Close Euclidean Shortest Path Crossing an Ordered 3D Skew Segment Sequence

Nguyen Tran(B) and Michael J. Dinneen

School of Computer Science, The University of Auckland, Auckland, New Zealand
ntra770@aucklanduni.ac.nz, mjd@cs.auckland.ac.nz

Abstract. Given $k$ skew segments in an ordered sequence $E$ and two points $s$ and $t$ in a three-dimensional environment, for any $\epsilon \in (0, 1)$, we study a classical geometric problem of finding a $(1 + \epsilon)$-approximation Euclidean shortest path between $s$ and $t$, crossing the segments in $E$ in order. Let $L$ be the maximum Euclidean length of the segments in $E$ and $h$ be the minimum distance between two consecutive segments in $E$. The running time of our algorithm is $O(k^3 \log(\frac{kh}{\epsilon^2}))$. Currently, the running time of finding the exact shortest path for this problem is exponential. Thus, most practical algorithms of this problem are approximations. Among these practical algorithms, placing discrete points, named Steiner points, on every segment in $E$, then constructing a graph to find an approximate path between $s$ and $t$, is most widely used in practice. However, using Steiner points will cause the running time of this approach to always depend on a polynomial function of the term $\frac{1}{\epsilon}$, which is not a close optimal solution. Differently, in this paper, we solve the problem directly in a continuous environment, without using Steiner points, in terms of the running time depending on a logarithmic function of the term $\frac{1}{\epsilon}$, which we call a close optimal solution.

Keywords: Euclidean shortest path · 3D path planning · Skew segment sequence

1 Introduction

Imagine that we are given two points $s$ and $t$ in a complex 3D space such that going straight between $s$ and $t$ is not allowed. For example, the space between $s$ and $t$ is intercepted by different screens (see Fig. 1). However, there is an ordered sequence of slits (on the screens, for example) where the path between $s$ and $t$ can only go through these slits. The problem in this paper is that how we can find the Euclidean shortest path between $s$ and $t$, going through the slits in such an environment. Let $E = (e_1, \ldots, e_k)$ be an ordered sequence of slits between $s$ and $t$, where $k \geq 1$. If the width of each slit $e_i$ in $E$ is extremely small, or negligible, $e_i$ can be considered as a segment. In practice, this problem is important in a wide range of applications, such as in computer-assisted surgery, military, optics, manufacturing, and game industry. We formally define the problem as follows.
Problem Statement: Euclidean Shortest Path crossing a 3D skew Segment Sequence (ESP-3D-3S). Let $E = (e_1, \ldots, e_k)$ be a sequence of $k \geq 1$ segments in 3D, where for every $i \in \{1, \ldots, k-1\}$, the lines through two consecutive segments $e_i$ and $e_{i+1}$ in $E$ are skew lines. This means that the lines through $e_i$ and $e_{i+1}$ are not parallel and do not intersect with each other (see Fig. 2a). Given $E$ and two points $s$ and $t$ in 3D, the ESP-3D-3S problem asks for the Euclidean shortest path $P^*(s, t) = (s = r_0, r_1, \ldots, r_l, r_{l+1} = t)$ such that $\sum_{i=0}^{l} d(r_i, r_{i+1})$ is minimum, where for every $i \in \{1, \ldots, k\}$, $r_i$, called a crossing point, is a point on the segment $e_i$ in $E$, and $d(r_i, r_{i+1})$ is the Euclidean distance between $r_i$ and $r_{i+1}$.

The main difficulty of the ESP-3D-3S problem is that because two consecutive segments $e_i$ and $e_{i+1}$ are on two skew lines, the planar unfolding fails. This means that there exists no common plane containing both $e_i$ and $e_{i+1}$.

Related Work of the ESP-3D-3S Problem. Let $O$ be a set of disjoint polyhedral obstacles, and $s$ and $t$ be two points in 3D. The problem of finding the shortest path between $s$ and $t$ and avoiding intersecting the interior of any obstacle in $O$ is classical and has been studied for decades. We call this problem ESP-3D. More specifically, the problem ESP-3D asks for a Euclidean shortest path $P^*(s, t) = (s = r_0, r_1, \ldots, r_l, r_{l+1} = t)$, where $l \geq 0$, and for every $i \in \{1, \ldots, l\}$, $r_i$ is a point on an edge of an obstacle in $O$ such that the segment between $r_i$ and $r_{i+1}$ does not intersect the interior of any obstacle, and $\sum_{i=0}^{l} d(r_i, r_{i+1})$ is minimum. It is proven that, to be an Euclidean shortest path, every crossing point $r_i$ of $P^*(s, t)$, $i \in \{1, \ldots, l\}$, need to be on an edge of a polyhedral obstacle in $O$. Thus, the difficulty of the ESP-3D problem is raised by the following two questions: (1) which ordered sequence of edges (segments), namely $E$, of the obstacles in $O$ does $P^*(s, t)$ cross, and (2) when $E$ is determined, how can we find $P^*(s, t)$ crossing $E$? The ESP-3D problem is well-known to be NP-hard by Canny and Reif [1]. However, the reason that makes the ESP-3D problem NP-hard is due to question (1). This means that finding an ordered sequence of
segments $E$ in 3D that $P^*(s,t)$ needs to cross is NP-hard. Thus, in the case that $E$ is given, how can we solve question (2)? Question (2) of the ESP-3D problem is exactly the ESP-3D-3S problem that we focus on in this paper.

For question (2), given $E$, the optimization criterion that makes $P^*(s,t)$ the Euclidean shortest path between $s$ and $t$ crossing $E$ is that when $P^*(s,t)$ crosses a segment $e_i$ in $E$, it must enter and leave $e_i$ at the same angle (see Known Fact 2). Due to this, we can set up an algebraic system of equations to find the crossing points for $P^*(s,t)$. However, this method will result in a system of equations of degree four, which leads to an exponential running time [2]. Because of this algebraic difficulty, most existing solutions for the ESP-3D-3S problem are approximations. We briefly present some main approximation approaches for the problem as follows.

One solution to solve the ESP-3D-3S problem is initially taking any point on every segment $e_i$ in $E$ to initialize the first path between $s$ and $t$ crossing $E$. Then, an iterative scheme of sliding these crossing points is performed to shorten the path, such as the work by Bajaj and Moh [3] or Le et al. [4]. However, the problem of this approach is that we do not know how long the iterative procedure will converge, or how close to the optimal path a returned solution can be [5]. Another work by Polishchuk and Mitchell [6] uses a second order cone program (SOCP) to solve the ESP-3D-3S problem. However, as they stated in the paper, using SOCP to solve the ESP-3D-3S problem is considered as a “black box”. This is also similar to the iterative approach mentioned above, in which we cannot predict or theoretically measure the result of the computed path.

Conversely, for the approximation methods that we can theoretically measure the result of the computed path, with an $\epsilon \in (0,1]$, let $P(s,t)$ be a $(1+\epsilon)$-approximation path between $s$ and $t$ crossing $E$, where the length of $P(s,t)$ is at most $(1+\epsilon)$ times of the length of $P^*(s,t)$. To find $P(s,t)$ in a polynomial running time of $k$ and $\epsilon$, we can basically use one of the existing approximation methods of the ESP-3D problem. That is, every segment in $E$ is discretized by placing points, named Steiner points across the length of the segment. Then, these Steiner points will be interconnected together to create a weighted graph before a shortest path graph algorithm (e.g. Dijkstra) is used to find an approximate shortest path among the Steiner points. For example, one typical result of this group is from the work by Aleksandrov et al. [7], which can compute an $(1+\epsilon)$-approximation $P(s,t)$ in $O(\frac{n}{\epsilon^3} \log \frac{1}{\epsilon} \log n)$ time, where $n$ is the number of vertices. For a comprehensive survey of different methods using Steiner points, we refer the interested readers to [8,9]. The drawback of this approach, however, is that the dependency on $\frac{1}{\epsilon}$ is always polynomial, which is not logarithmic [8]. This means that, when a close to optimal path is required, with $\epsilon$ being extremely small, then $\frac{1}{\epsilon}$ becomes extremely large. Thus, the algorithms using Steiner points can cause an impractical running time in comparison with the algorithms whose dependency on $\frac{1}{\epsilon}$ is logarithmic.

In terms of the dependency on $\frac{1}{\epsilon}$ being logarithmic, we notice the method by Burago et al. [10] whose dependency on $\frac{1}{\epsilon}$ is doubly logarithmic. Although they find an approximate shortest path $P(s,t)$ crossing a sequence of skew lines, not
a sequence of skew segments, the problem does not change much. Let \( L(E) = (l_1, \ldots, l_k) \) be the sequence of \( k \) lines with respect to \( E \), where every line \( l_i \), \( i \in \{1, \ldots, k\} \), is the line containing the corresponding segment \( e_i \) in \( E \). Let \( P'(s, t) = (s, \sigma'_1, \ldots, \sigma'_k, t) \) be a \((1 + \epsilon)\)-approximation shortest path between \( s \) and \( t \) crossing \( L(E) \), where every crossing point \( \sigma'_i \) is a point on \( l(e_i) \). Let \( P(s, t) = (s, \sigma_1, \ldots, \sigma_k, t) \) be a \((1 + \epsilon)\)-approximation shortest path between \( s \) and \( t \) crossing \( E \), where every crossing point \( \sigma_i \) is a point on \( e_i \). As mentioned in \[3\], \( P(s, t) \) can be calculated from \( P'(s, t) \) as follows. For every \( i \in \{1, \ldots, k\} \), if \( \sigma'_i \) is on the segment \( e_i \), \( \sigma_i \) is equal to \( \sigma'_i \). Otherwise, if \( \sigma'_i \) is outside the segment \( e_i \), \( \sigma_i \) will be one of the two endpoints of \( e_i \). Let \( \tilde{d} \) and \( \tilde{\alpha} \) be the minimal distance and minimal sine of the angles between two consecutive lines in \( L(E) \), respectively. The method of \[10\] can compute \( P'(s, t) \) crossing \( L(E) \) in time \( O((\frac{Rk}{d\alpha})^{16} + k^2 \log \log \frac{1}{\epsilon}) \), where \( R \) is defined as the radius of a ball in which the initial approximation can be placed. To our knowledge, this is the only method of computing a \((1 + \epsilon)\)-approximation shortest path crossing \( E \) whose dependency on \( \frac{1}{\epsilon} \) is doubly logarithmic.

**Our Work Summary.** To this end, we call a \((1 + \epsilon)\)-approximation algorithm as *close optimal* only if the dependency on \( \frac{1}{\epsilon} \) of the running time of the algorithm is logarithmic.

1. Our solution for the ESP-3D-3S problem in this paper is a close optimal solution, finding a \((1 + \epsilon)\)-approximation shortest path in \( O(k^3 \log^2 (\frac{kl}{\epsilon})) \) time.
2. In comparison, while the running times of the approximation algorithms using Steiner points currently depend on \( \frac{1}{\epsilon} \) polynomially, our method depends on \( \frac{1}{\epsilon} \) logarithmically. This means that, when a close optimal path is required, where \( \epsilon \) needs to be extremely small, our algorithm can run much faster than the algorithms using Steiner points.
3. As presented above, to our knowledge, currently, only the work by Burago et al. \[10\] is a close optimal algorithm for the ESP-3D-3S problem, with the running time being \( O((\frac{Rk}{d\alpha})^{16} + k^2 \log \log \frac{1}{\epsilon}) \). The dependency of this algorithm on \( \frac{1}{\epsilon} \) is doubly logarithmic. However, its dependency on \( k \) is up to \( k^{16} \) while the dependency on \( k \) of our algorithm is only \( k^3 \log k \).

The rest of the paper is organized as follows. Section 2 contains some definitions and preliminaries. Our proposed algorithm for solving the ESP-3D-3S problem is presented in Sect. 3. Finally, Section 4 concludes the paper.

## 2 Preliminaries

Let \( P^*(s, t) = (s = r_0, r_1, \ldots, r_k, r_{k+1} = t) \) be the exact Euclidean shortest path between \( s \) and \( t \) crossing a sequence \( E \) of \( k \) skew segments in order, where for every \( i \in \{1, \ldots, k\} \), \( r_i \) is a point on \( e_i \) in \( E \). With any two points \( u \) and \( v \), we denote \((u, v)\) as the segment between \( u \) and \( v \), and \( \overrightarrow{uv} \) as the vector from \( u \) to \( v \).

We first consider the optimization criterion that makes \( P^*(s, t) \) the Euclidean shortest path. For every segment \( e_i \) in \( E \), \( i \in \{1, \ldots, k\} \), let \( l(e_i) \) be the line...
The Euclidean shortest path $P^*(s, t) = (s, r_1, \ldots, r_k, t)$ crossing $k$ skew segments in $E = (e_1, \ldots, e_k)$ in order.

Every cone $C^r_i$, $i \in \{1, \ldots, k\}$ is created at $r_i$, with the axis ray being $r_i h_{i+1}^i$, and the acute cone angle being $\gamma_i$. Note that, all these cones are infinite.

Fig. 2. The Euclidean shortest path $P^*(s, t)$ and its optimization criterion.

containing $e_i$ (see Fig. 2a). Let $h_{i-1}^i$ and $h_{i+1}^i$ be two points on $l(e_i)$ such that the segments $(r_{i-1}, h_{i-1}^i)$ and $(r_{i+1}, h_{i+1}^i)$ are perpendicular to $l(e_i)$. Let $\theta_i$ and $\gamma_i$ be two acute angles created by $(r_i, r_{i-1})$ and $(r_i, h_{i-1}^i)$, and by $(r_i, r_{i+1})$ and $(r_i, h_{i+1}^i)$, respectively. The two known facts below are due to [2].

**Known Fact 1.** $P^*(s, t)$ is unique.

**Known Fact 2.** The path $P^*(s, t)$ is the exact Euclidean shortest path crossing $k$ segments in $E$ in order if and only if at every segment $e_i \in E$, $i \in \{1, \ldots, k\}$, for which $r_i$ is not an endpoint of $e_i$, $h_{i-1}^i$ and $h_{i+1}^i$ are on the two different half lines of $l(e_i)$ induced by $r_i$ and $\theta_i = \gamma_i$.

Known Fact 2 suggests that, for every $i \in \{1, \ldots, k\}$, if an infinite, single-sided, right circular cone, named $C^r_i$, is created at $r_i$ with the axis ray being the vector $r_i h_{i+1}^i$, and the acute cone angle being $\gamma_i$, the point $r_{i+1}$ will be the intersection point between the surface of the cone $C^r_i$ and $e_{i+1}$ (see Fig. 2b). We note that, all cones we use in this paper are infinite, single-sided, and right circular. Thus, to this end, to be simple, we call all of them as cones. Furthermore, to this end, when we mention cone, we mean the surface of the cone. That is, if we say a line or a segment intersects a cone, or two cones intersect with each other, we mean the line or segment intersects the surface of the cone, or the two surfaces of the cones intersect with each other. Additionally, a segment $e_i = (p_i, q_i)$ is denoted as being inside (resp. outside) a cone if both endpoints $p_i$ and $q_i$ are inside (resp. outside) the cone, or if $p_i$ or $q_i$ is on the surface of the cone, the remaining endpoint must be inside (resp. outside) the cone.
Definition 1. Cone-Create-Rule.

First, we consider the segment $e_1$ in $E$. Let $a_1$ be a point on $e_1$ (see Fig. 3). Let $h_0^1$ be the point on $e_1$ such that $(s, h_0^1)$ is perpendicular to $l(e_1)$. Let $\theta_1$ be the acute angle created by $(a_1, s)$ and $(a_1, h_0^1)$. Let $C_1^a$ be the cone that is created on $e_1$, at $a_1$, with the axis ray being opposite to the vector $a_1 h_0^1$, and the acute cone angle being $\gamma_1 = \theta_1$. We say that, $C_1^a$ is created by $s$ and $a_1$, based on the Cone-Create-Rule.

Similarly, for every $i \in \{2, \ldots, k\}$, we consider the segment $e_i = (p_i, q_i)$ in $E$. Suppose that $e_i$ intersects $C_{i-1}^a$ at $a_i$. Let $h_{i-1}^i$ be a point on $e_i$, where $(a_{i-1}, h_{i-1}^i)$ is perpendicular to $l(e_i)$. Let $\theta_i$ be the acute angle created by $(a_i, h_{i-1}^i)$ and $(a_i, a_{i-1})$. Let $C_i^a$ be the cone that is created on $e_i$, at $a_i$, with the axis ray being opposite to the vector $a_i h_{i-1}^i$, and the acute cone angle being $\gamma_i = \theta_i$. We say that, $C_i^a$ is created by $s$ and $a_i$, based on the Cone-Create-Rule.

Definition 2. Euclidean-Ray with respect to a cone sequence.

With a point $a_1$ on segment $e_1$, first, applying the Cone-Create-Rule, we can find the cone $C_1^a$ on $e_1$. Suppose that $e_2$ intersects $C_1^a$ at $a_2$. Then, using the Cone-Create-Rule, we can continue the calculations to obtain the path $R_a = (s, a_1, \ldots, a_g)$, with respect to the cone sequence $C_{g} = (C_1^a, \ldots, C_g^a)$, where $1 \leq g \leq k$ (see Fig. 3). This calculation stops at $e_g$ when $g = k$, or if one of the following two conditions holds: (i) the segment $e_{g+1}$ does not intersect the cone $C_g^a$, or (ii) $e_{g+1}$ intersects the cone $C_g^a$ at one of its two endpoints. We call $R_a$ as a Euclidean-Ray with respect to the cone sequence $C_a$, from $s$, crossing $E$ by starting at $a_1$ on $e_1$ to $a_g$ on $e_g$.

From Definition 2, to find $P^*(s, t)$ crossing $E$, the following question remains. How can we find the point $r_1$ on $e_1$ such that, after calculating the Euclidean-Ray $R_r = (s, r_1, \ldots, r_k)$ with respect to $C_r = (C_1^r, \ldots, C_k^r)$, $t$ is on $C_k^r$?

Proposition 1. Let $R_u = (s, u_1, \ldots, u_i)$ be a Euclidean-Ray with respect to $C_u = (C_1^u, \ldots, C_i^u)$, crossing $E$ from $s$, starting at $u_1$ on $e_1$ to $u_i$ on $e_i$, where $i \leq k$. Let $R_v = (s, v_1, \ldots, v_j)$ be another Euclidean-Ray with respect to $C_v = (C_1^v, \ldots, C_j^v)$, crossing $E$ from $s$, starting at $v_1$ on $e_1$ to $v_j$ on $e_j$, where $j \leq k$. If $u_1 \neq v_1$, for every $e_i$, $l \in \{1, \ldots, i\}$ if $i \leq j$, or $l \in \{1, \ldots, j\}$ if $j \leq i$, then two cones $C_l^u$ and $C_l^v$ cannot intersect with each other.

Proof. Without loss of generality, suppose that $i \leq j$. By contradiction, suppose that there exists a segment $e_f$, $1 \leq f \leq i$, that two cones $C_f^u$ of $C_u$ and $C_f^v$ of $C_v$ on $e_f$ intersect with each other (see Fig. 4). Let $O$ be the intersection circle between $C_f^u$ and $C_f^v$. Let $u_0$ and $v_0$ be the intersection points between $R_u$ and $O$ and between $R_v$ and $O$, respectively. Let $P_u(s, v_0) = (s, u_1, \ldots, u_f, v_0)$ and $P_v(s, v_0) = (s, v_1, \ldots, v_f, v_0)$ be two paths from $s$ to $v_0$ following $R_u$ and $R_v$, respectively. Since $v_0$ is also on $C_f^v$, the path $P_u(s, v_0)$ satisfies Known Fact 2. Thus, $P_u(s, v_0)$ is the Euclidean shortest path from $s$ to $v_0$. However, the path $P_v(s, v_0)$ also satisfies Known Fact 2 to make $P_v(s, v_0)$ the Euclidean shortest path from $s$ to $v_0$. Therefore, we have both $P_u(s, v_0)$ and $P_v(s, v_0)$ being two different Euclidean shortest paths from $s$ to $v_0$. This is contrary to Known Fact 1. □
Due to Proposition 1, for any segment $e_i$ in $E$, we now can see how all cones are distributed on $e_i$, as follows. On any segment $e_i = (p_i, q_i)$ in $E$, suppose that a Euclidean-Ray $R_a$, with respect to its cone sequence $C_a$, intersects $e_i$ at $a_i$, and the cone $C_i^a$ of $C_a$ at $a_i$ has the axis ray being $\overline{a_i p_i}$ and the acute cone angle being $\gamma_i$ (see Fig. 5). Then, imaging that, if other cones are created on $e_i$ gradually from $a_i$ to $p_i$ by different Euclidean-Rays from $s$, their acute cone angles must be gradually smaller in comparison with $\gamma_i$. This means that, for example, let $u_i$ and $v_i$ be two points on $(a_i, p_i)$ with $d(a_i, u_i) < d(a_i, v_i)$ and $C_i^u$ and $C_i^v$ be two cones created at $u_i$ and $v_i$ from two different Euclidean-Rays from $s$, respectively. The acute cone angle of $C_i^v$ must be smaller than the acute cone angle of $C_i^u$. It is easy to see that, if the acute cone angle of $C_i^u$ is larger than the acute cone angle of $C_i^v$, $C_i^u$ and $C_i^v$ will intersect with each other, and this is contrary to Proposition 1. Otherwise, if other cones are created on $e_i$ but gradually from $a_i$ to $q_i$, first, their acute cone angles must be gradually larger than $\gamma_i$ to $90^\circ$. From $90^\circ$, the axis rays of the remaining cones will be changed to $\overline{a_i q_i}$, then being gradually smaller.

Let $S$ and $S'$ be two sequences of segments. To this end, we use the notation $S \circ S'$ to denote the sequence of segments obtained from $S$ and $S'$ by appending $S'$ to the end of $S$.

### 3 Euclidean Shortest Path Crossing a Sequence of 3D Skew Segments

We now consider a Euclidean-Ray $R_a = (s, a_1, \ldots, a_g)$ with respect to $C_a = (C_1^a, \ldots, C_g^a)$, $g \leq k$, where for every $i \in \{1, \ldots, g-1\}$, $e_{i+1}$ intersects $C_i^a$ at only one point. We will consider the case that $e_{i+1}$ intersects $C_i^a$ at two points (see
Fig. 4. Two Euclidean rays $R_u = (s, u_1, \ldots, u_i)$, with respect to $C_u = (C_{u_1}^1, \ldots, C_{u_i}^v)$, and $R_v = (s, v_1, \ldots, v_j)$, with respect to $C_v = (C_{v_1}^1, \ldots, C_{v_j}^v)$, have $C_v^f$ and $C_u^f$ intersect with each other on a segment $e_f$. This case cannot happen.

Fig. 5. Different cones are distributed on a segment $e_i$.

Fig. 6) later. For every $e_i = (p_i, q_i)$, $i \in \{1, \ldots, g\}$, we name the two endpoints $p_i$ and $q_i$ of $e_i$ such that $\overrightarrow{a_ip_i}$ is the same direction with the axis ray of $C_{a_i}^a$ (see Fig. 3).

Let $R_r = (s, r_1, \ldots, r_k)$ with respect to $C_r = (C_{r_1}^r, \ldots, C_{r_k}^r)$ be the Euclidean-Ray that hits $t$, where $t$ is on the last cone $C_{r_k}^r$. Given a Euclidean-Ray $R_a = (s, a_1, \ldots, a_g)$ with respect to $C_a = (C_{a_1}^1, \ldots, C_{a_g}^g)$, $g \leq k$, for every $i \in \{1, \ldots, g\}$, $r_i$ on $e_i$ of $R_r$ can be determined on either $(a_i, p_i)$ or $(a_i, q_i)$, as follows. First, we see that if $e_{g+1}$ is inside (resp. outside) $C_g^r$ (see Fig. 3b), then $r_g$ must be on $(a_g, p_g)$ (resp. $(a_g, q_g)$). This observation is correct because if $r_g$ is on $(a_g, q_g)$, the cone $C_{r_g}^r$ at $r_g$ of $R_r$ needs to intersect $e_{g+1}$, then $C_{r_g}^r$ and $C_{a_g}^a$ will intersect with each other, which is contrary to Proposition 1. Now, we know that $r_g$ is on $(a_g, p_g)$. As we constrained above, $e_g$ intersects $C_{g-1}^a$ at only one point $a_g$.

Thus, between $(a_g, p_g)$ and $(a_g, q_g)$, one will be inside and the remaining one will be outside $C_{g-1}^a$. Suppose that $(a_g, p_g)$ is outside $C_{g-1}^a$. Then, we can similarly determine that $r_{g-1}$ on $e_{g-1}$ of $R_r$ must be on $(a_{g-1}, q_{g-1})$, which is outside $C_{g-1}^a$. By this way, if we continue tracing back from $g-1$ to 1, based on $R_a$, we can totally determine on every segment $e_i = (p_i, q_i)$ in $E$, which sub-segment, $(a_i, p_i)$ or $(a_i, q_i)$, of $e_i$ that $R_r$ crosses. Then, all the remaining sub-segments that $R_r$ does not cross, called unnecessary sub-segments, will be trimmed or deleted. We present this idea in the function $Trim-Segments$.

Let $P(s, t)$ be a $(1 + \epsilon)$-approximation shortest path of $P^*(s, t)$. We now present the main idea of the function $Find-Approximate-Path$ to find $P(s, t)$. Let $\delta$ be an extremely small value such that, if the Euclidean distance between
two points, or between a point and a line is less than or equal to \( \delta \), then the two points are considered to be the same, or the point is considered to be on the line. Let \( m_1 \) be the middle point of \( e_1 \). From \( m_1 \), we use the function \textit{Create-Euclidean-Ray} (presented later) to find the Euclidean-Ray \( R_m = (s, m_1, \ldots, m_g) \), \( g \leq k \), with respect to its cone sequence \( C_m = (C^m_1, \ldots, C^m_g) \). The function \textit{Create-Euclidean-Ray} helps find \( R_m \), along with trimming the segments in \( E \) as needed, to guarantee that for every \( i \in \{1, \ldots, g - 1\} \), the segment \( e_{i+1} \), after being trimmed, will intersect the cone \( C_i^m \) at only one point. We will present this trimming process later. After creating \( R_m \), as presented above, we can totally determine on every segment \( e_i \), \( i \in \{1, \ldots, g\} \), which sub-segment, \( (m_i, p_i) \) or \( (m_i, q_i) \), that \( R_r \) crosses. Then, all the remaining unnecessary sub-segments of \( e_1 \) to \( e_g \) that \( R_r \) crosses will be deleted. The function \textit{Find-Approximate-Path} iterates through this process until \( t \) is on \( C_k^m \), or all segments \( e_i = (p_i, q_i) \) in \( E \) are trimmed such that \( d(p_i, q_i) \leq \delta \).

\textbf{Find-Approximate-Path:}

\textbf{Input:} \( E = (e_1, \ldots, e_k) \), \( s \) and \( t \)

\textbf{Output:} An approximate shortest path \( P(s, t) \), from \( s \) to \( t \) crossing \( E \)

1. Initialize: \( P(s, t) = (\ ) \), root = \( s \), \( l = 1 \).
2. Let \( m_l \) be the middle point of \( e_l = (p_l, q_l) \). If \( d(p_l, q_l) \leq \delta \), go to Step 4. Otherwise, go to Step 3.
3. Set \( R_m = (\text{root}, m_l) \) and \( C_m = (\ ) \). Run the function \textit{Create-Euclidean-Ray}\( (E, R_m, C_m) \) to get \( R_m = (\text{root}, m_l, \ldots, m_g) \) and \( C_m = (C^m_1, \ldots, C^m_g) \) (Note that, some segments in \( E \) can be trimmed in the function \textit{Create-Euclidean-Ray}).
   3.1 If \( g = k \) and \( t \) is on \( C_g^m \), return \( P(s, t) \circ R_m \circ (t) \).
   3.2 Otherwise, if \( g = k \) and \( t \) is inside or outside \( C_g^m \), or \( g < k \), run the function \textit{Trim-Segments}\( (E, R_m, C_m, l, g, t) \) to delete all the unnecessary sub-segments of \( e_l \) to \( e_g \) that \( R_r \) does not cross. Go to Step 2.
4. \( P(s, t) = P(s, t) \circ (\text{root}) \).
   4.1 If \( l = k \), return \( P(s, t) = P(s, t) \circ (m_l, t) \).
   4.2 Otherwise, if \( l < k \), root = \( m_l \), \( l = l + 1 \), go to Step 2.

\textbf{Trim-Segments:}

\textbf{Input:} \( E = (e_1, \ldots, e_k) \), \( R_a = (\text{root}, a_1, \ldots, a_g) \), \( C_a = (C^a_1, \ldots, C^a_g) \), \( l, g, t \)

\textbf{Output:} The segments from \( e_l \) to \( e_g \) in \( E \) will be trimmed such that for every segment \( e_i = (p_i, q_i) \), \( i \in \{l, \ldots, g\} \), only the sub-segment \( (a_i, p_i) \) or \( (a_i, q_i) \) that \( R_r \) crosses will be kept.

1. Initialize: \( i = g \). If \( g = k \), \( e_{g+1} = (t, t) \).
2. If \( e_{i+1} \) is outside \( C^a_i \), \( e_i = (a_i, q_i) \), where \( (a_i, q_i) \) is outside \( C^a_i \).
3. Otherwise, if \( e_{i+1} \) is inside \( C^a_i \), \( e_i = (a_i, p_i) \), where \( (a_i, p_i) \) is inside \( C^a_i \).
4. \( i = i - 1 \). If \( i = l - 1 \), return. Otherwise, go to Step 2.

In the function \textit{Find-Approximate-Path}, we use two variables \( l \) and \( \text{root} \) such that when \( l = 1 \), \( \text{root} = s \). Then, when \( l \geq 2 \) and \( d(p_{l-1}, q_{l-1}) \leq \delta \), \( \text{root} \) will be
the middle point of $e_{i-1} = (p_{i-1}, q_{i-1})$. Let $R_a = (\text{root}, a_1, \ldots, a_y)$, $l \leq y < k$, with respect to the cone sequence $C_a = (C_1^a, \ldots, C_y^a)$, be an existing Euclidean-Ray from root, starting at $a_l$ on $e_l$ to $a_y$ on $e_y$. The function $\text{Create-Euclidean-Ray}(E, R_a, C_a)$ will receive $R_a$ and $C_a$, then continue constructing $R_a$, from $a_y$ on $e_y$, to $a_g$ on $e_g$, $g \leq k$, to output $R_a = (\text{root}, a_1, \ldots, a_y, a_g)$, with respect to $C_a = (C_1^a, \ldots, C_y^a, \ldots, C_g^a)$, where $g \leq k$. In the function $\text{Create-Euclidean-Ray}$, we also solve the special case that a segment $e_{i+1}$ in $E$, $i \in \{1, \ldots, g-1\}$, intersects $C_i^a$ at two points, named $a_{i+1}^1$ and $a_{i+1}^2$ (see Fig. 6), as follows.

We see that, from $a_i$, if we create two Euclidean-Rays following both $a_{i+1}^1$ and $a_{i+1}^2$, and continue the calculations by this way for all of the next segments, it will cause the number of the Euclidean-Rays to increase exponentially. However, observe that, there exists a Euclidean-Ray $R_s = (\text{root}, s_1, \ldots, s_i, s_{i+1}, \ldots, s_j)$, $j \leq k$, with respect to $C_s = (C_1^s, \ldots, C_i^s, C_{i+1}^s, \ldots, C_j^s)$, such that $e_{i+1}$ intersects $C_i^s$ at only one point $s_{i+1}$ on $(a_{i+1}^1, a_{i+1}^2)$, or $e_{i+1}$ is a tangent to $C_i^s$ (see Fig. 6). Thus, we first find the Euclidean-Ray $R_s$, then use the function $\text{Trim-Segments}$ to delete all the unnecessary sub-segments in $E$ that $R_r$ does not cross based on $R_s$. As presented above, after this trimming, the segment $e_{i+1} = (p_{i+1}, q_{i+1})$ will remains either $(s_{i+1}, p_{i+1})$ or $(s_{i+1}, q_{i+1})$. Thus, $e_{i+1}$ will intersect $C_i^s$ at only on point, which is either $a_{i+1}^1$ or $a_{i+1}^2$.

We call a part of the Euclidean-Ray $R_s$, from root to $s_{i+1}$, $\text{DR}_s(e_i, e_{i+1}) = (\text{root}, s_1, \ldots, s_i, s_{i+1})$, with its cone sequence $\text{DC}_s(e_i, e_{i+1}) = (C_1^s, \ldots, C_i^s, C_{i+1}^s)$, as Division-Ray between $e_i$ and $e_{i+1}$. The Division-Ray $\text{DR}_s(e_i, e_{i+1})$ between two consecutive segments $e_i$ and $e_{i+1}$ helps divide $e_{i+1} = (p_{i+1}, q_{i+1})$ into two sub-segments $(s_{i+1}, p_{i+1})$ and $(s_{i+1}, q_{i+1})$ such that one sub-segment will be kept and the remaining one will be deleted. We note that, in the function $\text{Create-Euclidean-Ray}$ below, we use a global array $A$ to store all Division-Rays between

\begin{figure}
\centering
\includegraphics[width=\textwidth]{division-ray}
\caption{Illustration of the Division-Ray between $e_i$ and $e_{i+1}$.}
\end{figure}
two consecutive segments in $E$ when they are found. The reason for using this array will be explained later.

**Create-Euclidean-Ray:**

**Input:** $E = (e_1, \ldots, e_k)$, $R_a = (\text{root} = a_{l-1}, a_l, \ldots, a_y)$, $C_a = (C^a_l, \ldots, C^a_y)$

**Output:** $R_a = (\text{root}, a_l, \ldots, a_y, \ldots, a_g)$, $C_a = (C^a_l, \ldots, C^a_y, \ldots, C^a_g)$

1. Initialize: $i = y$, $a_i = a_y$
   1.1 If $C_a = (\ )$, let $C^a_i$ be the cone created by $\text{root}$ and $a_i$, based on the Cone-Create-Rule. $C_a = (C^a_i)$.
   1.2 Otherwise, if $C_a \neq ( \ )$, $C^a_i = C^a_y$.
2. If $i = k$, return $R_a$ and $C_a$.
3. Otherwise, if $i < k$,
   3.1 If $e_{i+1}$ intersects $C^a_i$ at one point $a_{i+1}$, create the cone $C^a_{i+1}$ by $C^a_i$, based on the Cone-Create-Rule. $R_a = R_a \circ (a_{i+1})$, $C_a = C_a \circ (C^a_{i+1})$, $i = i + 1$, go to Step 2.
   3.2 If $e_{i+1}$ does not intersect $C^a_i$, return $R_a$ and $C_a$.
   3.3 If $e_{i+1}$ intersects $C^a_i$ at two points $a^1_{i+1}$ and $a^2_{i+1}$ (see Fig. 6),
      - If the global array $A$ contains the Division-Ray $DR_s(e_i, e_{i+1}) = (\text{root}, s_i, \ldots, s_i, s_{i+1})$, with respect to $DC_s(e_i, e_{i+1})$, between $e_i$ and $e_{i+1}$, get $DR_s(e_i, e_{i+1})$ from $A$.
      Otherwise, if $DR_s(e_i, e_{i+1})$ does not exist in $A$, run the function $\text{Find-Division-Ray}(E', R_a, C_a, a^1_{i+1}, a^2_{i+1})$ to find $DR_s(e_i, e_{i+1})$, where $E' = (e_i, \ldots, e_{i+1})$ is a copy of the segments from $e_l$ to $e_{i+1}$ in $E$. Then, store $DR_s(e_i, e_{i+1})$ into $A$.
      - Set $R_s = DR_s(e_i, e_{i+1})$ and $C_s = DC_s(e_i, e_{i+1})$.
      - Run the function $\text{Create-Euclidean-Ray}(E, R_s, C_s)$ to continue calculating the Euclidean-Ray $R_s$ from $e_{i+1}$ to have
        $R_s = (\text{root}, s_i, \ldots, s_i, s_{i+1}, \ldots, s_j)$, $i + 1 \leq j \leq k$, with respect to $C_s = (C^s_i, \ldots, C^s_j)$.
      - Run the function $\text{Trim-Segments}(E, R_s, C_s, l, j, t)$ (In this function, $e_{i+1} = (p_{i+1}, q_{i+1})$ will be trimmed, which remains either $(s_{i+1}, p_{i+1})$ or $(s_{i+1}, q_{i+1})$).
      - Let $a^t_{i+1}$ be the point that $e_{i+1}$, after being trimmed, intersects $C^a_i$.
        Now, $a^t_{i+1}$ is coincide with either $a^1_{i+1}$ or $a^2_{i+1}$. Create the cone $C^a_{i+1}$ by $C^a_i$ based on the Cone-Create-Rule, at the intersection point $a^t_{i+1}$.
        $R_a = R_a \circ (a^t_{i+1})$, $C_a = C_a \circ (C^a_{i+1})$, $i = i + 1$, go to Step 2.

We use the function $\text{Find-Division-Ray}$ to find the Division-Ray $DR_s(e_i, e_{i+1})$ between $e_i$ and $e_{i+1}$. The idea of this function is also similar to the idea of the function $\text{Find-Approximate-Path}$. First, we trim $e_{i+1}$ to be $(a^1_{i+1}, a^2_{i+1})$. Then, based on the input Euclidean-Ray $R_a$, we can determine which sub-segments from $e_l$ to $e_i$ that $DR_s(e_i, e_{i+1})$ crosses to intersect $(a^1_{i+1}, a^2_{i+1})$, then using the function $\text{Trim-Segments}$ to delete all the unnecessary sub-segments from $e_l$ to $e_i$ that $DR_s(e_i, e_{i+1})$ does not cross. After that, similar to the function $\text{Find-Approximate-Path}$, we create the Euclidean-Ray $R_m$ from $\text{root}$, starting at the middle point $m_l$ of $e_l$ to $e_i$. When $R_m$ can come to $e_i$
at $m_i$ and create the cone $C_i^m$, one of the following three conditions holds: (i) $e_{i+1}$ intersects $C_i^m$ at two points, named $m_{i+1}^1$ and $m_{i+1}^2$, or (ii) $e_{i+1}$ intersects $C_i^m$ at only one point $m_{i+1}$, or (iii) $e_{i+1}$ does not intersect $C_i^m$. For (i), $e_{i+1}$ will only need to keep the sub-segment $(m_{i+1}^1, m_{i+1}^2)$. Then, we use the function Trim-Segments to delete all the unnecessary sub-segments from $e_l$ to $e_i$ that $DR_s(e_i, e_{i+1})$ does not cross, based on $R_m$. For (ii), $R_m$ is the Division-Ray that we need to find. For (iii), since the cone $C_i^s$ at $e_i$ of $DR_s(e_i, e_{i+1})$ must cross $e_i$, then $s_i$ must be on $(m_i, q_i)$, where $(m_i, q_i)$ is outside the cone $C_i^m$. Thus, $e_i$ only needs to keep the sub-segment $(m_i, q_i)$. Then, we use the function Trim-Segments to delete all the unnecessary sub-segments from $e_l$ to $e_{i-1}$, based on $R_m$. The function Find-Division-Ray below iterates through this process until $e_{i+1}$ intersects $C_i^m$ at only one point, or all the segments $e_j = (p_j, q_j)$ from $e_l$ to $e_i$ are trimmed until $d(p_j, q_j) \leq \delta$.

An important note is that trimming segments from $e_l$ to $e_{i+1}$ in the function Find-Division-Ray is just to supply finding $DR_s(e_i, e_{i+1})$, which should not affect the process of finding $R_e$. Thus, in the function Create-Euclidean-Ray, we use a copy of the segments in $E$, named $E'$, to put into the function Find-Division-Ray. This is to notice that $E$ will not be affected when $E'$ is trimmed in the function Find-Division-Ray.

**Find-Division-Ray:**

**Input:** $E = (e_l, \ldots, e_i, e_{i+1})$, $R_a = (\text{root}, a_l, \ldots, a_i)$, $C_a = (C_1^a, \ldots, C_i^a)$, $a_{i+1}^1$, $a_{i+1}^2$ (where $e_{i+1}$ intersects $C_i^a$ at $a_{i+1}^1$ and $a_{i+1}^2$)

**Output:** $DR_s(e_i, e_{i+1}) = (\text{root}, s_1, \ldots, s_i, s_{i+1})$, with respect to $DC_s(e_i, e_{i+1}) = (C_1^s, \ldots, C_i^s, C_{i+1}^s)$ (where $e_{i+1}$ intersects $C_i^s$ at only one point $s_{i+1}$).

1. **Initialize:** $DR_s(e_i, e_{i+1}) = (\ )$, $DC_s(e_i, e_{i+1}) = (\ )$, $e_{i+1} = (a_{i+1}^1, a_{i+1}^2)$, $f = l$, $rootS = root$, run the function Trim-Segments($E$, $R_a$, $C_a$, $l$, $i$, null).
2. Let $m_f$ be the middle point of $e_f = (p_f, q_f)$. If $d(p_f, q_f) \leq \delta$, go to Step 4.
   Otherwise, go to Step 3.
3. Set $R_m = (rootS, m_f)$, $C_m = (\ )$. Run the function Create-Euclidean-Ray($E$-\{e_{i+1}\}, $R_m$, $C_m$) to get $R_m = (rootS, m_f, \ldots, m_g)$ and $C_m = (C_f^m, \ldots, C_g^m)$, where $E$-\{e_{i+1}\} is the segment sequence $E$ without $e_{i+1}$.
4. **If** $g = i$, 
   - If $e_{i+1}$ intersects $C_i^m$ at two points $m_{i+1}^1$ and $m_{i+1}^2$, $e_{i+1} = (m_{i+1}^1, m_{i+1}^2)$, Trim-Segments($E$, $R_m$, $C_m$, $f$, $i$, null). Go to Step 2.
   - If $e_{i+1}$ intersects $C_i^m$ at one point $m_{i+1}$, $DR_s(e_i, e_{i+1}) = DR_s(e_i, e_{i+1}) \circ R_m \circ (m_{i+1})$, $DC_s(e_i, e_{i+1}) = DC_s(e_i, e_{i+1}) \circ C_m \circ (C_{i+1}^m)$, where $C_{i+1}^m$ is the cone created at $m_{i+1}$ by $C_i^m$, based on the Cone-CREATE-Rule. Return $DR_s(e_i, e_{i+1})$ and $DC_s(e_i, e_{i+1})$.
   - Otherwise, if $e_{i+1}$ does not intersects $C_i^m$, $e_i = (m_i, q_i)$, where $(m_i, q_i)$ is outside $C_i^m$, Trim-Segments($E$, $R_m$, $C_m$, $f$, $i-1$, null). Go to Step 2.
4. **Else**, if $g < i$, Trim-Segments($E$, $R_m$, $C_m$, $f$, $g$, null). Go to Step 2.
Let Proposition 2.

If \( f = i \), \( DR_s(e_i, e_{i+1}) = DR_s(e_i, e_{i+1}) \circ (m_i, m_{i+1}) \), where \( m_{i+1} \) is the middle point of \((m_1^{i+1}, m_2^{i+1})\), and \( DC_s(e_i, e_{i+1}) = DC_s(e_i, e_{i+1}) \circ (C^m_{i+1}) \), where \( C^m_{i+1} \) is the cone created at \( m_{i+1} \) by \( C^m_i \), based on the Cone-Create-Rule. Return \( DR_s(e_i, e_{i+1}) \) and \( DC_s(e_i, e_{i+1}) \).

4.2 Otherwise, if \( f < i \), \( root = S = m_f, f = f + 1 \), go to Step 2.

We first note that, \( DR_s(e_i, e_{i+1}) = (root, s_1, \ldots, s_i, s_{i+1}) \), which is found by the function Find-Division-Ray is a \((1 + \varepsilon)\)-approximation Division-Ray (proved later), not the exact one. Suppose that \( root = s \). The function Find-Division-Ray found \( DR_s(e_i, e_{i+1}) = (s, s_1, \ldots, s_l-1, s_l, \ldots, s_i, s_{i+1}) \).

Let \( DR_s^*(e_i, e_{i+1}) = (s, s_1^*, \ldots, s_l^*, s_i^*, s_{i+1}^*) \) be the exact Division-Ray between \( e_i \) and \( e_{i+1} \). For every \( j \in \{1, \ldots, i + 1\} \), \( d(s_j, s_j^*) \leq \delta \).

Another note is that, suppose that when the function Create-Euclidean-Ray calls the function Find-Division-Ray, the approximate path \( P(s, t) \) has been found a part \((s, \sigma_1, \ldots, \sigma_{l-1} = root)\). The function Find-Division-Ray then only finds the Division-Ray between \( e_i \) and \( e_{i+1} \) from \( root \). Observe that if the Division-Ray between \( e_i \) and \( e_{i+1} \) is found from \( s \), we also have \( s_j = \sigma_j \), for every \( j \in \{1, \ldots, l - 1\} \). Thus, to be simple, we only need to keep \( DR_s(e_i, e_{i+1}) \) from \( root \).

Next, we prove in Proposition 2 that, \( DR_s^*(e_i, e_{i+1}) \) is unique. Thus, when \( DR_s(e_i, e_{i+1}) \) is found and \( e_{i+1} = (p_{i+1}, q_{i+1}) \) is trimmed to remain either \((s_{i+1}, p_{i+1})\) or \((s_{i+1}, q_{i+1})\), any Euclidean-Ray \( R \) that is created after that will have the cone \( C^m_i \) on \( e_i \) intersecting \( e_{i+1} \) at only one point.

We also see that, due to the recursion used in the function Create-Euclidean-Ray, finding a Division-Ray \( DR(e_i, e_{i+1}) \) between any two consecutive segments \( e_i \) and \( e_{i+1} \), \( i \in \{1, \ldots, k - 1\} \), can be repeated many times. This might cause an exponential running time in the worst case. As deduced from Proposition 2 that if the Division-Ray \( DR(e_i, e_{i+1}) \) between two consecutive segments \( e_i \) and \( e_{i+1} \) exists, it is unique, we use a global array \( A \) to store all of the Division-Rays between two consecutive segments that appear in the calculation to avoid finding any Division-Ray for the second time.

**Proposition 2.** Let \( DR_s^*(e_i, e_{i+1}) = (s, s_1^*, \ldots, s_i^*, s_{i+1}^*) \) be the exact Division-Ray between two consecutive segments \( e_i \) and \( e_{i+1} \) in \( E \). For every \( i \in \{1, \ldots, k - 1\} \), if \( DR_s^*(e_i, e_{i+1}) \) exists, it is unique.

**Proof.** Let \( R_a \) be a Euclidean-Ray from \( s \) crossing \( E \) by starting at a point \( a_1 \) on \( e_1 \) such that \( e_{i+1} \) intersects the cone \( C^a_i \) of \( R_a \) at two points \( a_1^{i+1} \) and \( a_2^{i+1} \). Suppose that the axis ray of \( C^a_i \) is \( \overline{a_1p_1} \) (see Fig. 7a). Let \( R_b \) be a Euclidean-Ray from \( s \) crossing \( E \) by starting at a point \( b_1 \) on \( e_1 \) such that \( a_1 \neq b_1 \) and \( R_b \) intersects \( e_i \) at \( b_1 \) on \((a_i, q_i)\). We first prove the fact that, for any \( b_i \) on \((a_i, q_i)\), if \( e_{i+1} \) intersects the cone \( C^b_i \) of \( R_b \) at two points, named \( b_1^{i+1} \) and \( b_2^{i+1} \), then the segment \((b_1^{i+1}, b_2^{i+1})\) always contains the segment \((a_1^{i+1}, a_2^{i+1})\) (see Figure 7a). If this fact is correct, then \( DR_s(e_i, e_{i+1}) \) is unique.

To see that the fact is correct, by contradiction, we suppose that there exists a place on \((a_i, q_i)\) for \( b_i \) such that \( e_{i+1} \) intersects the cone \( C^b_i \) of \( R_b \) at \( b_1^{i+1} \) and \( b_2^{i+1} \), but \((b_1^{i+1}, b_2^{i+1})\) does not contain \((a_1^{i+1}, a_2^{i+1})\) (see Fig. 7b). We next prove...
The segment $(b_{i+1}^1, b_{i+1}^2)$ contains the segment $(a_{i+1}^1, a_{i+1}^2)$. (a)

The segment $(b_{i+1}^1, b_{i+1}^2)$ does not contain the segment $(a_{i+1}^1, a_{i+1}^2)$. This case cannot happen. (b)

That this case cannot exist. Let $R_\omega$ be a Euclidean-Ray from $s$ that intersects $e_i$ at $\omega_i$ such that the cone $C_\omega$ created at $\omega_i$ has the acute cone angle $90^\circ$. One of the following two conditions holds: (i) $b_i$ is on $(a_i, \omega_i)$, or (ii) $b_i$ is on $(\omega_i, q_i)$. First, for (i), if $b_i$ is on $(a_i, \omega_i)$, as presented previously, $C_b^i$ will have the same axis ray $\overrightarrow{a_i p_i}$ with $C_a^i$ and the acute cone angle of $C_b^i$ will be larger than the acute cone angle of $C_a^i$. Thus, if $e_{i+1}$ intersects $C_b^i$ at two points $b_{i+1}^1$ and $b_{i+1}^2$, the segment $(b_{i+1}^1, b_{i+1}^2)$ always contains the segment $(a_{i+1}^1, a_{i+1}^2)$. Next, for (ii), suppose that $b_i$ is on $(\omega_i, q_i)$ and $e_{i+1}$ intersects $C_b^i$ at two points $b_{i+1}^1$ and $b_{i+1}^2$. Because $b_i$ is on $(\omega_i, q_i)$, $C_b^i$ must have the axis ray being $\overrightarrow{b_i q_i}$. Let $C$ be a cone created at $b_i$ such that $C$ is opposite to $C_b^i$ with the axis ray being $\overrightarrow{b_i p_i}$ and the acute cone angle being equal to the acute cone angle of $C_b^i$. Because $e_{i+1}$ intersects $C_a^i$ and $C_b^i$ each at two points, $e_{i+1}$ must intersect $C$. Thus, $e_{i+1}$ intersects $C_b^i$ at two points and $C$ at one point at least. This case cannot happen because a line can only intersect two opposite but equal cones at two points at most.

Lemma 1. Let $\epsilon \in (0,1)$ be an error tolerance and $\delta = \frac{h \epsilon}{6k}$, where $h$ is the minimum distance between two consecutive segments in $E$. The Euclidean length of $P(s,t)$ found by the function Find-Approximate-Path is at most $(1 + \epsilon)$ times the Euclidean length of $P^*(s,t)$.
Proof. We need to prove $d(P(s, t)) \leq (1 + \epsilon)d(P^*(s, t))$. The following proof is deduced from the proof of Lemma 8.1 in [5]. First, we prove, by induction, that (1) holds for every $i \in \{0, \ldots, k\}$:

$$d(P(\sigma_i, t)) \leq \left(1 + \frac{\epsilon}{2}\right)d(P^*(\sigma_i, t)) + 3(k - i)\delta$$  \hspace{1cm} (1)

When $i = k$, (1) becomes $d(P(\sigma_k, t)) \leq (1 + \frac{\epsilon}{2})d(P^*(\sigma_k, t))$. We have $P(\sigma_k, t) = P^*(\sigma_k, t) = (\sigma_k, t)$ (see Fig. 8). Thus, (1) holds for $i = k$. Assume that (1) holds for $i = j$, we have,

$$d(P(\sigma_j, t)) \leq \left(1 + \frac{\epsilon}{2}\right)d(P^*(\sigma_j, t)) + 3(k - j)\delta$$  \hspace{1cm} (2)

We need to prove that (1) also holds for $i = j - 1$, which is

$$d(P(\sigma_{j-1}, t)) \leq \left(1 + \frac{\epsilon}{2}\right)d(P^*(\sigma_{j-1}, t)) + 3(k - j + 1)\delta$$  \hspace{1cm} (3)

Then based on (2), we have,

$$d(P(\sigma_{j-1}, t)) = d(\sigma_{j-1}, \sigma_j) + d(P(\sigma_j, t))$$

$$\leq d(\sigma_{j-1}, \sigma_j) + \left(1 + \frac{\epsilon}{2}\right)d(P^*(\sigma_j, t)) + 3(k - j)\delta$$  \hspace{1cm} (4)

Suppose that $P^*(\sigma_{j-1}, t)$, which is the exact shortest path from $\sigma_{j-1}$ to $t$, crosses $e_j$ at $z_j$. We have,

$$d(P^*(\sigma_j, t)) \leq d(\sigma_j, z_j) + d(P^*(z_j, t))$$  \hspace{1cm} (5)

From (4) and (5), we have,

$$d(P(\sigma_{j-1}, t)) \leq d(\sigma_{j-1}, \sigma_j) + \left(1 + \frac{\epsilon}{2}\right)(d(\sigma_j, z_j) + d(P^*(z_j, t))) + 3(k - j)\delta$$

$$= d(\sigma_{j-1}, \sigma_j) + \left(1 + \frac{\epsilon}{2}\right)d(\sigma_j, z_j) + 3(k - j)\delta + \left(1 + \frac{\epsilon}{2}\right)d(P^*(z_j, t))$$  \hspace{1cm} (6)
In the triangle created by $\sigma_{j-1}$, $\sigma_j$, $z_j$, we have, $d(\sigma_{j-1}, \sigma_j) \leq d(\sigma_j, z_j) + d(\sigma_{j-1}, z_j)$. Thus, the part $B$ of (6) becomes

$$B \leq d(\sigma_j, z_j) + d(\sigma_{j-1}, z_j) + \left(1 + \frac{\epsilon}{2}\right) d(\sigma_j, z_j) + 3(k - j)\delta$$

$$= d(\sigma_{j-1}, z_j) + \left(2 + \frac{\epsilon}{2}\right) d(\sigma_j, z_j) + 3(k - j)\delta$$

(7)

Because $d(\sigma_j, z_j) \leq \delta$, we have $(2 + \frac{\epsilon}{2}) d(\sigma_j, z_j) < 3d(\sigma_j, z_j) \leq 3\delta$. Thus, (7) becomes

$$B \leq d(\sigma_{j-1}, z_j) + 3(k - j + 1)\delta$$

(8)

From (6) and (8), we have,

$$d(P(\sigma_{j-1}, t)) \leq d(\sigma_{j-1}, z_j) + 3(k - j + 1)\delta + \left(1 + \frac{\epsilon}{2}\right) d(P^*(z_j, t))$$

$$\leq \left(1 + \frac{\epsilon}{2}\right) (d(\sigma_{j-1}, z_j) + d(P^*(z_j, t))) + 3(k - j + 1)\delta$$

$$= \left(1 + \frac{\epsilon}{2}\right) d(P^*(\sigma_{j-1}, t)) + 3(k - j + 1)\delta$$

(9)

Thus, (3) is correct, which means that (1) holds for every $i \in \{0, \ldots, k\}$. Now, with $i = 0$, (1) becomes

$$d(P(\sigma_0, t)) \leq \left(1 + \frac{\epsilon}{2}\right) d(P^*(\sigma_0, t)) + 3k\delta$$

$$d(P(s, t)) \leq \left(1 + \frac{\epsilon}{2}\right) d(P^*(s, t)) + \frac{h\epsilon}{2}, \text{ because } \delta = \frac{h\epsilon}{6k}$$

(10)

Since $h \leq d(P^*(s, t))$, (10) becomes

$$d(P(s, t)) \leq \left(1 + \frac{\epsilon}{2}\right) d(P^*(s, t)) + \frac{\epsilon}{2} d(P^*(s, t))$$

$$= (1 + \epsilon) d(P^*(s, t))$$

(11)

\[ \square \]

**Lemma 2.** The function Find-Approximate-Path runs in $O(k^3 \log(\frac{L}{h\epsilon}))$ time, where $L$ is the maximum Euclidean length of the segments in $E$ and $h$ is the minimum distance between two consecutive segments in $E$.

**Proof.** We first consider the case that in the function Create-Euclidean-Ray, when creating any Euclidean-Ray, finding Division-Rays is not required. The case that finding Division-Rays is required will be considered later. Initially, the function Find-Approximate-Path sets $l = 1$ and root = $s$. Then a Euclidean-Ray $R_m$ from root, starting at $m_l$ on $e_l$ is created. Based on $R_m$, the segments in $E$ are trimmed by the function Trim-Segments. After every trimming, the length of $e_l$ is reduced by a half. Thus, to make the length of $e_l$ less than or equal to $\delta$, we need to create $O(\log \frac{L}{\delta})$ Euclidean-Rays. Since $E$ has $k$ segments, in the worst case, we need to create $O(k \log(\frac{L}{\delta}))$ Euclidean-Rays along with trimming.
Creating a Euclidean-Ray and then trimming the segments in \( E \) based on the Euclidean-Ray take \( O(k) \) time. Thus, the total running time, in the worst case, of the function \( \text{Find-Approximate-Path} \) when finding Division-Rays is not required is \( O(k^2 \log \left( \frac{L}{\delta} \right)) \).

The idea of the function \( \text{Find-Division-Ray} \) is similar to the idea of the function \( \text{Find-Approximate-Path} \). Thus, the running time of the function \( \text{Find-Division-Ray} \) when finding the Division-Ray between two consecutive segments \( e_i \) and \( e_{i+1} \), \( i \in \{1, \ldots, k - 1 \} \), in the case that no other Division-Ray between \( e_j \) and \( e_{j+1} \), \( j < i \), is required, is \( O(i^2 \log \left( \frac{L}{\delta} \right)) \).

We now consider the case that in the function \( \text{Create-Euclidean-Ray} \), finding any Division-Ray is not required. However, if a Division-Ray \( ds \) \((e_i, e_{i+1})\) is required, we have \( \text{TER}(l) = O(k) + \text{TER}(i, i + 1) + \text{TER}(i + 1)^* + O(k) + \text{TER}(i + 1) \), which is explained as follows.

The Euclidean-Ray \( R_m \) is created from \( e_1 \) to \( e_k \), taking \( O(k) \) time in the worst case, then will be stopped to wait for finding the Division-Ray \( \text{DR}_s(e_i, e_{i+1}) \). Finding \( \text{DR}_s(e_i, e_{i+1}) \) takes \( \text{TER}(i, i + 1) \) time. If \( \text{DR}_s(e_i, e_{i+1}) \) is calculated previously and stored in the global array \( A \), then, continuing finding the Euclidean-Ray \( R_s \) for \( \text{DR}_s(e_i, e_{i+1}) \) takes \( \text{TER}(i + 1)^* \) time. Trimming the segments in \( E \) based on \( R_s \) takes \( O(k) \) time, in the worst case. After this trimming, \( R_m \), which is stopped at \( e_i \), now can continue being calculated, takes \( \text{TER}(i + 1)^* \) time. We use \( \text{TER}(i + 1)^* \) and \( \text{TER}(i + 1) \) just to distinguish between the running times of creating Euclidean-Rays from \( e_{i+1} \) for \( R_s \), and from \( e_{i+1} \) for \( R_m \), respectively. Observe that, finding another Division-Ray \( \text{DR}_s(e_j, e_{j+1}) \), \( i \neq j \), can be required in the processes of \( \text{TER}(i, i + 1) \), or \( \text{TER}(i + 1)^* \) of \( R_s \), or \( \text{TER}(i + 1) \) of \( R_m \). Thus, we consider the worst case that all the Division-Rays between every two consecutive segments, from \( \text{DR}_s(e_1, e_2) \) to \( \text{DR}_s(e_{k-1}, e_k) \), are required. Let \( f \) be the number of the Euclidean-Rays that related to the appearance of \( \text{DR}_s(e_1, e_2) \) to \( \text{DR}_s(e_{k-1}, e_k) \). We have \( f \leq k - 1 \). Let \( T_f \) be the running time for calculating these \( f \) Euclidean-Rays. We have \( T_f = O(fk) + TDR(e_1, e_2) + \cdots + TDR(e_{k-1}, e_k) \). Every \( TDR(e_i, e_{i+1}) \), \( i \in \{1, \ldots, k - 1\} \), at this time, is the running time of the function \( \text{Find-Division-Ray} \), where no other Division-Ray \( \text{DR}_s(e_j, e_{j+1}) \), \( j < i \), is required to find. As presented above, in this case, \( TDR(e_i, e_{i+1}) = O(i^2 \log \left( \frac{L}{\delta} \right)) \). Therefore, \( T_f = O(fk + (1^2 + 2^2 + \cdots + (k - 1)^2) \log \left( \frac{L}{\delta} \right)) = O(fk + \frac{(k-1)k(2k-1)}{6} \log \left( \frac{L}{\delta} \right)) = O(k^3 \log \left( \frac{L}{\delta} \right)) \).

Also as presented above, the function \( \text{Find-Approximate-Path} \) needs to create total \( O(k \log \left( \frac{L}{\delta} \right)) \) Euclidean-Rays, where the segments in \( E \) will be trimmed right after every of these Euclidean-Rays is created. Thus, excluding the \( f \) Euclidean-Rays that take \( T_f \) time above, the function \( \text{Find-Approximate-Path} \) needs \( O(k \log \left( \frac{L}{\delta} \right) - f) \) Euclidean-Rays where finding any Division-Ray is not
required, or it can be required, but it is stored in the global array \( A \) already. Let \( T_r \) be the running time for creating these \( O(k \log(\frac{k}{\delta}) - f) \) Euclidean-Rays. We have, \( T_r = O(k(k \log(\frac{k}{\delta}) - f)) = O(k^2 \log(\frac{k}{\delta})) \). Let \( T_t \) be the total time of trimming the segments in \( E \) after every Euclidean-Ray is created. We have \( T_t = O(k^2 \log(\frac{k}{\delta})) \). In total, the function \( \text{Find-Approximate-Path} \) takes \( T_f + T_r + T_t = O(k^3 \log(\frac{kL}{h\epsilon})) = O(k^3 \log(\frac{kL}{h\epsilon^2})) \), where \( \delta = \frac{h\epsilon}{6k} \).

\[ \square \]

4 Conclusion

We have presented a \((1 + \epsilon)\)-approximation algorithm, running in \( O(k^3 \log(\frac{kL}{h\epsilon})) \) time for the ESP-3D-3S problem. We first propose in Proposition 1 that two Euclidean-Rays cannot intersect with each other. Based on this geometrical characteristic, the algorithm is created. The most difficult problem that we need to process in the algorithm is that, in some cases, a segment \( e_{i+1} \) can intersect a cone \( C_i^e \) at two points, which can lead the number of the possible optimal paths to increase exponentially. We then use the idea of the Division-Rays to deal with this difficulty. For future work, we will measure the running times of the algorithm by using practical experiments.

References

1. Canny, J., Reif, J.: New lower bound techniques for robot motion planning problems. In: 28th Annual Symposium on Foundations of Computer Science, pp. 49–60 (1987)
2. Akman, V.: Solution of the general instance of FINDPATH. In: Akman, V. (ed.) Unobstructed Shortest Paths in Polyhedral Environments. LNCS, vol. 251, pp. 19–33. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17629-2_8
3. Bajaj, C., Moh, T.T.: Generalized unfoldings for shortest paths. Int. J. Robot. Res. 7(1), 71–76 (1988)
4. Trang, L.H., Truong, Q.C., Dang, T.K.: An iterative algorithm for computing shortest paths through line segments in 3D. In: Dang, T.K., Wagner, R., Küng, J., Thoai, N., Takizawa, M., Neuhold, E.J. (eds.) FDSE 2017. LNCS, vol. 10646, pp. 73–84. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70004-5_5
5. Mitchell, J.S.B., Papadimitriou, C.H.: The weighted region problem: finding shortest paths through a weighted planar subdivision. J. ACM 38(1), 18–73 (1991)
6. Polishchuk, V., Mitchell, J.S.: Touring convex bodies - a conic programming solution. In: Canadian Conference on Computational Geometry, pp. 290–293 (2005)
7. Aleksandrov, L., Maheshwari, A., Sack, J.R.: Approximation algorithms for geometric shortest path problems. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 286–295 (2000)
8. Mitchell, J.S.B.: Chapter 31: shortest paths and network. In: Toth, C.D., O’Rourke, J., Goodman, J.E. (eds.) Handbook of Discrete and Computational Geometry. Chapman and Hall/CRC (2017)
9. Bose, P., Maheshwari, A., Shu, C., Wuhrer, S.: A survey of geodesic paths on 3D surfaces. Comput. Geom. 44(9), 486–498 (2011)
10. Burago, D., Grigoriev, D., Slissenko, A.: Approximating shortest path for the skew lines problem in time doubly logarithmic in 1/\( \epsilon \). Theoret. Comput. Sci. 315(2–3), 371–404 (2004)