d$, the problem can be solved in $O(d^3 \log n)$ query time using a data structure of size $O(n^{\lceil n/2 \rceil + \varepsilon})$ [7, 15]. Time/space trade-offs can be obtained by formulating the nearest-neighbor query among points in $\mathbb{R}^d$ as a ray-shooting problem in a convex polytope in $\mathbb{R}^{d+1}$ [4], achieving roughly $O(n/m^{1/\lceil d/2 \rceil})$ query time using $O(m)$ space.

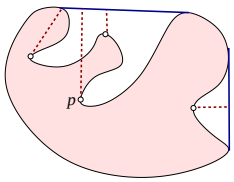
Given the high time or space complexity of this problem in high dimensions, the problem of finding ***approximate nearest neighbors*** has been considered [11]. In particular, given a query point p, let $\mathrm{N}_X(p)$ denote the nearest neighbor of p in set X. An ***ε-approximate nearest neighbor*** (ε-ANN) of p in X is a point $q \in X$ such that $\mathrm{d}(p, q) \leq (1 + \varepsilon)\mathrm{d}(p, \mathrm{N}_X(p))$, where $\mathrm{d}(x, y)$ is the Euclidean distance between x and y. Arya et al. [6] presented an algorithm that answers each ε-ANN query in $O(\log(n/\varepsilon))$ time by constructing an approximate Voronoi diagram of size $O(n/\varepsilon^{d-1})$ for the input points. The above algorithms all have exponential dependence on the dimension of space d, making them hardly practical when d is large (say greater than 20). This is the so-called

*Department of Computer Science and Engineering, Ohio State University, Columbus, OH 43016; *yusu@cse.ohio-state.edu*.

curse of dimensionality behavior, and much research has been done to reduce its influence; see the survey paper [11].

Much less is known for finding nearest neighbors, exactly or approximately, among objects other than point sets. If the input is the set of simplices (i.e, vertices, edges, and triangles) of a convex polyhedron P in $\mathbb{R}^3$, and the query point p is outside P, then the nearest neighbor of p can be computed exactly in $O(\log n)$ time by using the Dobkin-Kirkpatrick hierarchical representation of P which has $O(n)$ size [8]. If the query point is inside P, then an ε-ANN query can be answered in $O(\log n/\varepsilon^2)$ time by performing $O(1/\varepsilon^2)$ number of ray-shooting queries within a convex polytope, using a data structure of size $O(n)$ [9]. If the input is a set of disjoint polyhedra, then Koltun and Sharir [12] showed that an ε-ANN can be computed in $O(\log(n/\varepsilon))$ time using a data structure of near quadratic size, by building their Voronoi diagram under some convex distance function of constant complexity.

In this paper, we consider a collection X of n points, segments, and triangles in three-dimensional space that are in ***convex position***; namely, X is a subset of some convex polyhedron in $\mathbb{R}^3$. We also call X a ***convex cover***. We show that after pre-processing X into a data structure of size $O(n/\varepsilon^2)$, we can answer any ε-ANN query in $O(\log^2 n/\varepsilon^2)$ time. We remark that straightforward extensions of previous approaches using Dobkin-Kirkpatrick hierarchy or ray-shooting queries fail in this scenario, although the result by Koltun and Sharir [12] still holds, with a data structure of near quadratic size.



This ***nearest neighbor to convex cover problem*** is partly motivated by developing shape descriptors for measuring the concavity of a point on a surface, which has applications in graphics, such as computing the convex decomposition of an input polyhedron [13], and in computational biology, where it is important to segment molecular surfaces to capture cavities [16]. In particular, given a surface M, let X be the closure of $\mathrm{CH}(M) \setminus M$, where $\mathrm{CH}(N)$ is the convex hull of M. Given a point $p \in M$, one way to measure its ***concavity*** is by its closest distance to this convex cover X. See figure on the right for a simplified example in the plane. There are other more descriptive ways to capture the concavity, such as the elevation function proposed in [1], or measuring the shortest *collision-free* distance from p to X. However, they are much harder to compute.

Finally, we remark that the nearest neighbor problem has many close relatives in computational geometry, such as the Voronoi diagram, ray shooting problem, and range searching problem. We refer the readers to [2, 11, 17] for reviews in some related fields.

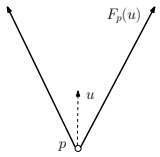
2 Algorithm Overview

Given a convex cover $S \subset \mathbb{R}^3$ of size n, let V, E, and T denote the set of vertices, edges and triangles of S, respectively. Let $\delta_S(p) = \mathrm{d}(p, \mathrm{N}_S(p))$ denote the shortest distance from a point $p \in \mathbb{R}^3$ to S; S may be omitted from notations when its choice is clear. The goal is to preprocess S into a data structure of size $O(n/\varepsilon^2)$ so that it can ε-approximate $\delta(p)$ in $O(\log^2 n/\varepsilon^2)$ time for any query point p.

Let $\gamma_p(u)$ represent the *ray* originated from p in direction $u \in \mathbb{S}^2$, where $\mathbb{S}^2$ is the sphere of directions. One way to view $\mathrm{N}(p)$ is that it is the closest intersection point between S and all possible rays originated from p. The main idea of our approach is to approximate $\mathrm{N}(p)$ by considering only a small number of discrete ray directions. In particular, we consider the following concept.

A simplicial cone cover. In $\mathbb{R}^d$, d hyperplanes in general position intersect at a point o. A *simplicial cone* is the intersection of d half-spaces bounded by these hyperplanes, and o is called

its *apex*. For $d = 3$ (as is in our case), each simplicial cone is bounded by three sides, called *fans*, and three *rays*; o is also the ***apex*** of these fans and rays. Given any $0 < \theta \leq \pi$, one can compute a collection of $O(1/\theta^{d-1})$ number of simplicial cones $\mathbb{C}$, called a *θ-simplicial cone cover* [20], such that (i) the apex of each cone is at the origin, (ii) the angular diameter of each cone is at most θ, and (iii) the union of cones from $\mathbb{C}$ cover $\mathbb{R}^d$. In the remainder of this paper, we assume that we are given a θ-simplicial cone cover $\mathbb{C}$ of $\mathbb{R}^3$ of size $O(1/\theta^2)$, where the exact value of θ will be specified later.



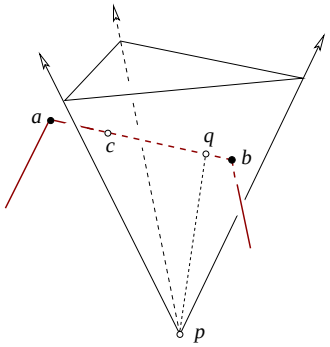
Now given a fan, we refer to the direction of bisection as its ***axis***, or its ***fan direction***. Denote by $\mathbb{C}_p$ the translation of $\mathbb{C}$ to apex p. Note that once $\mathbb{C}_p$ is fixed, then the fan direction u decides the fan uniquely, which we denote by $F_p(u)$. See figure on the right. Furthermore, for any $p \in \mathbb{R}^3$, the set of fan directions and the set of ray directions from $\mathbb{C}_p$ are the same as those sets from $\mathbb{C}$, respectively. We use $\Gamma_f \subset \mathbb{S}^2$ and $\Gamma_r \subset \mathbb{S}^2$ to represent these two sets; $|\Gamma_f| = O(\Gamma_r) = O(1/\theta^2)$.

Finally, for simplicity of exposition, we assume that we are given a convex polyhedron $\mathcal{H}$ of size $O(n)$ with S being its subset. Note that if $\mathcal{H}$ is not given, then we can compute such a polyhedron in $O(n \log n)$ time by constructing the convex hull of the set of vertices V of S.

2.1 Algorithm outline

Given a query point $p \in \mathbb{R}^3$, let $d_v^*(p)$ be the smallest distance from p to vertices in V, $d_e^*(p)$ the smallest distance from p to the intersections between E and fans from $\mathbb{C}_p$, and $d_f^*(p)$ the smallest distance from p to the intersections between T and the set of rays from $\mathbb{C}_p$; p is sometimes omitted from the notations when its choice is clear. The following lemma forms the basis of our approach.

Lemma 2.1 *Let $\mathbb{C}_p$ be a θ-simplicial cone cover with apex p, and $0 < \theta \leq \pi/3$. Then $\min\{d_v^*, d_e^*, d_f^*\}$ is a 2θ-approximation for $\delta(p)$; i.e, $\delta(p) \leq \min\{d_v^*, d_e^*, d_f^*\} \leq (1 + 2\theta)\delta(p)$.*



Proof: The left inequality is straightforward. We now show the right inequality. Let $q = \mathrm{N}(p)$. The claim is obviously true if $q \in V$, in which case we have that $\delta(p) = d_v^*$. Otherwise, q lies in the interior of either some edge $e \in E$ or some face $f \in T$. Assume first that $q \in e$ and let $C \in \mathbb{C}_p$ be the cone from the simplicial cone cover that contains q. Note that as q is not an endpoint of e and q is the nearest neighbor of p in e, it is necessary that segment pq is perpendicular to edge e (see right figure where C is bounded by the three arrowed rays, and $e = ab$). On the other hand, either e is completely contained in the cone C or it intersects at least one bounding fan of C. Consider the endpoint(s) of e contained inside C and the intersection point(s) of e with fans of C — let q' be the one closer to q among them. Since the angular diameter of C is at most θ, $\angle q'pq \leq \theta/2$. For example, in figure on the right, we assume that $b \in C$, qa intersects one fan of C at c, and $||qb|| \leq ||qc||$; so $q' = b$ and $\angle bpq \leq \theta/2$. It then follows that:

$$\mathrm{d}(p, q') \;\leq\; \frac{\mathrm{d}(p,q)}{\cos\theta/2} \;\leq\; \frac{\mathrm{d}(p,q)}{1 - \sin\theta/2} \;\leq\; (1 + 2\sin\frac{\theta}{2})\mathrm{d}(p,q) \;\leq\; (1+\theta)\mathrm{d}(p,q), \qquad (1)$$

where the last inequality holds whenever $0 < \theta \leq \pi/3$. This implies that:

$$\min\{\delta_v^*, \delta_e^*\} \;\leq\; \mathrm{d}(p, q') \;\leq\; (1+\theta)\mathrm{d}(p,q) \;=\; (1+\theta)\delta(p).$$

If q is in the interior of a face f, then a similar argument shows that $d_f^* \leq (1+2\theta)\mathrm{d}(p,q) = (1+2\theta)\delta(p)$. In particular, we now consider the set of vertices of f that are contained inside C, the set of intersections between edges of f with fans of C, as well as the set of intersections between f and rays of C, and let q' be the one closest to q among them. The lemma then follows. ■

Corollary 2.2 *Let d_v, d_e and d_f be κ-approximations of d_v^*, d_e^* and d_f^*, respectively. Then*

$$\delta(p) \;\leq\; \min\{d_v, d_e, d_f\} \;\leq\; (1+2\theta)(1+\kappa)\;\delta(p).$$

Hence in the high level, our algorithm simply computes a κ-approximation d_x for each d_x^*'s ($x =$ 'v', 'e', or 'f'), and returns the minimum of them. In particular, we will choose $\kappa = \theta$ and $\theta = \varepsilon/5$. By the above corollary, the approximation factor then is $(1+2\theta)(1+\kappa) \leq (1+5\theta) = (1+\varepsilon)$.

3 Approximation Algorithm

In this section, we describe how to θ-approximate $d_x^*(p)$ efficiently for any query point $p \in \mathbb{R}^3$, where $x =$ 'v', 'e', or 'f'. This in turns leads to an ε-approximation for $\delta_S(p)$.

3.1 Approximating d_v^* and d_f^*

The computation of d_v, a θ-approximation of d_v^*, is straightforward by using approximate Voronoi diagrams [6]. Indeed, after preprocessing V into a data structure of size $O(n/\theta^2)$ in $O(\frac{n}{\theta^2}\log(n/\theta))$ time, a θ-approximate nearest neighbor of p in V can be computed in $O(\log(n/\theta))$ time.

For d_f^*, we can compute it exactly by performing $O(1/\theta^2)$ ray shooting queries, one for each direction from Γ_r. Assume that all triangles in T are oriented consistently with the underlying convex polyhedron $\mathcal{H}$. For any direction $u \in \Gamma_r$, we partition T into two sets T_u and $T_b = T \setminus T_u$ such that T_u consists of triangles whose normals form an acute angle with direction u. Now project T_u to the plane Π perpendicular to u and construct a planar point location data structure of size $O(|T_u|)$ for its projection. For any $p \in \mathbb{R}^3$, a ray-shooting query for ray $\gamma_p(u)$ in T_u is simply a point location query of the projection of p in Π, and can be performed in $O(\log|T_u|)$ time. There are altogether $O(1/\theta^2)$ ray directions in Γ_r, and we build a point location data structure of T_u (as well as of T_b) for each of them. Hence after preprocessing T into a data structure of size $O(n/\theta^2)$ in $O(n\log n/\theta^2)$ time, $d_f^*(p)$ can be computed by performing $O(1/\theta^2)$ number of ray shooting queries in $\mathrm{O}(\log n/\theta^2)$ total time for any query point p.

3.2 Approximating d_e^*

The next goal is to build a data structure so that we can approximate $d_e^*(p)$ efficiently for any query point $p \in \mathbb{R}^3$. For any fan $F_p(u)$ from $\mathbb{C}_p$, let $\rho(p,u)$ denote the closest distance from p to the intersection of $F_p(u)$ and E. Then we have that $d_e^*(p) = \min_{u\in\Gamma_f}\rho(p,u)$. Obviously, a κ-approximation of $\rho(p,u)$ for every $u \in \Gamma_f$ results in a κ-approximation for δ_e^*. In what follows, we describe how to build a data structure for a particular fan direction $u \in \Gamma_f$, which can approximate $\rho(p,u)$ efficiently for any query point p. To compute δ_e^*, we construct $O(1/\theta^2)$ such data structures, one for each fan from $\mathbb{C}$.

3.2.1 Overall framework

Given a fan $F_p(u)$ from $\mathbb{C}_p$, let $R(\mathbf{w}, F_p(u))$ denote the specific rectangle as shown in Figure 1; $F_p(u)$ is sometimes omitted when is choice is clear. We call such rectangles *fan-rectangles* and u is referred to as its ***axis***. Note that as $\mathbf{w}$ increases, the two upper vertices of $R(\mathbf{w}, F_p(u))$ progress along the bounding rays of $F_p(u)$.

The *fan-rectangle emptiness problem*, RecEmpty(R, E), asks whether a query fan-rectangle R intersects any edge from E or not. In particular, RecEmpty(R, E) returns *-1* if there is no intersection, RecEmpty(R, E) returns *0* if the only intersection point lies on the boundary of R, and RecEmpty(R, E) returns *1* otherwise. Note that the returned value of RecEmpty$(R(\mathbf{w}), E)$ is monotone in the sense that as $\mathbf{w}$ increases, RecEmpty$(R(\mathbf{w}), E)$ changes from negative to positive, and it can be equal to *0* only once.

Let $\mathbf{w}^*(p, u)$ be the smallest $\mathbf{w}$ such that RecEmpty$(R(\mathbf{w}), E) \geq 0$, i.e, RecEmpty$(R(\mathbf{w}^*, F_p(u)), E) = 0$, and $q^*(p, u)$ the intersection point between $R(\mathbf{w}^*)$ and E at this moment. We have the following result.

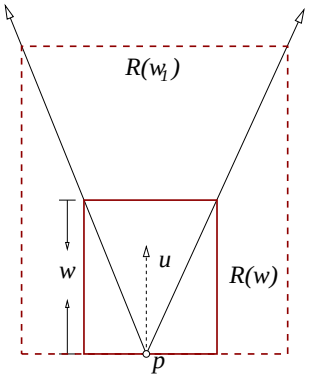
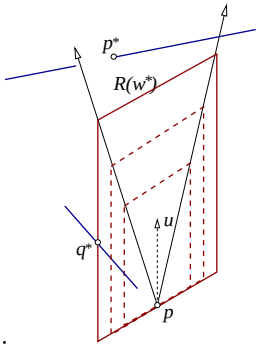


Figure 1: Fan rectangle

Lemma 3.1 *Given $0 < \theta \leq \pi/3$, for any fan $F_p(u)$, we have that* $\mathrm{d}(p, q^*(p, u)) \leq (1 + \theta)\rho(p, u)$.

Proof: Assume without loss of generality that u is the positive z-direction, p is at the origin, and the fan $F_p(u)$ lies in the yz-plane. Suppose that $p^* \in F_p(u)$ is the intersection point between E and $F_p(u)$ that is closest to p (i.e, $\rho(p, u) = \mathrm{d}(p, p^*)$), and $R(\lambda)$ is the fan-rectangle that touches p^*. See figure on the right. Obviously, $\mathbf{w}^*(p, u) \leq \lambda \leq \mathrm{d}(p, w)$. On the other hand, easy to see that $\mathrm{d}(p, q^*(p, u)) \leq \mathbf{w}^*(p, u)/\cos\theta/2$, implying that $\mathrm{d}(p, q^*(p, u)) \leq (1 + \theta)\mathbf{w}^*(p, u)$ by the same argument as in Eq. (1) if $0 < \theta \leq \pi/3$. The lemma then follows. ■



We remark that the above lemma provides only an upper bound for $\mathrm{d}(p, q^*(p, u))$; $\mathrm{d}(p, q^*(p, u))$ can be much smaller than $\rho(p, u)$. Nevertheless, it leads to the following corollary, which, as can be easily verified, still guarantees the correctness of Corollary 2.2.

Corollary 3.2 *Let $d_e(p) = \min_{u \in \Gamma_f} \mathrm{d}(p, q^*(p, u))$. Then $\delta(p) \leq d_e(p) \leq (1 + \theta)d_e^*(p)$.*

Hence below, we focus on computing $\mathrm{d}(p, q^*(p, u))$ for a query point p. In particular, for a fixed fan direction $u \in \Gamma_f$, first, we present an algorithm LowestIntersect(p, u, E) which computes $\mathbf{w}^*(p, u)$ and $q^*(p, u)$ for a query fan $F_p(u)$ using procedure RecEmpty(R, E) as subroutines. We then describe how to implement procedure RecEmpty(R, E) to answer fan-rectangle emptiness queries.

3.3 Algorithm LowestIntersect()

The goal here is to compute $d(p, q^*(p, u))$ for a query fan $F_p(u)$, where $q^*(p, u)$ is the first intersection point between E and some fan-rectangle $R(\mathbf{w}, F_p(u))$ as $\mathbf{w}$ increases. Similar to the framework used to answer ray shooting queries [3], we use the parametric search technique together with procedure RecEmpty$(R(\mathbf{w}, F_p(u)), E)$ to identify $q^*(p, u)$. In particular, given a fan $F_p(u)$, and a sequential algorithm RecEmpty() that runs in A_s time, we will simulate the execution of

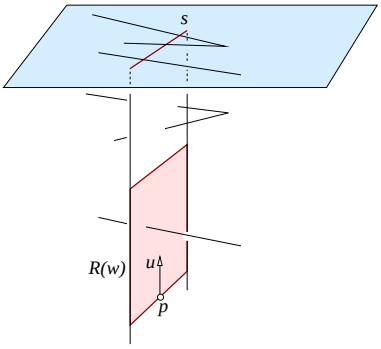
RecEmpty$(R(\mathbf{w}^*, F_p(u)), E)$ where $\mathbf{w}^* = \mathbf{w}^*(p, u)$ is the height of the first fan-rectangle intersecting E. In other words, we run the algorithm generically without knowing the value of $\mathbf{w}^*$. Assume that the flow of execution of RecEmpty$(R(\mathbf{w}), E)$ depends on comparisons, each of which involves testing the sign of a constant-degree polynomial in $\mathbf{w}$ whose coefficients are independent of $\mathbf{w}^*$. Throughout the simulation of RecEmpty$(R(\mathbf{w}^*), E)$, we maintain an interval I which is either a singleton or an open interval containing $\mathbf{w}^*$; I is initialized to be $(-\infty, +\infty)$. During the execution of algorithm RecEmpty$(R(\mathbf{w}^*), E)$, there are times when we need to make decisions with respect to the value of $\mathbf{w}^*$. At such moments, we compute the few roots $r_1, \ldots, r_m$ of the polynomial associated with this comparison (m is considered to be a constant). We then run $\log m$ number of queries of RecEmpty$(R(r_i), E)$ to locate the interval (r_j, r_{j+1}) that contains $\mathbf{w}^*$, and refine I by intersecting I with (r_j, r_{j+1}). If some RecEmpty$(R(r_i), E)$ returns 0, then we return with $\mathbf{w}^* = r_i$. As the execution proceeds, we obtain a sequence of progressively smaller intervals that contain $\mathbf{w}^*$. Since the answer of RecEmpty$(R(x), E)$ will change sign at $\mathbf{w}^*$, at least one of the polynomials should change its sign at $\mathbf{w}^*$, which implies that eventually $\mathbf{w}^*$ will be the root of some polynomial associated with some comparison resolved by algorithm RecEmpty(R, E) (see [5] for a proof). Hence the procedure will terminate with the correct value of $\mathbf{w}^*$. The running time of the above procedure is obviously A_s^2, as it has A_s steps, and at each step, it evokes $\log m = O(1)$ number of sequential algorithm RecEmpty(R, E).

In general, the time complexity can be further improved if one can design an efficient parallel version of the algorithm. In our case, $A_s = O(\log n)$ for a fixed fan direction $u \in \Gamma_f$ as we will show shortly. Furthermore, our algorithm RecEmpty() involves performing binary searches, for which the best parallel algorithm [19] does not improve this $O(A_s^2) = O(\log^2 n)$ time complexity. Hence algorithm LowestIntersect(p, u, E) computes $\mathbf{w}^*(p, u)$ and $q^*(p, u)$ in $O(\log^2 n)$ time for any $p \in \mathbb{R}^3$.

3.3.1 Algorithm RecEmpty()

In this section, for a fixed fan direction $u \in \Gamma_f$, we show how to build a data structure for the set of input edges E, so that a fan-rectangle emptiness query for fan-rectangle $R(\mathbf{w}, F_p(u))$ can be answered efficiently for any apex $p \in \mathbb{R}^3$ and any positive value $\mathbf{w} \in \mathbb{R}$. Note that all fan-rectangles with axis u are parallel to each other. In particular, p and $\mathbf{w}$ uniquely decide a fan-rectangle. So for simplicity, from now on we use $R_p(\mathbf{w})$ to denote $R(x, F_p(u))$. We also assume without loss of generality that u is in the positive z direction, and $F_p(u)$ is parallel to the yz-plane.

Let Π be the horizontal plane at $z = +\infty$ (so Π is parallel to xy-plane), and $\widehat{X}$ the parallel projection of any object X onto Π in z direction. Clearly, the projection of $R_p(\mathbf{w})$ is simply a segment s parallel to the y-axis in the plane Π. The set of edges from $\widehat{E}$ intersecting s corresponds to the set of edges from E intersecting $L_p(w)$, which is the *slab* bounded by the two lines passing through the two vertical sides of $R_p(\mathbf{w})$. See figure on the right for an illustration.



Recall that $\mathcal{H}$ is a convex polyhedron of complexity $O(n)$ that contains S as its subset. Let $\mathcal{H}_u$ be the upper hull of $\mathcal{H}$ which contains all triangles in $\mathcal{H}$ whose normal forms an acute angle with direction u, and $\mathcal{H}_l = \mathcal{H} \setminus \mathcal{H}_u$ is the lower hull of $\mathcal{H}$. Let E_u and E_l denote edges of E from upper hull $\mathcal{H}_u$ and lower hull $\mathcal{H}_l$, respectively. We shall build data structures for E_u and E_l separately and query $R_p(\mathbf{w})$ in each structure. The processing of E_u and E_l are similar. Hence below, we only describe the case for E_u.

To check whether $R_p(\mathbf{w})$ intersects any edge from E_u, we consider the vertical plane $\mathcal{G}p$ that contains the fan $F_p(u)$; $\mathcal{G}p$ is parallel to the yz-plane and is uniquely decided by the x-coordinate

of the point p (see Figure 2). The intersection between $\mathcal{G}p$ and $\mathcal{H}_u$ is a polygonal curve $P \subset \mathcal{G}p$, which is convex as well as monotone in the z-direction due to the convexity of $\mathcal{H}_u$. Since E_u is a subset of $\mathcal{H}_u$, the intersection between E_u and $\mathcal{G}p$, denoted by Q, is a subset of vertices of P. See Figure 2 where empty dots along the polyline P are points from Q.

The emptiness query RecEmpty$(R_p(\mathbf{w}), E_u)$ is now simply an orthogonal range query in the plane $\mathcal{G}p$ w.r.t. the set of points Q. In fact, to further improve the efficiency, we use the following procedure. To test whether $R_p(\mathbf{w})$ is empty of points from Q or not, we compute the lowest point $q^* \in Q$ (if exists) contained inside $\mathcal{U}_p(\mathbf{w})$, the three-sided rectangle formed by the portion of $L_p(\mathbf{w})$ above the bottom edge of $R_p(\mathbf{w})$. We then check whether q^* is below the top edge of $R_p(\mathbf{w})$ or not. If q^* does not exist, then RecEmpty$(R_p(\mathbf{w}), E_u)$ returns $'-1'$. Otherwise, RecEmpty$(R_p(\mathbf{w}), E_u)$ returns $'1', '0'$, or $'-1'$ if q^* is in the interior, on the boundary, or outside of $R_p(\mathbf{w})$, respectively.

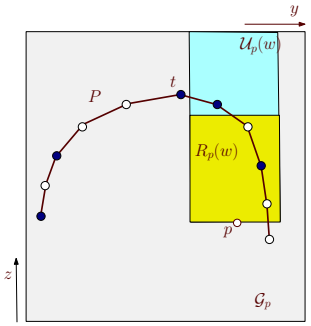


Figure 2: P and Q

Computing q^* if Q is given. First, assume that Q, the set of intersections between E_u and the plane $\mathcal{G}p$, is given, and we wish to compute q^* as described above. To this end, we split Q into $Q = Q_L \bigcup Q_R$ such that points in Q_L (resp. in Q_R) have a smaller (resp. larger) y-coordinates than that of the highest point of P. (See Figure 2, where Q_L and Q_R are empty dots to the left and right of t, respectively.) We compute the lowest point q_L^* and q_R^* from Q_L and Q_R, respectively, that are inside $\mathcal{U}_p(x)$. Clearly, q^* is the lower of the two. Easy to verify that the ordered sequence of points in Q_L and Q_R are monotone both in y and in z directions. Now consider the computation of q_L^* (that of q_R^* is symmetric). Let y_0 be the y-coordinate of the left side of $R_p(\mathbf{w})$, and z_0 the z-coordinate of the bottom edge of $R_p(\mathbf{w})$. The following observation is straightforward.

Observation 3.3 *Let q_1 (resp. q_2) be the point from Q_L with smallest y-coordinate (resp. z-coordinates) that is greater than y_0 (resp. z_0). Then q_L^* is the higher of the two.*

Both q_1 and q_2 can be easily identified by a binary search in the ordered sequence of points in Q_L in $O(\log |Q_L|)$ time. In fact, one binary search suffice as the order of points in Q_L is the same when sorted by either y or z coordinates. Hence q_L^* (and similarly q_R^*) can be computed in $O(\log n)$ time. In other words, the fan-rectangle emptiness query can be answer in $O(\log n)$ time by performing constant number of binary searches if Q is known, for any fan-rectangle with axis u.

Maintaining Q for any p. Note that the vertical plane $\mathcal{G}p$ and its intersection Q with E_u remain the same for all points p with the same x-coordinate. To answer the fan-rectangle emptiness query for any $p \in \mathbb{R}^3$, we wish to maintain a sequence of $Q_L(\mathbf{x})$'s (and $Q_R(\mathbf{x})$'s) as we sweep the plane $\mathcal{G}(\mathbf{x})$ from $\mathbf{x} = -\infty$ to $+\infty$, where $\mathcal{G}(\mathbf{x})$ is the plane $x = \mathbf{x}$. During this sweeping process, as long as $\mathcal{G}(\mathbf{x})$ does not pass through any vertex of E_u, $\mathcal{G}(\mathbf{x})$ intersects the same set of edges from E in the same order, both in the y and in the z directions. Hence there are altogether $\beta = O(n)$ intervals $I_1, I_2, \ldots I_\beta$, such that for any $i \in [1, \beta]$, $Q(\mathbf{x})$ has the same topology for any $\mathbf{x} \in I_i$. This implies that we can use one binary search tree to maintain $Q(\mathbf{x})$'s for the entire interval I_i. Furthermore, let d_i be the complexity of the changes in the ordering of $Q(\mathbf{x})$ between interval I_i and I_{i+1}. Easy to see that $\sum_{i=1}^{\beta-1} d_i = O(n)$. To this end, one can use the standard persistence data structure to maintain all $Q_L(\mathbf{x})$'s in $O(n)$ space, such that a binary search query within any $Q_L(\mathbf{x})$ can still be performed in $O(\log n)$ time [10]. This persistence data structure can be built in $O(n \log n)$ time. The case for $Q_R(\mathbf{x})$'s is symmetric. Hence for a fixed direction u, by building a persistence data

structure of size $O(n)$, one can compute q^*, thus answering RecEmpty$(R_p(\mathbf{w}), E)$ for any query fan-rectangle $R_p(\mathbf{w}) = R(\mathbf{w}, F_p(u))$ in $O(n \log n)$ time for any $p \in \mathbb{R}^3$.

3.3.2 Putting everything together

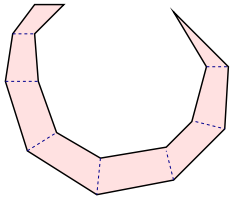
In summary, we pre-process E_u and E_l into $O(1/\theta^2)$ number of persistent binary search data structures, one for each fan direction u from Γ_f. These data structures can be built in $O(n \log n/\theta^2)$ time, and the total space complexity is $O(n/\theta^2)$. Given a query fan-rectangle $R(x, F_p(u))$ in a particular direction $u \in \Gamma_f$, the emptiness query can be answered in $O(\log n)$ time, thus the algorithm LowestIntersect(p, u, E) computes $d(p, q^*(p, u))$ in $O(\log^2 n)$ time by parametric search technique. It then follows from Corollary 3.2 that given any query point $p \in \mathbb{R}^3$, a θ-approximation $d_e(p)$ for $\delta_e^*(p)$ can be computed in $O(\log^2 n/\theta^2)$ time.

Finally, we set $\theta = \varepsilon/5$. Given a query point p, we compute $\delta_f^*(p)$ exactly, and θ-approximate $\delta_v^*(p)$ and $\delta_e^*(p)$ as introduced above. By Corollary 2.2, the minimum of the three values is indeed an ε-approximation for $\delta_S(p)$.

Theorem 3.4 *Given a convex cover S in $\mathbb{R}^3$ of size n, one can preprocess it into a data structure of size $O(n/\varepsilon^2)$ in $O(n \log n/\varepsilon^2)$ time, so that given any query point $p \in \mathbb{R}^3$, an ε-approximation of its shortest distance to S can be computed in time $O(\log^2 n/\varepsilon^2)$.*

4 Conclusion

In this paper, we have presented an efficient algorithm to approximate the shortest distance from a query point p to a convex cover $S \subset \mathbb{R}^3$. The proposed algorithm is mostly practical for implementation. The only impractical part is the parametric search technique. However, one can simply replace the parametric search by a binary search. Although this will change the query time from $O(\log^2 n)$ to $O(\log n \log_{1+\varepsilon} \Delta)$, where Δ is the ratio between the diameter of S and $\delta_S(p)$, we expect Δ to be small in practice.



The performance of our proposed algorithm is based on the convexity of the input triangles. It will be interesting to see whether or not similar ideas can be extended to general polyhedron by decomposing it to several *layers* of convex covers. We remark that this type of decomposition may be interesting by itself. For example, current convex decomposition approaches decompose the input polyhedron into a set of disjoint convex bodies, which may result in a large number of components. It is possible to decompose the input shape into a much smaller number convex covers. See figure on the right for an example in the plane, where the standard decomposition scheme decomposes the shaded polygon into regions separated by dashed segment, while the polygon can also be decomposed into two convex polylines (the outer convex and the inner concave portions, respectively). We leave the problem of understanding such convex cover decompositions, as well as developing efficient algorithms for nearest neighbor problem among general polyhedra as directions for future research.

Acknowledgement. The author would like to thank Pankaj K. Agarwal and Sariel Har-Peled for helpful discussions.

References

[1] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme elevation on a 2-manifold. *Discrete and Computational Geometry*, pages 553–572, 2006.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In Chazelle, Goodman, and Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society Press, 1999.

[3] P. K. Agarwal and J. Matoušěk. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[4] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. COmput.*, 22:794–806, 1993.

[5] P. K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11(2):185–195, 1994.

[6] S. Arya, T. Malamatos, and D. M. Mount. Space-efficient approximate voronoi diagrams. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 721–730, New York, NY, USA, 2002. ACM Press.

[7] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[8] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersections. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 154–165, 1982.

[9] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(3):381–392, 1985.

[10] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[11] P. Indyk. Nearest neighbors in high-dimensional spaces. In Goodman and O'Rouke, editors, *Handbook of Discrete and Computational Geometry*, pages 877–892. CRC Press, 2nd edition, 2004.

[12] V. Koltun and M. Sharir. Polyhedral voronoi diagrams of polyhedra in three dimensions. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 227–236, 2002.

[13] J.-M. Lien. *Approximate Convex Decomposition and Its Applications.* PhD thesis, Dept. of Comput. Sci., Texas A&M University, 2006.

[14] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9:615–627, 1980.

[15] S. Meiser. Point location in arrangements of hyperplanes. *Inf. Comput.*, 106(2):286–303, 1993.

[16] V. Natarajan, Y. Wang, P.-T. Bremer, V. Pascucci, and B. Hamann. Segmenting molecular surfaces. *Computer Aided Geometric Design*, pages 495–509, 2006.

[17] M. Pellegrini. Ray shooting and lines in space. In Goodman and O'Rouke, editors, *Handbook of discrete and computational geometry*, pages 839–856. CRC Press, Inc., 2nd edition, 2005.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[19] R. Tamassia and J. S. Vitter. Parallel transitive closure and point location in planar structures. *SIAM J. Comput.*, 20(4):708–725, 1991.

[20] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.