Chapter 1

Euclidean Shortest Paths in Simple Polygons

Fajie Li and Reinhard Klette
Computer Science Department, The University of Auckland
Auckland, New Zealand

Let p and q be two points in a simple polygon Π. This chapter provides two rubberband algorithms for computing a shortest path between p and q that is contained in Π. The two algorithms are based on previously known results on triangular or trapezoidal decompositions of simple polygons, and have either $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ or $\kappa(\varepsilon) \cdot \mathcal{O}(n\ log\ n)$ time complexity, where $\kappa(\varepsilon) = (L_0 - L)/\varepsilon$, for the true length L of the shortest path and length L_0 of a used initial polygonal path. Rubberband algorithms follow a straightforward design strategy, and the proposed algorithms are easy to implement (after having the decompositions at hand).

1.1. Introduction

Algorithms for computing Euclidean shortest paths (ESPs) between two points p and q of a simple polygon Π, where the path is restricted to be fully contained in Π, have applications in two-dimensional (2D) pattern recognition, picture analysis, robotics, and so forth. They have been intensively studied.[6–8,14]

There is Chazelle's[3] linear-time algorithm for triangulating a simple polygon, or an easier to describe, but $\mathcal{O}(n \log n)$ algorithm for partitioning a simple polygon into trapezoids.[13] (The super-linear time complexity is only due to an initial sorting step for all vertices of the given simple polygon Π.) The design of algorithms for calculating ESPs within a simple polygon may use one of both partitioning algorithms as a preprocess. This chapter shows how rubberband algorithms[2] may be used to calculate approximate ESPs within simple polygons, using either decompositions into triangles or into trapezoids.

For a start we prove a basic property of exact ESPs for such cases; see also:[10]

Proposition 1.1. *Each vertex ($\neq$ p, q) of the shortest path is a vertex of* Π.

To see this, let $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$ be the shortest path from p to q completely contained in simple polygon Π. Assume that at least one $p_i \in \rho$ is not a vertex of Π. Also assume that each p_i is not *redundant*, which means that $p_{i-1}p_ip_{i+1}$

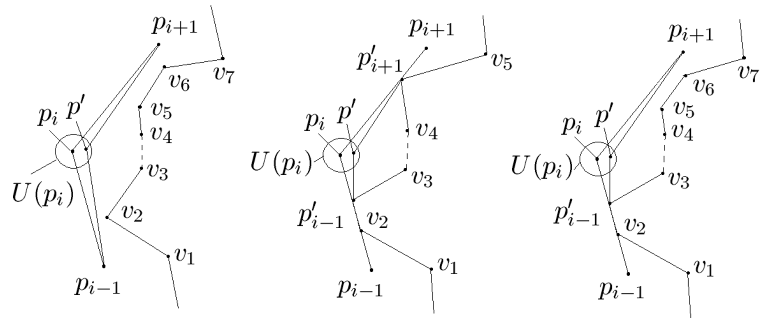

Fig. 1.1. Illustration that each vertex of a shortest path is a vertex of Π, where $v_1v_2v_3v_4v_5...$ is a polygonal part of the border of the simple polygon Π. Left, middle, right illustrate Cases 1, 2, 3 as discussed in the text, respectively.

must be a triangle (i.e., three points p_{i-1}, p_i and p_{i+1} are not collinear), where $i = 1, 2, \ldots, k$ and $p_0 = p$, $p_{k+1} = q$.

Case 1: Non of the two edges $p_{i-1}p_i$ and p_ip_{i+1} is on a tangent of Π (see Figure 1.1, left); then there exists a sufficiently small neighborhood of p_i, denoted by $U(p_i)$, such that for each point $p' \in U(p_i) \cap \triangle p_{i-1}p_ip_{i+1} \subset \Pi^\bullet$ (the topological closure of a simple polygon Π), both edges $p_{i-1}p_i$ and p_ip_{i+1} are completely contained in Π. By elementary geometry, we have that $d_e(p_{i-1}, p') + d_e(p', p_{i+1}) < d_e(p_{i-1}, p_i) + d_e(p_i, p_{i+1})$, where d_e denotes Euclidean distance. Therefore we may obtain a shorter path from p to q by replacing p_i by p'. This is a contraction to the assumption that p_i is a vertex of the shortest path ρ.

Case 2: Both $p_{i-1}p_i$ and p_ip_{i+1} are on tangents of Π (see Figure 1.1, middle); then we can also derive a contradiction. In fact, let p'_{i-1} and p'_{i+1} be the closest vertices of Π such that $p'_{i-1}p_i$ and $p_ip'_{i+1}$ are on tangents of Π. Analogous to the first case, there exists a point p' such that the polygonal path $p'_{i-1}p'p'_{i+1}$ is completely contained in $\Pi^\bullet$ and the length of $p'_{i-1}p'p'_{i+1}$ is shorter than $p'_{i-1}p_ip'_{i+1}$. This is a contradiction as well.

Case 3: Either $p_{i-1}p_i$ or p_ip_{i+1} is a tangent of Π (see Figure 1.1, right); then we may arrive at the same result as in Case 2.

This chapter is organized as follows. At first we introduce into rubberband algorithms. Then we recall briefly decompositions of simple polygons and specify (as a preliminary result) two approximate rubberband algorithms; we provide examples of using them. All the given algorithms are then analyzed with respect to correctness and time complexity. We also provide suggestions how to further improve the given two rubberband algorithms.

1.2. Basics of Rubberband Algorithms

We explain basic ideas of a rubberband algorithm by using the following, very simple 2D example. In general, rubberband algorithms are for (approximate) calculations

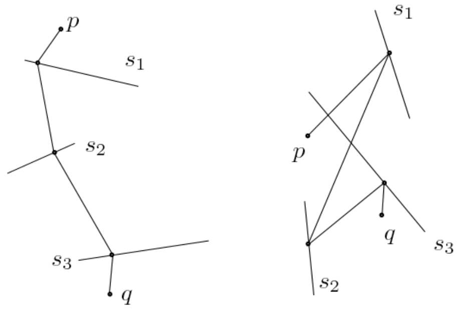

Fig. 1.2. Two step sets with possible initializations of Algorithm 1, both for $k = 3$.

of ESPs for 2D or 3D applications.[12]

Let Π be a plane. Assume that there are $k > 1$ line segments $s_i \subset \Pi$ (for $i = 1, 2, \ldots, k$) such that $s_i \cap s_j = \emptyset$, for $i \neq j$ and $i, j = 1, 2, \ldots, k$; see Figure 1.2. The following simple rubberband algorithm (see Figure 1.3) approximates a shortest path from p to q that intersects all the given segments s_i (at least once) in the given order.

The accuracy parameter in Step 1 can be chosen such that maximum possible numerical accuracy (on the given computer) is guaranteed. The initial path in Step 2 may, for example, be defined by centers of line segments. Vertices of the calculated path move by local optimization, until the total length of the path between two iterations only differs by ε at most. The series of lengths L calculated for each

```
1. Let $\varepsilon = 10^{-10}$ (the chosen accuracy).
2. Compute the length $L_1$ of the initial path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
3. Let $q_1 = p$ and $i = 1$.
4. While $i < k - 1$ do:
4.1. Let $q_3 = p_{i+1}$.
4.2. Compute a point $q_2 \in s_i$ such that
       $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in s_i\}$.
4.3. Update $\rho$ by replacing $p_i$ by $q_2$.
4.4. Let $q_1 = p_i$ and $i = i + 1$.
5.1. Let $q_3 = q$.
5.2. Compute $q_2 \in s_k$ such that
       $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in s_k\}$.
5.3. Update $\rho$ by replacing $p_k$ by $q_2$.
6. Compute the length $L_2$ of the updated path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
7. Let $\delta = L_1 - L_2$.
8. If $\delta > \varepsilon$, then let $L_1 = L_2$ and go to Step 3.
   Otherwise, stop.
```

Fig. 1.3. Algorithm 1: a simple rubberband algorithm for a given set of line segments.

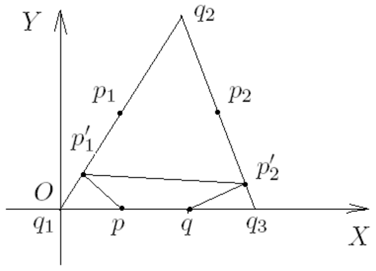

Fig. 1.4. Illustration of steps with joint endpoints.

iteration forms a decreasing Cauchy sequence lower bounded by zero, and is thus guaranteed to converge to a minimum length. The path defined by this convergence is called *the limit path* of Algorithm 1. In relation to Proposition 1.1, we have the following for Algorithm 1:

Proposition 1.2. *Each vertex ($\neq p$, q) of the limit path of Algorithm 1 is a vertex of* Π.

Proof. Let $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$ be the limit path from p to q of Algorithm 1. Let $i = 1, 2, \ldots,$ or k and $p_0 = p$, $p_{k+1} = q$. Assume that each $p_i \in \rho$ is not redundant. Then p_i must be an endpoint of s_i. (Otherwise, $p_i = p_{i-1}p_{i+1} \cap s_i$. This contradicts the assumption that p_i is not redundant.) It follows that p_i must be a vertex of Π. □

The set $\{s_1, s_2, \ldots, s_k\}$ is a *step set* of a rubberband algorithm if its union contains all the vertices of the calculated path, and each s_i is a *step element* of the rubberband algorithm that contains at least one vertex of the calculated path, for $i = 1, 2, \ldots, k$.

In this chapter, step sets are sets of line segments, which may have joint endpoints, but cannot have further points in common. Furthermore, in this chapter, each step element contains exactly one vertex of the shortest path. For example, if the input for Algorithm 1 is as in Figure 1.4, with

$$s_1 = q_1q_2,\ s_2 = q_2q_3,\ q_1 = (0,0),\ q_2 = (2,4),\ q_3 = (3,0),\ p = (1,0),\ q = (2,0)$$

then we also have segments with joint endpoints. Assume a path initialization using p_1 and p_2, the centers of s_1 and s_2, respectively [i.e., $p_1 = (1,2)$, and $p_2 = (2.5,2)$]. We obtain that the length of the initialized polyline $\rho = \langle p, p_1, p_2, q\rangle$ is equal to 5.5616 (rounded to four digits). Algorithm 1 calculates an approximate shortest path $\rho = \langle p, p_1', p_2', q\rangle$ where $p_1' = (0.3646, 0.7291)$, $p_2' = (2.8636, 0.5455)$ and the length of it is equal to 4.4944 (see Table 1.1, which lists resulting δs for the number I of iterations). That means, Algorithm 1 is also able to deal with this input for the assumed initialization.

However, if we assume a different initialization, such that $p_1 = p_2 = q_2$; in this case, Algorithm 1 will fail because the output of Step 4.2 in Algorithm 1 will be

Table 1.1. Number I of iterations and resulting δs for the initialization illustrated by Figure 1.4 [i.e., with $p_1 = (1, 2)$ and $p_2 = (2.5, 2)$ as initial points on the path].

I	δ	I	δ	I	δ	I	δ
1	-0.8900	3	-0.0019	5	-8.4435e-008	7	-3.5740e-012
2	-0.1752	4	-1.2935e-005	6	-5.4930e-010		

false: the calculated path equals $\rho = \langle p, p'_1, p'_2, q\rangle$, where $p'_1 = q_2$ and $p'_2 = q_2$, and its length equals 8.1231. (Referring to Lemma 16,[11] we see that $p_1 \neq p_0$ and p_2 in this example.)

We call a situation as in this initialization example a *degenerate path* within an application of a rubberband algorithm, and it may occur within initialization, or at a later iteration of the algorithm. In general, it is defined by the occurrence of at least two identical vertices of an initial or updated polygonal path. Such a degenerate case causes Step 4.2 in Algorithm 1 to fail.

A degenerate path can be dealt with approximately: we will not allow $p_2 = q_2$. To do so, we remove sufficiently small segments from both segments s_1 and s_2. The following shows how to handle such a degenerate case (for example) for the assumed data in Figure 1.4.

We modify the initial values of x_1 and x_2, and of y_1 and y_2 as follows:

$$\begin{aligned} &\delta' = 2.221 \times 10^{-16} \quad \text{(for a reason, see below)} \\ &x_1 = 2 - \delta' \quad \text{and} \quad y_1 = 2 \times x_1 \\ &x_2 = 2 + \delta' \quad \text{and} \quad y_2 = -4 \times (x_2 - 3) \\ &p_1 = (x_1, y_1) \quad \text{and} \quad p_2 = (x_2, y_2) \end{aligned}$$

Furthermore, let the accuracy be equals $\varepsilon = 1.0 \times 10^{-100}$. The length of the initialized polyline $\rho = \langle p, p_1, p_2, q\rangle$ is equal to 8.1231. Algorithm 1 will approximate a shortest path $\rho = \langle p, p'_1, p'_2, q\rangle$, where $p'_1 = (0.3646, 0.7291)$ and $p'_2 = (2.8636, 0.5455)$, and its length equals 4.4944 (see Table 1.2 for resulting δs in dependency of the number I of iterations).

Of course, if we leave the accuracy to be equals $\varepsilon = 1.0 \times 10^{-10}$ then the

Table 1.2. Number I of iterations and resulting δs, for the example shown in Figure 1.4, with $p_1 = (2 - \delta', 2(2 - \delta'))$ and $p_2 = (2 + \delta', -4((2 + \delta') - 3))$ as initialization points and $\delta' =$ 2.221e-16.

I	δ	I	δ	I	δ	I	δ
1	-5.4831e-007	7	-1.2313	13	-7.0319e-010	19	8.8818e-016
2	-6.2779e-006	8	-2.0286	14	-4.5732e-012	20	8.8818e-016
3	-7.7817e-005	9	-0.2104	15	-3.0198e-014	21	-8.8818e-016
4	-9.6471e-004	10	-0.0024	16	-8.8818e-016	22	8.8818e-016
5	-0.0119	11	-1.6550e-005	17	8.8818e-016	23	-8.8818e-016
6	-0.1430	12	-1.0809e-007	18	-8.8818e-016	24	0

algorithm will stop sooner, after less iterations. – The algorithm was implemented on a Pentium 4 PC using Matlab 7.04. If we changed the value of δ' into $\delta' = 2.22 \times 10^{-16}$ then we obtained the same wrong result as that for identical points $p_1 = p_2 = q_2$. This is because the computer was not able to recognize a difference between x_1 and $x_1 \mp 2.22 \times 10^{-16}$. However, for practical applications, the value $\delta' = 2.221 \times 10^{-16}$ should be small or accurate enough in general (for this or a matching implementation environment).

With the example above we also illustrate that the approximate algorithm may be already *de facto an exact algorithm* if ε was chosen small enough (i.e., obtained results are accurate within the given numerical limits of the used implementation environment).

1.3. Decompositions and Approximate ESPs

There are (at least) two ways of decomposing a simple polygon: into triangles[3] or trapezoids.[13] In the first case, Theorem 4.3[3] says that it is possible to compute a triangulation of a simple polygon in linear time (and the algorithm is "fairly complicated"). In the second case, Theorem 1[13] says that a given ("simple") algorithm for the decomposition into trapezoids has time complexity $\mathcal{O}(n\ log\ n)$, where n is the number of vertices of the original simple polygon Π.

Step sets can be defined by selecting edges of triangles or trapzoids of those decompositions.

1.3.1. *Triangulation*

Let Π be a simple polygon. Let $T_1 = \{\triangle_1, \triangle_2, \ldots, \triangle_m\}$ be such that $\Pi = \cup_{i=1}^{m} \triangle_i$ and $\triangle_i \cap \triangle_j = \emptyset$ or $= e_{ij}$, where e_{ij} is an edge of both triangles $\triangle_i$ and $\triangle_j$, $i \neq j$ and $i, j = 1, 2, \ldots, m$. We construct a corresponding simple graph $G = [V, E]$ where $V = \{v_1, v_2, \ldots, v_m\}$ and each edge $e \in E$ is defined as follows: If $\triangle_i \cap \triangle_j = e_{ij} \neq \emptyset$, then let $e = v_i v_j$ (where e_{ij} is an edge of both triangles $\triangle_i$ and $\triangle_j$); and if $\triangle_i \cap \triangle_j = \emptyset$, then there is not an edge between v_i and v_j, $i < j$ and $i, j = 1, 2, \ldots, m$. We say that G is a *(corresponding) graph* with respect to the triangulated simple polygon Π, denoted by G_Π.

Lemma 1.1. *For each triangulated simple polygon Π, its corresponding graph G_Π is a tree.*

Proof. By contradiction. Suppose that G_Π is not a tree. Then there is a cycle $u_1 u_2 \cdots u_{m'} u_1$ in G_Π. Consequently, there are a sequence of triangles $\{\triangle'_1, \triangle'_2, \ldots, \triangle'_{m'}\} \subseteq T_1$ such that $\triangle'_i \cap \triangle'_j \neq \emptyset$, where $i \neq j$ and $i, j = 1, 2, \ldots, m'$. It follows that there is a polygonal curve $\rho = w_1 w_2 \cdots w_{m'} w_1 \subset \cup_{i=1}^{m'} \triangle'_i$. Since Π is a simple polygon, ρ can be contracted into a single point inside of Π. Note that we can find ρ such that there is a vertex of $\triangle'_1$, denoted by w, that is

```
Input: the (original) tree $T$ and two points $p'$, $q' \in V(T)$.
Output: a unique path $\rho$ from $p'$ to $q'$ in $T$.

1. Let $S_i = \{v : d(v) = 1 \wedge v \in V(T)\} \setminus \{p', q'\}$.
2. If $S_i = \emptyset$, stop (the current $T$ is already a path from $p'$ to $q'$).
3. Otherwise, let Let the unique neighbor of $v \in S_i$ be $n_v$.
4. For each $v \in S_i$, do the following:
4.1. Let $V_{i_j} = \emptyset$.
4.2. While $d(n_v) = 1$ do:
4.2.1. If $v = p'$ or $q'$, then skip this while loop.
4.2.2. Otherwise, let $V_{i_j} = V_{i_j} \cup \{v\}$.
4.2.3. $v = n_v$
4.3. Update $T$ by removing $v$ from the set of neighbors of $n_v$.
4.4. Update $T$ by removing $V_{i_j}$ from $V(T)$.
5. Goto Step 1.
```

Fig. 1.5. Procedure 1: step set calculation for a given triangulation.

inside of the region enclosed by ρ. Therefore, w must be a redundant vertex. This contradicts to the fact that w is a vertex of Π. □

Let T be a tree and $p \neq q$, p, $q \in V(T)$. The following procedure will compute a unique path from p to q in T. Although there exists a linear algorithm for computing the shortest path between two vertices in a positive integer weighted graph,[19] our procedure below is much simpler because here the graph is (just) a tree.

We apply Procedure 1 (see Figure 1.5) as follows: Let $T = G_\Pi$ and p', q' be the vertices of T corresponding to the triangle containing p, q, respectively. Let a sequence of triangles $\{\triangle'_1, \triangle'_2, \ldots, \triangle'_{m'}\}$ correspond to the vertices of the path calculated by Procedure 1. Let $\{e_1, e_2, \ldots, e_{m'-1}\}$ be a sequence of edges such that $e_i = \triangle_i \cap \triangle_{i+1}$, where $i = 1, 2, \ldots, m' - 1$. Let $\{e'_1, e'_2, \ldots, e'_{m'-1}\}$ be a sequence of edges such that e'_i is obtained by removing a sufficiently small segment (Assume that the length of the removed segment is δ'.) from both endpoints of e_i, where $i = 1, 2, \ldots, m' - 1$. Set $\{e'_1, e'_2, \ldots, e'_{m'-1}\}$ is the approximate step set we are looking for.

1.3.2. *Trapezoidal Decomposition*

Analogously to Section 1.3.1, let Π be a simple polygon, and let $T_2 = \{t_1, t_2, \ldots, t_m\}$ be such that $\Pi = \cup_{i=1}^{m} t_i$ and $t_i \cap t_j = \emptyset$ or e_{ij}, where e_{ij} is a part (a subset) of a joint edge of trapezoids t_i and t_j, $i \neq j$ and $i, j = 1, 2, \ldots, m$. We construct a corresponding simple graph $G = [V, E]$ where $V = \{v_1, v_2, \ldots, v_m\}$, and each edge $e \in E$ is defined as follows: If $t_i \cap t_j = e_{ij} \neq \emptyset$, then let $e = v_i v_j$ (where e_{ij} is a subset of a joint edge of trapezoids t_i and t_j); and if $t_i \cap t_j = \emptyset$, then there is not an edge between v_i and v_j, $i < j$ and $i, j = 1, 2, \ldots, m$. We say that G is a *(corresponding)* *graph* with respect to the trapezoidal decomposition of simple polygon Π, denoted by G_Π. – Analogously to Lemma 1.1, we also have the following

Lemma 1.2. *For each trapezoidal decomposition of a simple polygon* Π*, its corresponding graph* G_Π *is a tree.*

Following Section 1.3.1, we apply Procedure 1 as follows: Let $T = G_\Pi$ and p', q' be the vertices of T corresponding to the trapezoids containing p, q respectively. Let a sequence of trapezoids $\{t'_1, t'_2, \ldots, t'_{m'}\}$ correspond to the vertices of the path obtained by Procedure 1. Let $E' = \{e_1, e_2, \ldots, e_{m'-1}\}$ be a sequence of edges such that $e_i = t_i \cap t_{i+1}$, where $i = 1, 2, \ldots, m' - 1$. For each $i \in \{1, 2, \ldots, m' - 2\}$, if $e_i \cap e_{i+1} \neq \emptyset$, then update e_i and e_{i+1} in E' by removing sufficiently small segments from both sides of this intersection point. Then the updated set E' is the approximate step set.

1. Apply Chazelle's algorithm to decompose Π into triangles.
2. Construct the corresponding graph with respect to the decomposed Π, denoted by G_Π.
3. Apply Procedure 1 to compute the unique path from p' to q', denoted by ρ.
4. Let $\delta' = \varepsilon$. Compute the step set from ρ, denoted by S, where removed segments have length δ'.
5. Let S, p and q as input, apply Algorithm 1 to compute the approximate ESP from p to q.

Fig. 1.6. Algorithm 2: approximate ESP after triangulation.

Modify Step 1 in Algorithm 2 as follows:
Apply a trapezoidal decomposition algorithm[13] to Π.

Fig. 1.7. Algorithm 3: approximate ESP after trapezoidal decomposition.

1.3.3. *Two Approximate Algorithms*

Figures 1.6 and 1.7 show the main algorithms having decomposition, step set construction, and ESP approximation as their subprocedures. For Step 4, see the description following Lemma 1.1. For Step 5 note that the approximation is not due to Algorithm 1 but due to removing small segments of length δ'.

We illustrate Algorithms 2 and 3 by a few examples, using the simple polygon in Figure 1.8, with coordinates of vertices provided in Table 1.3. After illustrating triangulation and Algorithm 2, we also illustrate decomposition into trapezoids and Algorithm 3.

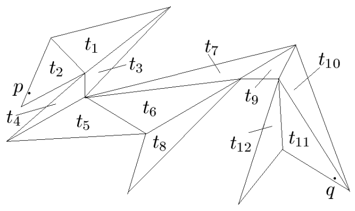

Fig. 1.8. A possible triangulation of a simple polygon.

Table 1.3. Vertices of the simple polygon in Figure 1.8, where p = (59,201) and q = (707,382).

v_i	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
x_i	42	178	11	306	269	506	589	503	595	736	623	176	358	106
y_i	230	158	304	286	411	173	173	436	320	408	100	211	19	84

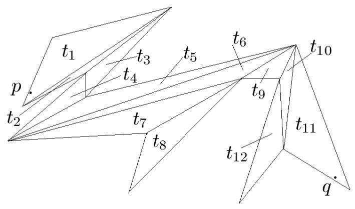

Fig. 1.9. Another possible triangulation of the simple polygon of Figure 1.8.

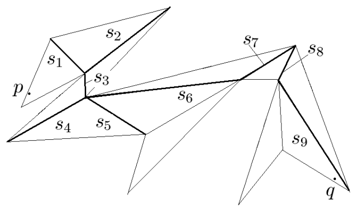

Fig. 1.10. The step set of the triangulation shown in Figure 1.8.

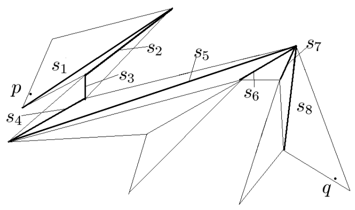

Fig. 1.11. The step set of the triangulation shown in Figure 1.9.

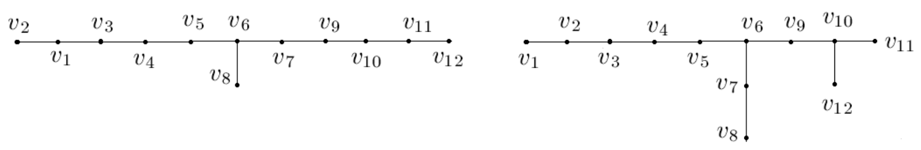

Fig. 1.12. Left (right): corresponding graph (tree) with respect to Figure 1.8 (Figure 1.9).

Table 1.4. Vertices p_i calculated by Algorithm 2 for the simple polygon in Figure 1.8. The length of the path equals 1246.0330730004.

p_i	(x_i, y_i)	p_i	(x_i, y_i)
p_1	(177.9999999928, 157.9999999926)	p_6	(374.5899740372, 188.1320635957)
p_2	(178.000000018, 157.9999999861)	p_7	(506.0000000117, 172.9999999927)
p_3	(176.9605570407, 185.5452384224)	p_8	(589.0000000034, 172.9999999927)
p_4	(175.9999999835, 211.0000000093)	p_9	(589.0000000772, 173.0000001234)
p_5	(176.000000013, 211.0000000075)		

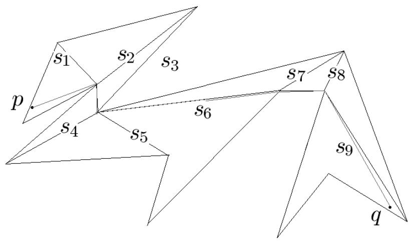

Fig. 1.13. The approximate ESP with respect to the triangulated simple polygon of Figure 1.8.

Table 1.5. Vertices p_i calculated by Algorithm 2 for the simple polygon in Figure 1.9. The length of the path equals 1323.510103408.

p_i	(x_i, y_i)	p_i	(x_i, y_i)
p_1	(123.3191615501, 175.7014459270)	p_5	(420.0869708340, 167.6376763887)
p_2	(178.000000018, 157.9999999861)	p_6	(510.0186257061, 170.4926523372)
p_3	(176.9605570407,185.5452384224)	p_7	(589.0000000034, 172.9999999927)
p_4	(175.9999999835, 211.0000000093)	p_8	(609.1637118080,208.7136929370)

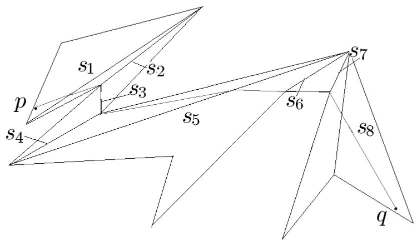

Fig. 1.14. The approximate ESP with respect to the triangulated simple polygon of Figure 1.9.

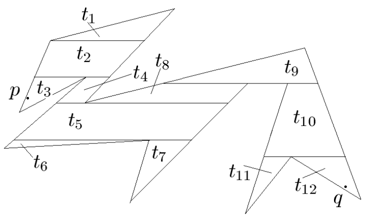

Fig. 1.15. A trapezoidal decomposition of the simple polygon of Figure 1.8.

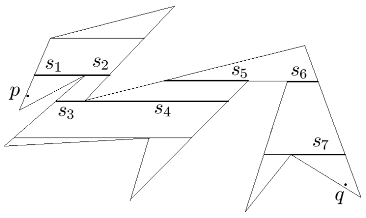

Fig. 1.16. The step set of those trapezoids in Figure 1.15.

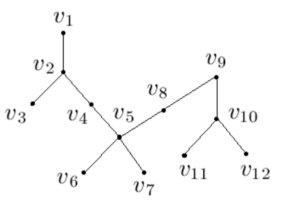

Fig. 1.17. Corresponding graph with respect to the trapezoidal decomposition in Figure 1.15.

Table 1.6. Vertices p_i calculated by Algorithm 3 for the simple polygon in Figure 1.15. The length of the path equals 1356.7016610946.

p_i	(x_i, y_i)	p_i	(x_i, y_i)
p_1	(170.9999999999, 149)	p_5	(504, 161)
p_2	(171.0000000001, 149)	p_6	(584, 161)
p_3	(171.9999999999, 202)	p_7	(669.1611374407582, 312)
p_4	(172.0000000001, 202)		

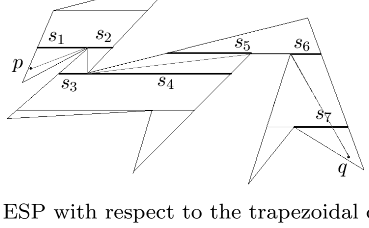

Fig. 1.18. The approximate ESP with respect to the trapezoidal decomposition in Figure 1.15.

1.4. Proofs of Correctness

Obviously, correctness of Algorithm 1 implies that of Algorithms 2 and 3. In this subsection we present two versions of proofs to show that Algorithm 1 is correct for any sequence of pairwise disjoint segments. The first one is longer but leads to a stronger result: we not only prove that the algorithm is correct but also show that the ESP is unique. The second one is very short but without proving the uniqueness of the ESP.

1.4.1. *Definitions*

We start with introducing a few definitions used in those proofs. Some of them are from mathematical analysis or multivariable calculus or from elementary topology textbook.

Definition 1.1. An *iteration* of Algorithm 1 is a complete pass through its loop. At the end of iteration $n \geq 1$ we obtain the n-th *approximate ESP*, denoted by $AESP_n(S)$, for a given sequence of segments $S = \{s_1, s_2, \ldots, s_k\}$.

We assume that the sequence of the n-th approximate ESPs is converging towards a polygonal path; let

$$AESP(S) = \lim_{n \to \infty} AESP_n(S)$$

be this polygonal path.

Let $p_i(t_{i_0})$ be the i-th vertex of the $AESP(S)$, for $i = 1, 2, \ldots, k$. Parameter $t_{i_0} \in [0, 1]$ identifies the i-th vertex of $AESP(S)$ that is on the segment s_i. Let $p_0 = p$, $p_{k+1} = q$, and $d_i = d_e(p_{i-1}, p_i) + d_e(p_i, p_{i+1})$ for $i = 1$, 2, …, or k. Let

$$d(t_1, t_2, \ldots, t_k) = \sum_{i=1}^{k} d_i$$

Obviously, $d(t_1, t_2, \ldots, t_k)$ is an k-ary function on the domain $[0, 1]^k$.

Let $p_i \in s_i$, for $i = 1, 2, \ldots, k$. We call the k tuple (p_1, p_2, …, p_k) a *point tuple* of S. We call it an *AESP critical point tuple* of S if it is the set of the vertices of the $AESP$ of S.

Now let P =(p_1, p_2, …, p_k) be an $AESP$ critical point tuple of S. Using P as an initial point set, defining $AESP_0(S)$, and n iterations of Algorithm 1, we get another critical point tuple of S, say $P' = (p'_1, p'_2, \ldots, p'_k)$, which defines (see above) the n-th approximate polygonal path $AESP_n(S)$, or $AESP_n$ for short.

Definition 1.2. Let

$$\frac{\partial d(t_1, t_2, \ldots, t_k)}{\partial t_i}|_{t_i = t_{i0}} = 0$$

where $i = 1$, 2, …, or k. Then we say that $(t_{10}, t_{20}, \ldots, t_{k0})$ is a *critical point* of

$$d(t_1, t_2, \ldots, t_k)$$

Let P =(p_1, p_2, …, p_k) be a critical point tuple of S. Using P as an initial point set, n iterations of the Algorithm 1, we calculate an *n-rubberband transform* of P, denoted by $P \rightarrow_{rb_n} Q$, or $P \rightarrow Q$ for short, where Q is the resulting critical point tuple of S, and n is a positive integer.

Let P =(p_1, p_2, …, p_k) be a critical point tuple of S. For sufficiently small real $\varepsilon > 0$, the set

$$\begin{aligned}\{(p'_1, p'_2, \ldots, p'_k) : \ & x'_i \in (x_i - \varepsilon, x_i + \varepsilon) \ \wedge \ y'_i \in (y_i - \varepsilon, y_i + \varepsilon) \\ & \wedge\, p'_i = (x'_i, y'_i) \\ & \wedge\, p_i = (x_i, y_i) \ \wedge \ i = 1, 2, \ldots, k\}\end{aligned}$$

is the *ε-neighborhood* of P, denoted by $U_\varepsilon(P)$.

The ε-neighborhood of P is an open set in the Euclidean k-dimensional topological space $(\mathbb{R}^k, T)$; T is the topology, that means the family of all open sets in $\mathbb{R}^k$. We also use the following definition (see Definition 4.1[15]):

Definition 1.3. Let $Y \subset X$, where (X, T) is a topological space. Let T' be the family of sets defined as follows: A set W belongs to T' iff there is a set $U \in T$ such that W $= Y \cap U$. The family T' is called *the relativization of T to Y*, denoted by $T|_Y$.

1.4.2. *A Proof Without Using Convex Analysis*

We express a point

$$p_i(t_i) = (x_i + k_{x_i} t_i, y_i + k_{y_i} t_i)$$

on s_i in general form, with $t_i \in [0, 1]$, where i = 1, 2, …, or k. In the following, $p_i(t_i)$ will also be denoted by p_i for short, where i = 1, 2, …, or k.

The following is a multivariable version of Fermat's Theorem in mathematical analysis (see Theorem 8.8.1[5]). We will use it for proving Lemma 1.3; this lemma is then applied in the proofs of Lemmas 1.4 and Theorem 1.2.

Theorem 1.1. (Fermat's Theorem) *Let f = f($t_1, t_2, \ldots, t_k$) be a real-valued function defined on an open set U in $\mathbb{R}^k$. Let C = ($t_{10}, t_{20}, \ldots, t_{k0}$) be a point of U. Suppose that f is differentiable at C. If f has a local extremum at C, then*

$$\frac{\partial f}{\partial t_i} = 0$$

where i = 1, 2, …, k.

Let $p_i(t_{i_0})$ be i-th vertex of an $AESP$, where $i = 1, 2, \ldots, k$. Then we have the following:

Lemma 1.3. $(t_{10}, t_{20}, \ldots, t_{k0})$ *is a critical point of* $d(t_1, t_2, \ldots, t_k)$.

Proof. $d(t_1, t_2, \ldots, t_k)$ is differentiable at each point

$$(t_1, t_2, \ldots, t_k) \in [0, 1]^k$$

Because $AESP_n(S)$ is the n-th polygonal path of S, where $n = 1, 2, \ldots$, and

$$AESP = \lim_{n \to \infty} AESP_n(S)$$

it follows that $d(t_{1_0}, t_{2_0}, \ldots, t_{k_0})$ is a local minimum of $d(t_1, t_2, \ldots, t_k)$. By Theorem 1.1,

$$\frac{\partial d}{\partial t_i} = 0$$

where $i = 1, 2, \ldots, k$. By Definition 1.2, $(t_{10}, t_{20}, \ldots, t_{k0})$ is a critical point of $d(t_1, t_2, \ldots, t_k)$. $\square$

By Lemmas 1.2 and 1.3, we have the following:

Lemma 1.4. *Any sequence S of pairwise disjoint segments has only a finite number of $AESP$ critical point tuples.*

This is our first important lemma in this subsection. In the rest of this subsection, based on Lemma 1.4, we show a much stronger result: S has actually only one (!) $AESP$ critical point tuple.

Let $p_i = (p_{i_1}, p_{i_2})$ be on s_i, for $i = 1, 2, 3$. The proof of the following lemma specifies an explicit expression for the relation between parameter t and the optimum point p_2.

Lemma 1.5. *Optimum point $p_2 \in s_2$, defined by*

$$d_e(p_2, p_1) + d_e(p_2, p_3) = \min\{p'_2 : d_e(p'_2, p_1) + d_e(p'_2, p_3) \wedge p'_2 \in s_2\}$$

can be computed in $\mathcal{O}(1)$ time.

Proof. Let the two endpoints of s_2 be $a_2 = (a_{2_1}, a_{2_2})$ and $b_2 = (b_{2_1}, b_{2_2})$. Let $p_1 = (p_{1_1}, p_{1_2})$. Point p_2 can be written as

$$(a_{2_1} + (b_{2_1} - a_{2_1})t, a_{2_2} + (b_{2_2} - a_{2_2})t)$$

The formula

$$d_e(p_2, p_1) = \sqrt{\sum_{i=1}^{2} [(a_{2_i} - p_{1_i}) + (b_{2_i} - a_{2_i})t]^2}$$

can be simplified: We can rotate the coordinate system such that s_2 is parallel to one of the two coordinate axes. It follows that only one element of the set

$$\{b_{2_i} - a_{2_i} : i = 1, 2\}$$

is equal to a real number $\alpha \neq 0$, and the other is equal to 0. Without loss of generality we can assume that

$$d_e(p_2, p_1) = \sqrt{(\alpha t + A_1)^2 + B_1}$$

where A_1 and B_1 are functions of a_{2_i}, b_{2_i} and p_{1_i}, for $i = 1, 2$. – Analogously,

$$d_e(p_2, p_3) = \sqrt{(\alpha t + A_2)^2 + B_2}$$

where A_2 and B_2 are functions of a_{2_i}, b_{2_i} and p_{3_i}, for $i = 1, 2$. In order to find a point $p_2 \in s_2$ such that

$$d_e(p_2, p_1) + d_e(p_2, p_3) = \min\{p'_2 : d_e(p'_2, p_1) + d_e(p'_2, p_3), p_2 \in s_2\}$$

we can solve the equation

$$\frac{\partial(d_e(p_2, p_1) + d_e(p_2, p_3))}{\partial t} = 0$$

The unique solution is

$$t = -1/\alpha \times (A_1B_2 + A_2B_1)/(B_2 + B_1)$$

This proves the lemma. □

By the proof of Lemma 1.5, assuming the representation

$$p_i = (a_{i_1} + (b_{i_1} - a_{i_1})t_i, a_{i_2} + (b_{i_2} - a_{i_2})t_i)$$

we have defined a function f, $t_2 = f(t_1, t_3)$, for which we have the following:

Lemma 1.6. *The function $t_2 = f(t_1, t_3)$ is continuous at each tuple $(t_1, t_3) \in [0, 1]^2$.*

This is used to prove the following:

Lemma 1.7. *If $P \rightarrow_{rb_1} Q$, then for every sufficiently small real $\varepsilon > 0$, there is a sufficiently small real $\delta > 0$ such that $P' \in U_\delta(P)$ and $P' \rightarrow_{rb_1} Q'$ implies $Q' \in U_\varepsilon(Q)$.*

Proof. By Lemma 1.5 and note that S has k segments; thus we use Lemma 1.6 repeatedly k times, and this proves this lemma. □

By Lemma 1.7, we have the following:

Lemma 1.8. *If $P \rightarrow_{rb_n} Q$, then, for every sufficiently small real $\varepsilon > 0$, there is a sufficiently small real $\delta_\varepsilon > 0$ and a sufficiently large integer N_ε, such that $P' \in U_{\delta_\varepsilon}(P)$ and $P' \rightarrow_{rb_{n'}} Q'$ implies $Q' \in U_\varepsilon(Q)$, where n' is an integer and $n' > N_\varepsilon$.*

This lemma is used to prove Lemma 1.12; the latter one and the following three lemmas are then finally applied to prove the second important lemma (i.e., Lemma 1.13) in this section. Lemmas 1.13 and 1.3 imply then the main theorem (i.e., Theorem 1.2 below) of this section.

By Lemma 1.4, let Q_1, Q_2, …, Q_N with $N \geq 1$ be the set of all $AESP$ critical point tuples of S. Let ε be a sufficiently small positive real such that

$$U_\varepsilon(Q_i) \cap U_\varepsilon(Q_j) = \emptyset$$

for $i, j = 1, 2, \ldots, N$ and $i \neq j$. Let

$$D_i = \{P : P \to Q' \wedge Q' \in U_\varepsilon(Q_i) \wedge P \in [0,1]^k\}$$

for $i = 1, 2, \ldots, N$.

The statements in the following two lemmas are obvious:

Lemma 1.9. *If $N > 1$ then $D_i \cap D_j = \emptyset$, for $i, j = 1, 2, \ldots, N$ and $i \neq j$.*

Lemma 1.10. $\bigcup_{i=1}^{N} D_i = [0,1]^k$

We consider the Euclidean topology T on $\mathbb{R}^k$, and its relativization $T = \mathbb{R}^k|_{[0,1]^k}$.

Lemma 1.11. *D_i is an open set of T, where $i = 1, 2, \ldots, N$ with $N \geq 1$.*

Proof. By Lemma 1.8, for each $P \in D_i$, there is a sufficiently small real $\delta_P > 0$ such that

$$U_{\delta_P}(P) \subseteq D_i$$

So we have

$$\bigcup_{P \in D_i} U_{\delta_P}(P) \subseteq D_i$$

On the other hand, for $P \in U_{\delta_P}(P)$, we have

$$D_i = \cup P \subseteq \bigcup_{P \in D_i} U_{\delta_P}(P)$$

Note that $U_{\delta_P}(P)$ is an open set of T. Thus,

$$D_i = \bigcup_{P \in D_i} U_{\delta_P}(P)$$

is an open set of T. □

The following basic lemma is characterizing open sets in general (see Proposition 5.1.4[20]).

Lemma 1.12. *Let $U \subset \mathbb{R}$ be an arbitrary open set. Then there are countably many pairwise disjoint open intervals U_n such that $U = \cup U_n$.*

Now we are prepared to approach the second important lemma in this subsection:

Lemma 1.13. *S has a unique $AESP$ critical point tuple.*

Proof. By contradiction. Suppose that Q_1, Q_2, …, Q_N with $N > 1$ are all the $AESP$ critical point tuples of S. Then there exists $i \in \{1, 2, \ldots, N\}$ such that

$$D_i|_{s_j} \subset [0, 1]$$

where s_j is a segment in S, for $i, j = 1, 2, \ldots, N$. Otherwise we have

$$D_1 = D_2 = \cdots = D_N$$

This is a contradiction to Lemma 1.9.

Let

$$E = \{s_j : D_i|_{s_j} \subseteq [0, 1]\}$$

where s_j is any segment in S. We can select a critical point tuple of S as follows: go through each $s \in \{s_1, s_2, \ldots, s_k\}$. If $s \in E$, by Lemmas 1.11 and 1.12, select the minimum left endpoint of the open intervals whose union is $D_i|_s$. Otherwise select the midpoint of s. We denote the resulting critical point tuple as

$$P = (p_1, p_2, \ldots, p_k)$$

By the selection of P, we know that P is not in D_i. By Lemma 1.10 there is a $j \in \{1, 2, \ldots, N\} - \{i\}$ such that $P \in D_j$. Therefore, there is a sufficiently small real $\delta > 0$ such that $U_\delta(P) \subset D_j$. Again by the selection of P, there is a sufficiently small real $\delta' > 0$ such that $U'_\delta(P) \cap D_i \neq \emptyset$. Let $\delta'' = \min\{\delta, \delta'\}$. Then we have $U''_\delta(P) \subset D_j$ and $U''_\delta(P) \cap D_i \neq \emptyset$. This implies that $D_i \cap D_j \neq \emptyset$, and this is a contradiction to Lemma 1.9. □

Let S be a sequence of pairwise disjoint segments. Let $AESP_n(S)$ be the n-th approximate polygonal path of S, for $n = 1, 2, \ldots$. The subsection has shown that

$$AESP = \lim_{n \to \infty} AESP_n(S)$$

exists, and we can conclude the following main result of this section:

Theorem 1.2. *The $AESP$ of S is the ESP of S, or, in short $AESP = ESP$.*

Proof. By Lemma 1.13 and the proof of Lemma 1.3, $d(t_1, t_2, \ldots, t_k)$ has a unique local minimal value. This implies that the $AESP$ of S is the ESP of S. □

1.4.3. *A Shorter Proof by Using Convex Analysis*

This subsection gives a shorter proof of the correctness of Algorithm 1 by applying some basic results from convex analysis (but without obtaining the uniqueness result for the ESP). We cite a few basic results of convex analysis:[1,16,17]

Proposition 1.3. *(page 27[1]) Each line segment is a convex set.*

Proposition 1.4. *(page 72[1]) Each norm on $\mathbb{R}^n$ is a convex function.*

Proposition 1.5. *(page 79[1]) A nonnegative weighted sum of convex functions is a convex function.*

Theorem 1.3. *(Theorem 3.5[17]) Let S_1 and S_2 be convex sets in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively. Then*

$$\{(x, y) : x \in S_1 \wedge y \in S_2\}$$

is a convex set in $\mathbb{R}^{m+n}$, where m, $n \in \mathbb{N}$.

Proposition 1.6. *(page 264[17]) Let f be a convex function. If x is a point where f has a finite local minimum, then x is a point where f has its global minimum.*

By Proposition 1.3, the interval [0, 1] is a convex set. By Theorem 1.3, $[0, 1]^k$ is a convex set. For any p, $q \in \mathbb{R}^n$, $d_e(p, q)$ is a norm (see, for example, page 78[9]). By Proposition 1.4 and 1.5, $d(t_1, t_2, \ldots, t_k)$ (see Section 1.4.2) is a convex function on $[0, 1]^k$. Since $d(t_1, t_2, \ldots, t_k)$ is continuous on $[0, 1]^k$, so its minimum is attained. It is clear that, for any sequence of pairwise disjoint segments S, Algorithm 1 will always produce an exact local minimum of the function $d(t_1, t_2, \ldots, t_k)$. By Proposition 1.6, each local minimum of $d(t_1, t_2, \ldots, t_k)$ is its global minimum. Therefore, we have proved Theorem 1.2 once again.

To recall, we proved that Algorithm 1 is correct for the family of sequences of pairwise disjoint segments, and that there is unique ESP for a given sequence of pairwise disjoint segments.

Although the proof in Subsection 1.4.2 is much more complicated than the one in Subsection 1.4.3, we proved a stronger result there, namely, that for each sequence of pairwise disjoint segments, Algorithm 1 will converge to a unique ESP.

See also Lemma 1,[4] Lemma 3.3,[18] and Lemma 1,[21] for proofs of the uniqueness of an ESP. Our proof is actually also completely suitable for the "curve case", where $p = q$.

1.5. Computational Complexity

This section analyzes the time complexities of Algorithms 1, 2 and 3.

Theorem 1.4. *Algorithm 1 has a time complexity in $\kappa(\varepsilon) \cdot \mathcal{O}(k)$ time, where $\kappa(\varepsilon) = (L - L_0)/\varepsilon$, L is the true length of the ESP of S, L_0 that of an initial polygonal path, and k is the number of segments of the set S.*

Proof. Let L_n be the true length of the polygonal path after n iterations. We slightly modify Algorithm 1 as follows:[a]

For each iteration, we update the vertices with odd indices first and then update those with even indices later (i.e., for each iteration, we update the following vertices p_1, p_3, p_5, ..., then the following vertices p_2, p_4, p_6, ...

Thus, $\{L_n\}_{n\to\infty}$ is a strict decreasing sequence with lower bound 0, since $L_0 - L$ can be written as $ak + b$ (i.e., it is a linear function of k), where a, b are constants such that $a \neq 0$. Because Algorithm 1 will not stop if $L_n - L_{n+1} > \varepsilon$ (see Step 8, Algorithm 1), it follows that $L_n - L_{n+1}$ will also depend on k. Again, since $L_n - L_{n+1}$ can be written as $ck + d$, where c and d are constants such that $c \neq 0$. Then we have that

$$\lim_{k\to\infty} \frac{ak+b}{ck+d} = \frac{a}{c}$$

Therefore, Algorithm 1 will stop after at most $\lceil a/(c\varepsilon)\rceil$ iterations. (Note that, if we would not modify Algorithm 1, then it stops after at most

$$\lceil (L_0 - L)/\varepsilon \rceil$$

iterations.)

We apply Lemma 1.5; it follows that the time complexity of Algorithm 1 is in $\lceil (L_0 - L)/(\varepsilon)\rceil \cdot \mathcal{O}(k) = \kappa(\varepsilon)\cdot\mathcal{O}(k)$, where $\kappa(\varepsilon) = (L - L_0)/\varepsilon$, L be the true length of the ESP of S, L_0 that of an initial polygonal path, and k is the number of segments of the set S. □

Lemma 1.14. *Procedure 1 can be computed in $\mathcal{O}(n)$ time, where $n = |V(T)|$.*

Proof. Step 1 can be computed in $\mathcal{O}(|V(T)|)$ time, where T is the original input tree. Steps 2 and 3 can be computed in $\mathcal{O}(1)$ time. Steps 4.2.1–4.2.3 can be computed in $\mathcal{O}(1)$ time. So each while loop can be computed in $\mathcal{O}(|V_{i_j}|)$ time. Steps 4.1 and 4.3 can be computed in $\mathcal{O}(1)$ time. Step 4.4 can be computed in $\mathcal{O}(|V_{i_j}|)$ time. So each for loop can be computed in $\mathcal{O}(\sum_{j=1}^{|S_i|}|V_{i_j}|) \leq |V(T)|$ time, where T is the original input tree. Therefore each iteration (Steps 1–5) can be computed in $\mathcal{O}(|V(T)|)$ time, where T is the original input tree. Since Step 5 can occur at most once, Procedure 1 can be computed in $\mathcal{O}(|V(T)|)$ time, where T is the original input tree. □

Theorem 1.5. *Algorithms 2 and 3 can be computed in $\kappa(\varepsilon)\cdot\mathcal{O}(k)$ or $\kappa(\varepsilon)\cdot\mathcal{O}(k\log k)$ time, respectively, where $\kappa(\varepsilon) = (L - L_0)/\varepsilon$, L be the true length of the ESP from p to q, L_0 that of an initial polygonal path from p to q, and k is the number of vertices of Π.*

Proof. Steps 2, 4 can be computed in $\mathcal{O}(k)$ time. Theorem 4.3[3] (Theorem 1[13]), Lemma 1.14 and Theorem 1.4 prove the conclusion for Algorithm 2 (3). □

[a]This is just for the purpose of time complexity analysis. By experience, Algorithm 1 runs faster without such a modification.

1.6. Improving the Algorithms

We mention some possible ways to improve Algorithms 1, 2 and 3 without changing their time complexity.

For Algorithm 1, to initialize, let p_i be the center of s_i; let a_i and b_i be the endpoints of s_i, l_i the line such that $s_i \subset l_i$, for $i = 1, 2, \ldots, k$. We modify Steps 4.2 and 5.2 as follows:

4.2.1. Let $q'_2 = l_i \cap q_1q_3$.
4.2.2. If $q'_2 \in s_i$ then let $q_2 = q'_2$. Otherwise, let $q_2 \in s_i$ such that
$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \{a_i, b_i\} \cap V(\Pi)\}.$$

and

5.2.1. Let $q'_2 = l_k \cap q_1q_3$.
5.2.2. If $q'_2 \in s_k$ then let $q_2 = q'_2$. Otherwise, let $q_2 \in s_k$ such that
$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \{a_k, b_k\} \cap V(\Pi)\}.$$

For Algorithms 2 and 3, we may revert the removal of some segments to obtain exact vertices of Π. Furthermore, for Algorithm 3, we may prove that each possible intersection point of e_i and e_{i+1} (see the discussions after Lemma 1.2) is a vertex of the ESP and thus we do not have to remove any small segment in this case.

1.7. Conclusions

This chapter provided two approximate algorithms for calculating ESPs in simple polygons. Depending on the used preprocessing step (triangular or trapezoidal decomposition), they are either $\kappa(\varepsilon)$-linear or in $\kappa(\varepsilon) \cdot \mathcal{O}(n \log n)$ time. But note that the trapezoidal decomposition algorithm[13] is substantially simpler to implement than the triangulation algorithm.[3] The chapter illustrates that rubberband algorithms are of simple design, easy to implement, and can be used to solve ESP problems either approximate or in a de facto exact way if the used accuracy threshold was chosen small enough.

References

1. S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, UK, 2004.
2. T. Bülow and R. Klette. Digital curves in 3D space and a linear-time length estimation algorithm. *IEEE Trans. Pattern Analysis Machine Intelligence*, **24**:962–970, 2002.
3. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Computational Geometry*, **6**:485–524, 1991.
4. J. Choi, J. Sellen, and C.-K. Yap. Precision-sensitive Euclidean shortest path in 3-space. In Proc. *Annu. ACM Sympos. Computational Geometry*, pages 350–359, 1995.
5. S. A. Douglass. *Introduction to Mathematical Analysis.* Addison-Wesley, 1996.
6. L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algo-

rithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, **2**:209–233, 1987.

7. L. Guibas, J. Hershberger Optimal shortest path queries in a simple polygon. *J. Computer System Sciences* **39**:126–152, 1989.
8. J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, **38**:231-235, 1991.
9. R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis.* Morgan Kaufmann, San Francisco, 2004.
10. D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks* **14**:393–410, 1984.
11. F. Li and R. Klette. Exact and approximate algorithms for the calculation of shortest paths. Report 2141 on *www.ima.umn.edu/preprints/oct2006*, 2006.
12. F. Li and R. Klette. Rubberband algorithms for solving various 2D or 3D shortest path problems. In Proc. “Computing: Theory and Applications”, Platinum Jubilee Conference of The Indian Statistical Institute, pages 9 - 18, IEEE, 2007.
13. F. Li and R. Klette. Decomposing a simple polygon into trapezoids. In Proc. *Computer Analysis Images Patterns*, Vienna, Springer, Berlin, 2007.
14. J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack, J. Urrutia, eds, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers, 2000.
15. T. O. Moore. *Elementary General Topology.* Prentice-Hall, Englewood Cliffs, N.J., 1964.
16. A. W. Roberts and V. D. Varberg. *Convex Functions.* Academic Press, New York, 1973.
17. R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, N.J., 1970.
18. M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, **15**:193–215, 1986.
19. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, **3**:362–394, 1999.
20. B. G. Wachsmuth. *Interactive real analysis.* See http://www.shu.edu/projects/reals/index.html, 2007.
21. C.-K. Yap. Towards exact geometric computation. *Computational Geometry: Theory Applications.*, **7**:3–23, 1997.